An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau   Ben Moszkowski   David Smallwood

April 16, 2020

Abstract

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

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

Contents

1 Finite Intervals

theory *Interval*
 imports
  Main
begin

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

The usual operations on intervals are defined: length (*intlen*), *prefix*, *suffix*, *sub*, *nth*, *intfirst*, *intlast*, *intapp* and *intrev*.

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

1.1 Definitions

datatype (*set*: *'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 ⟨*x*⟩ = *0*
 | *intlen* (*x*⊙*xs*) = *1*+ (*intlen xs*)

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

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

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

definition *sub*:: *nat* ⇒ *nat* ⇒ *'a interval* ⇒ *'a interval*
where
sub n k xs = *prefix* (*k*−*n*) (*suffix n xs*)

definition *intfirst* :: *'a interval* ⇒ *'a* **where**
intfirst xs = (*nth xs 0*)

definition *intlast* :: *'a interval* ⇒ *'a* **where**
intlast xs = (*nth xs* (*intlen xs*))

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

primrec *intrev* :: *'a interval* ⇒ *'a interval* **where**
intrev ⟨*x*⟩ = ⟨*x*⟩
| *intrev* (*x* ⊙*xs*) = (*intrev xs*) ⊖ ⟨*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*. *x*−*k*)

primrec *upt* :: *nat* ⇒ *nat* ⇒ *nat interval* ((*1*[-..≤/-']))
where
upt-0 : [*i*..≤*0*] = ⟨*0*⟩
| *upt-Suc*: [*i*..≤(*Suc j*)] = (*if i* ≤ *j then* [*i*..≤*j*] ⊖ ⟨(*Suc j*)⟩ *else* ⟨(*Suc j*)⟩)

1.2 Lemmas

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

1.2.1 Shifting indexes to zero

lemma *interval-shift-index-to-zero-a*:
shows (∀ (*i*::*nat*). *a* ≤ *i* ∧ *i* < *a*+*b* ⟶ *f* (*g1*(*i*)) (*g2*(*Suc i*))) =
(∀ *i*. *0* ≤ *i* ∧ *i* < *b* ⟶ *f* (*g1*(*i*+*a*)) (*g2*((*Suc i*)+*a*)))

by (*metis add-diff-cancel-left' le0 le-add2 le-add-diff-inverse2 less-diff-conv plus-nat.simps*(*2*))

lemma *interval-shift-index-to-zero-b*:
shows ($\forall$ (*i*::*nat*). $a \le i \wedge i < a+b \longrightarrow f$ ($g(i-a)$) ($g2((Suc\ i)-a)$)) =
($\forall$ *i*. $0 \le i \wedge i < b \longrightarrow f$ ($g(i)$) ($g2(Suc\ i)$)) (**is** *?L=?R*)
proof −
have *1*: *?L* $\Longrightarrow$ *?R*
by (*metis add-Suc add-diff-cancel-left' add-diff-cancel-right' le-add2 less-diff-conv*)
have *2*: *?R* $\Longrightarrow$ *?L*
by (*metis Nat.add-diff-assoc add-diff-cancel-left' le0 le-add-diff-inverse2 less-diff-conv plus-1-eq-Suc*)
show *?thesis* **using** *1 2* **by** *blast*
qed

1.2.2 Interval Length

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

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

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

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

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-all*

1.2.3 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*:
$\quad$ *nth* ($x \odot xs$) (*intlen* ($x \odot xs$)) = *nth xs* (*intlen xs*)
by *simp*

lemma *interval-nth-last-stutter*:
nth xs (*intlen xs* + *i*) = *nth xs* (*intlen xs*)
proof (*induction xs arbitrary*:*i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
  proof (*cases i*)
  case *0*
  then show *?thesis* **by** *simp*
  next
  case (*Suc nat*)
  then show *?thesis*
  by (*metis Cons.IH ab-semigroup-add-class.add-ac*(*1*) *add-Suc-right interval-intlen-cons interval-nth-Suc interval-nth-last*)
  qed
qed

lemma *interval-nth-cons-a*:
assumes $0<i$
shows *nth*($x \odot xs$) *i* = *nth xs* ($i-1$)
using *assms* **by** (*metis Suc-diff-1 interval-nth-Suc*)

lemma *interval-nth-cons-b*:
shows *nth*($x \odot xs$) ($i+1$) = *nth xs i*
by *simp*

lemma *interval-nth-cons*:
assumes $0<i$
shows *nth*($x \odot xs$) *i* = *nth xs* ($i-1$) $\wedge$
nth($x \odot xs$) ($i+1$) = *nth xs i*
by (*meson assms interval-nth-cons-a interval-nth-cons-b*)

lemma *interval-nth-zero-intfirst* [*simp*]:
intfirst xs = *nth xs 0*
by (*simp add*: *intfirst-def*)

lemma *interval-nth-intlen-intlast* [*simp*]:
intlast xs = *nth xs* (*intlen xs*)
by (*simp add*: *intlast-def*)

lemma *interval-st-intlen* :
$\quad$ ($xs = \langle x \rangle$) $\longleftrightarrow$ *intlen xs* = 0 $\wedge$ *nth xs 0* = *x*
by (*cases xs*) *auto*

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

lemma *interval-nth-map* :
$nth\ (map\ f\ xs)\ i = f\ (nth\ xs\ i)$
proof
(*induct xs arbitrary*: *i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases i*)
case *0*
then show *?thesis* **by** *simp*
next
case (*Suc nat*)
then show *?thesis* **using** *Cons.hyps* **by** *simp*
qed
qed

1.2.4 prefix, suffix and sub

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

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

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

by *auto*

lemma *interval-prefix-zero-intfirst* [*simp*]:
  prefix 0 xs = ⟨*nth xs 0*⟩
by (*induct xs*) *simp-all*

lemma *interval-intfirst-prefix* [*simp*]:
 shows *intfirst* (*prefix i xs*) = *intfirst xs*
proof
 (*induct xs arbitrary*: *i*)
 case (*St x*)
 then show *?case* **by** *auto*
 next
 case (*Cons x1a xs*)
 then show *?case*
  proof (*cases i*)
  case *0*
  then show *?thesis* **by** *auto*
  next
  case (*Suc nat*)
  then show *?thesis* **using** *Cons.hyps* **by** *auto*
  qed
 qed

lemma *interval-intlast-suffix* [*simp*]:
 shows *intlast* (*suffix i xs*) = *intlast xs*
proof
 (*induct xs arbitrary*: *i*)
 case (*St x*)
 then show *?case* **by** *auto*
 next
 case (*Cons x1a xs*)
 then show *?case*
  proof (*cases i*)
  case *0*
  then show *?thesis* **by** *auto*
  next
  case (*Suc nat*)
  then show *?thesis* **using** *Cons.hyps* **by** *auto*
  qed
 qed

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-code* [*code*]:
    intlen (*prefix i xs*) = (*if i*≤ *intlen xs then i else intlen xs*)
proof
 (*induct xs arbitrary*: *i*)
 case (*St x*)
 then show *?case* **by** *simp*
 next
 case (*Cons x1a xs*)
 then show *?case*
  proof (*cases i*)
  case *0*
  then show *?thesis* **by** *auto*
  next
  case (*Suc nat*)
  then show *?thesis* **using** *Cons.hyps* **by** *auto*
  qed
 qed

lemma *interval-prefix-length* [*simp*]:
    intlen (*prefix i xs*) = *min i* (*intlen xs*)
by (*simp add*: *interval-prefix-length-code min-def*)

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

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

lemma *interval-pref-intlen-bound* :
 shows *intlen* (*prefix i xs*) ≤ *intlen xs*
by *simp*

lemma *interval-suffix-length-code* [*code*]:
    intlen (*suffix i xs*) = (*if i*≤ *intlen xs then* (*intlen xs*)−*i else 0*)
proof
 (*induct xs arbitrary*: *i*)
 case (*St x*)
 then show *?case* **by** *simp*
 next
 case (*Cons x1a xs*)

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

lemma interval-suffix-length [simp]:
  intlen (suffix i xs ) = (intlen xs)−i
by (simp add: interval-suffix-length-code)

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

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

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

lemma interval-nth-prefix [simp]:
 assumes   k ≤ i
 shows     nth (prefix i xs) k = nth xs k
using assms
 proof
  (induct i arbitrary: xs k)
  case 0
  then show ?case
    proof (cases xs)
    case (St x1)
    then show ?thesis by auto
    next
    case (Cons x21 x22)
    then show ?thesis using 0.prems by auto
    qed
  next
  case (Suc i)
  then show ?case
    proof (cases xs)
    case (St x1)
    then show ?thesis by auto
    next
```

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

lemma interval-nth-suffix [simp]:
 assumes  k ≤ intlen xs −i
 shows    nth (suffix i xs) k = nth xs (i+k)
using assms
proof (induct xs arbitrary: i k)
case (St x)
then show ?case by simp
next
case (Cons x1a xs)
then show ?case
  proof (cases i)
  case 0
  then show ?thesis by auto
  next
  case (Suc nat)
  then show ?thesis
    proof auto
      assume a0:  i = Suc nat
      show Interval.nth (suffix nat xs) k = Interval.nth xs (nat + k)
      using a0 Cons.hyps Cons.prems by auto
    qed
  qed
qed

lemma interval-suffix-prefix-help-1:
 assumes  ia+i ≤ intlen xs
          k ≤ ia
 shows  nth (prefix ia (suffix i xs)) k = nth (suffix i (prefix (ia +i) xs)) k
proof −
 have 1:  nth (prefix ia (suffix i xs)) k = nth (suffix i xs) k
 using interval-nth-prefix assms by metis
 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 $prefix\ ia\ (suffix\ i\ xs) = suffix\ i\ (prefix\ (ia\ +i)\ xs)$
using *assms* **using** *interval-eq-nth-eq* **by** *fastforce*

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-good less-imp-add-positive not-less*)

lemma *interval-pref-pref-3* [*simp*]:
$(prefix\ i\ (prefix\ (i+k)\ xs)) = prefix\ i\ xs$
proof
(*induct xs arbitrary*: *i k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases i*)
case *0*
then show *?thesis*
by (*auto simp add*: *Nitpick.case-nat-unfold*)
next
case (*Suc nat*)
then show *?thesis*
by (*simp add*: *Cons.hyps*)
qed
qed

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-good ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)

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

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

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

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

lemma *interval-hd-tail*:
assumes *intlen xs* >*0*
shows *xs* = (*intfirst xs*) ⊙ (*suffix 1 xs*)
by (*metis One-nat-def assms interval-intlen-cons-1 interval-nth-zero interval-nth-zero-intfirst interval-suffix-suc interval-suffix-zero*)

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

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

lemma *interval-suffix-suffix* [*simp*]:
suffix i (*suffix j xs*) = *suffix* (*i*+*j*) *xs*
proof
(*induct xs arbitrary*: *i j*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*

```
  proof (cases i)
  case 0
  then show ?thesis
    by auto
  next
  case (Suc nat)
  then show ?thesis
    by (simp add: Nitpick.case-nat-unfold add.commute local.Cons)
  qed
qed

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

lemma interval-prefix-suffix-intlen [simp]:
    intlen (prefix ia (suffix i xs)) =
     min ia (intlen xs −i)

by auto

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 auto

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

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

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 $i < j$
  $j \leq (intlen\ xs) - k$
  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 $0 \leq i$
  $ia+i \leq intlen\ xs$
  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 $0 \leq i$
  $i \leq j$
  $j \leq intlen\ xs$
  shows (*sub i j xs*) = (*prefix* $(j-i)$ (*suffix i xs*))
using *assms* **by** (*simp add*: *Interval.sub-def*)

lemma *interval-intlast-prefix*:
  assumes $k \leq intlen\ xs$
  shows *intlast*(*prefix k xs*) = (*nth xs k*)
using *assms interval-prefix-length-good* **by** *fastforce*

lemma *interval-intfirst-suffix*:
  assumes $k \leq intlen\ xs$
  shows *intfirst*(*suffix k xs*) = (*nth xs k*)
by (*simp add*: *assms*)

lemma *interval-suffix-gr*:
  assumes $i > intlen\ xs$
  shows *suffix i xs* = $\langle$*intlast*(*xs*)$\rangle$
  by (*metis add.commute assms interval-suffix-intlast interval-suffix-suffix less-imp-add-positive suffix.simps*(*1*))

lemma *interval-intlast-intfirst*:
  (*intlast* (*prefix i xs*)) = (*intfirst* (*suffix i xs*))
proof −
  have *1*: (*intlast* (*prefix i xs*)) = (*nth* (*prefix i xs*) (*intlen* (*prefix i xs*)))
    by *simp*
  have *2*: $i \leq intlen\ xs \longrightarrow$ *intlen* (*prefix i xs*) = *i*
    using *interval-prefix-length-good* **by** *blast*
  have *3*: $i > intlen\ xs \longrightarrow$ *intlen* (*prefix i xs*) = *intlen xs*
    using *interval-prefix-length-bad* **by** *blast*
  have *4*: $i \leq intlen\ xs \longrightarrow$
    (*nth* (*prefix i xs*) (*intlen* (*prefix i xs*))) = (*nth xs i*)
    using *interval-intlast-prefix* **by** *auto*
  have *5*: $i > intlen\ xs \longrightarrow$

```
          (nth (prefix i xs) (intlen (prefix i xs))) = (nth xs (intlen xs))
    using 3 by auto
  have 6: (intfirst (suffix i xs)) = (nth (suffix i xs) 0)
      by simp
  have 7: i ≤ intlen xs ⟶
          (nth (suffix i xs) 0) = (nth xs i)
    by simp
  have 8: i>intlen xs ⟶
          (nth (suffix i xs) 0) = (nth xs (intlen xs))
     by (simp add: interval-suffix-gr)
  show ?thesis using 4 5 8 by auto
qed


lemma interval-intlen-sub [simp]:
 assumes k≤ n
         n ≤ intlen xs
 shows   intlen(sub k n xs ) = (n−k)
using assms
by (metis Interval.sub-def interval-prefix-length-good interval-suffix-length
    interval-suffix-prefix-swap le-add-diff-inverse2)

lemma interval-nth-sub [simp]:
 assumes k≤ n
         n ≤ intlen xs
         j≤n−k
 shows    nth(sub k n xs) j = (nth xs (k+j))
proof −
 have 1: nth(sub k n xs) j = nth (prefix (n−k) (suffix k xs)) j
   by (simp add: Interval.sub-def)
 have 2: n−k ≤ intlen (suffix k xs)
    using Interval.sub-def assms by auto
 have 3: j≤ (n−k)
    using assms by auto
 have 4: nth (prefix (n−k) (suffix k xs)) j =
          nth (suffix k xs) j
    using 2 assms interval-nth-prefix by blast
 have 5: nth (suffix k xs) j = nth xs (k+j)
       using assms by auto
 show ?thesis
 by (simp add: 1 4 5)
qed

lemma interval-intlast-sub:
 assumes k ≤ n
         n ≤ intlen xs
 shows   intlast (sub k n xs) = (nth xs n)
by (simp add: assms)

lemma interval-intfirst-sub:
```

assumes $k \leq n$
$n \leq intlen\ xs$
shows $intfirst\ (sub\ k\ n\ xs) = (nth\ xs\ k)$
by $(simp\ add: assms)$

lemma *interval-sub-sub*:
assumes $n1 \leq n2$
$n0 \leq n4$
$n2 \leq n4-n0$
$n4 \leq n3$
$n3 \leq intlen\ xs$
shows $(sub\ n1\ n2\ (sub\ n0\ n3\ xs)) = (sub\ n1\ n2\ (sub\ n0\ n4\ xs))$
proof $-$
have 1: $intlen(sub\ n0\ n3\ xs) = n3\ -n0$
by $(meson\ assms\ interval\text{-}intlen\text{-}sub\ le\text{-}trans)$
have 2: $intlen\ (sub\ n1\ n2\ (sub\ n0\ n3\ xs)) = n2\ -n1$
using *interval-intlen-sub assms* **by** *auto*
have 3: $intlen(sub\ n0\ n4\ xs) = n4\ -n0$
using *assms interval-intlen-sub le-trans* **by** *blast*
have 4: $intlen\ (sub\ n1\ n2\ (sub\ n0\ n4\ xs)) = n2\ -n1$
by $(simp\ add: 3\ assms)$
have 5: $\bigwedge i.\ i \leq (n3\ -n0) \longrightarrow (nth\ (sub\ n0\ n3\ xs)\ i\ ) = (nth\ xs\ (n0+i))$
using *assms interval-nth-sub le-trans* **by** *blast*
have 6: $\bigwedge i\ .\ i \leq (n2-n1) \longrightarrow (nth\ (sub\ n1\ n2\ (sub\ n0\ n3\ xs))\ i) = (nth\ (sub\ n0\ n3\ xs)\ (n1+i))$
using *interval-nth-sub assms*
by $(metis\ 1\ Nat.le\text{-}diff\text{-}conv2\ le\text{-}trans)$
have 7: $\bigwedge i\ .\ i \leq (n2-n1) \longrightarrow (nth\ (sub\ n1\ n2\ (sub\ n0\ n3\ xs))\ i) = (nth\ xs\ (n0+(n1+i)))$
using *5 6 assms* **by** *auto*
have 8: $n0 \leq n4 \wedge n4 \leq intlen\ xs$
using *assms le-trans* **by** *blast*
have 9: $\bigwedge i.\ i \leq (n4\ -n0) \longrightarrow (nth\ (sub\ n0\ n4\ xs)\ i\ ) = (nth\ xs\ (n0+i))$
using *8 interval-nth-sub* **by** *blast*
have 10: $\bigwedge i.\ i \leq (n2-n1) \longrightarrow (nth\ (sub\ n1\ n2\ (sub\ n0\ n4\ xs))\ i) = (nth\ (sub\ n0\ n4\ xs)\ (n1+i))$
by $(simp\ add: 3\ assms)$
have 11: $\bigwedge i.\ i \leq (n2-n1) \longrightarrow (nth\ (sub\ n1\ n2\ (sub\ n0\ n4\ xs))\ i) = (nth\ xs\ (n0\ +(n1+i)))$
by $(metis\ 3\ Nat.le\text{-}diff\text{-}conv2\ add.commute\ assms\ interval\text{-}nth\text{-}sub\ le\text{-}trans)$
have 12: $\bigwedge i.\ i \leq (n2-n1) \longrightarrow$
$(nth\ (sub\ n1\ n2\ (sub\ n0\ n3\ xs))\ i) = (nth\ (sub\ n1\ n2\ (sub\ n0\ n4\ xs))\ i)$
by $(simp\ add: 11\ 7)$
from *12 2 4* **show** *?thesis* **by** $(simp\ add: interval\text{-}eq\text{-}nth\text{-}eq)$
qed

lemma *interval-sub-sub-1*:
assumes $n1 \leq n2$
$n0 \leq n3$
$n2 \leq n3-n0$
$n3 \leq intlen\ xs$
shows $(sub\ n1\ n2\ (sub\ n0\ n3\ xs)) = (sub\ (n0+n1)\ (n0+n2)\ xs)$
proof $-$

```
 have 1: intlen(sub n1 n2 (sub n0 n3 xs)) = intlen(sub (n1+n0) (n2+n0) xs)
   using assms by auto
 have 2: ⋀i. i≤ (n2−n1) ⟶ (nth (sub n1 n2 (sub n0 n3 xs)) i) = (nth xs (n0+(n1+i)))
   by (simp add: add.commute assms le-trans ordered-cancel-comm-monoid-diff-class.le-diff-conv2)
 have 3: ⋀i. i≤ (n2−n1) ⟶ (nth (sub (n0+n1) (n0+n2) xs) i) = (nth xs (n0+(n1+i)))
     by (metis (no-types, hide-lams) add.commute add-diff-cancel-left add-mono assms(1) assms(2)
         assms(3) assms(4) interval-nth-sub le-refl le-trans
         ordered-cancel-comm-monoid-diff-class.le-diff-conv2 semiring-normalization-rules(25))
 have 4: ⋀i. i≤ (n2−n1) ⟶ (nth (sub n1 n2 (sub n0 n3 xs)) i) = (nth (sub (n0+n1) (n0+n2) xs) i)
   by (simp add: 2 3)
 show ?thesis
 by (metis 1 4 add.commute assms interval-eq-nth-eq interval-intlen-sub)
qed

lemma interval-suf-first-upto:
assumes (∃i<k. f (suffix i xs))
        k ≤ intlen xs +1
shows   (∃i<k. f (suffix i xs) ∧
          (∀j<i . ¬ (f (suffix j xs))))
using assms
proof (induct xs arbitrary:k)
case (St x)
then show ?case by auto
next
case (Cons x1a xs)
then show ?case
  proof −
   have 0: k=0 ⟶ ( ∃i<k. f (suffix i (x1a ⊙ xs)) ∧ (∀j<i. ¬ f (suffix j (x1a ⊙ xs))))
     using Cons.prems(1) by blast
   have 1: ⋀n. k = (Suc n) ⟶ (∃i< (Suc n). f (suffix i (x1a ⊙ xs)) ∧
           (∀j<i. ¬ f (suffix j (x1a ⊙ xs)))) =
           ( f (x1a ⊙ xs) ∨
           (∃i. 1≤ i ∧ i<(Suc n) ∧ f (suffix i (x1a ⊙ xs)) ∧
                (∀j<i. ¬ f (suffix j (x1a ⊙ xs)))))

         by auto
            (metis One-nat-def less-one nat.split-sels(1) not-le-imp-less)
  have 2: ⋀n. k = (Suc n) ⟶ (∃i. 1≤ i ∧ i<(Suc n) ∧ f (suffix i (x1a ⊙ xs)) ∧
                (∀j<i. ¬ f (suffix j (x1a ⊙ xs)))) =
           (∃i.  i<n  ∧ f (suffix (Suc i) (x1a ⊙ xs)) ∧
                (∀j<(Suc i). ¬ f (suffix j (x1a ⊙ xs))))

       by auto
         (metis Nitpick.case-nat-unfold Suc-le-D diff-Suc-1 less-Suc-eq-0-disj)
  have 3: ⋀n. k= (Suc n) ⟶ (∃i.  i<n  ∧ f (suffix (Suc i) (x1a ⊙ xs)) ∧
                (∀j<(Suc i). ¬ f (suffix j (x1a ⊙ xs)))) =
          (∃i.  i<n  ∧ f (suffix i xs) ∧
                ¬(f (suffix 0 (x1a⊙xs))) ∧
                (∀j. 1 ≤ j ∧ j<(Suc i) ⟶ ¬ f (suffix j (x1a ⊙ xs))))
```

```
        by (metis interval-suffix-suc le-add1 less-Suc-eq-0-disj plus-1-eq-Suc)
   have 4: ⋀n. k= (Suc n) ⟶ (∃ i.  i<n  ∧ f (suffix i xs) ∧
              ¬(f (suffix 0 (x1a⊙xs))) ∧
              (∀ j. 1 ≤ j ∧ j<(Suc i) ⟶ ¬ f (suffix j (x1a ⊙ xs)))) =
          ( ¬(f (x1a⊙xs)) ∧
            (∃ i.  i<n  ∧ f (suffix i xs) ∧
              (∀ j.  j<i ⟶ ¬ f (suffix (Suc j) (x1a ⊙ xs)))) )
           using Cons.hyps Cons.prems by (auto, auto simp add: less-Suc-eq-0-disj)
   have 5: ⋀n. k =(Suc n) ⟶ ( ¬(f (x1a⊙xs)) ∧
            (∃ i.  i<n  ∧ f (suffix i xs) ∧
              (∀ j.  j<i ⟶ ¬ f (suffix (Suc j) (x1a ⊙ xs)))) ) =
           ( ¬(f (x1a⊙xs)) ∧
            (∃ i<n.   f (suffix i xs) ∧
              (∀ j<i . ¬ f (suffix j  xs))) )
      by auto
   have 6: ⋀n. k =(Suc n) ⟶  (∃ i< (Suc n). f (suffix i (x1a ⊙ xs)) ∧
          (∀ j<i. ¬ f (suffix j (x1a ⊙ xs)))) =
          ( f (x1a ⊙ xs) ∨ ( ¬(f (x1a⊙xs)) ∧
            (∃ i<n.   f (suffix i xs) ∧
              (∀ j<i . ¬ f (suffix j  xs))) )  )
       using 1 2 3 4 by auto
   have 7: ⋀n. k =(Suc n) ⟶ ( f (x1a ⊙ xs) ∨ ( ¬(f (x1a⊙xs)) ∧
            (∃ i<n.   f (suffix i xs) ∧
              (∀ j<i . ¬ f (suffix j  xs))) )  ) =
             (  f (x1a ⊙ xs) ∨ (∃ i<n.   f (suffix i xs) ∧
              (∀ j<i . ¬ f (suffix j  xs))))

        by auto
   have 8: ⋀n. k= (Suc n) ⟶ n ≤ intlen xs +1
      using Cons.prems(2) by auto
   have 9: ⋀n. k =(Suc n) ⟶ (  f (x1a ⊙ xs) ∨ (∃ i<n.   f (suffix i xs) ∧
              (∀ j<i . ¬ f (suffix j  xs)))) =
           (  f (x1a ⊙ xs) ∨ (∃ i<n.   f (suffix i xs)))
       using 7 8 Cons.hyps by fastforce
   have 10: ⋀n. k= (Suc n) ⟶ (∃ i<k. f (suffix i (x1a ⊙ xs))) =
          (f (x1a ⊙ xs)  ∨ (∃ i<n. f (suffix i (xs))))

       using Nitpick.case-nat-unfold less-Suc-eq-0-disj by auto
   have 11: ⋀n. k =(Suc n) ⟶
           ( ∃ i<k. f (suffix i (x1a ⊙ xs)) ∧ (∀ j<i. ¬ f (suffix j (x1a ⊙ xs))))
         using 9 10 6 Cons.prems(1) by force
    show ?thesis
   using 0 11 less-imp-Suc-add by blast
  qed
qed
```

1.2.5 intapp

lemma *interval-intlen-snoc-1*:
intlen $l > 0 = (\exists\ x\ ls.\ l = ls \ominus \langle x\rangle)$

proof (*induct l*)
case (*St x*)
then show *?case* **by** *fastforce*
next
case (*Cons x1a l*)
then show *?case*
by (*metis Suc-eq-plus1 intapp-Cons intapp-St interval.exhaust interval-intlen-intapp nat.simps*(*3*) *neq0-conv*)
qed

lemma *interval-prefix-intapp* [*simp*]:
prefix (*intlen xs* $-$*k*) (*xs* $\ominus$ *ys*) = *prefix* (*intlen xs* $-$*k*) *xs*
proof
(*induct xs arbitrary*:*k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases k*)
case *0*
then show *?thesis*
by (*metis Cons.hyps diff-zero intapp-Cons interval-prefix-suc intlen.simps*(*2*) *plus-1-eq-Suc*)
next
case (*Suc nat*)
then show *?thesis*
by (*auto simp add*: *Cons.hyps Nitpick.case-nat-unfold*)
qed
qed

lemma *interval-prefix-intapp2* [*simp*]:
prefix (*intlen xs* + *k* +*1*) (*xs* $\ominus$ *ys*) = *xs* $\ominus$ *prefix k ys*
proof
(*induct xs arbitrary*: *k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases k*)
case *0*
then show *?thesis*
by *auto*
(*metis Cons.hyps Suc-eq-plus1 add.right-neutral interval-prefix-zero-intfirst*)
next
case (*Suc nat*)
then show *?thesis*
by *auto*

    (*metis Cons.hyps Suc-eq-plus1 add-Suc-right*)
  qed
qed

lemma *interval-suffix-intapp* [*simp*]:
suffix (*intlen xs* +*m* +*1*) (*xs* $\ominus$ *ys*) = *suffix* (*m*) *ys*
proof
 (*induct xs arbitrary*:*m*)
 case (*St x*)
 then show *?case* **by** *simp*
 next
 case (*Cons x1a xs*)
 then show *?case*
  proof (*cases m*)
  case *0*
  then show *?thesis*
   by *auto*
    (*metis Cons.hyps One-nat-def Suc-eq-plus1 interval-suffix-zero plus-1-eq-Suc semiring-normalization-rules*(*23*))
  next
  case (*Suc nat*)
  then show *?thesis*
   by *auto*
    (*metis Cons.hyps Suc-eq-plus1 interval-suffix-suffix*)
  qed
qed

lemma *interval-suffix-intapp2* [*simp*]:
(*suffix* (*intlen xs* − *k*) *xs*) $\ominus$ *ys* = *suffix* (*intlen xs* − *k*) (*xs* $\ominus$ *ys*)
proof
(*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
  by (*auto simp add*: *Nitpick.case-nat-unfold*)
qed

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

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

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

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

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

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

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

lemma *interval-intapp-not-state* [*simp*]:
$xs \ominus ys \neq \langle x\rangle$
by (*induct xs arbitrary*: *ys*) *simp-all*

lemma *interval-intapp-eq-intapp-conv* [*simp*]:
assumes $intlen\ xs = intlen\ ys \lor intlen\ us = intlen\ vs$
shows $(xs \ominus us = ys \ominus vs) = (xs = ys \land us = vs)$
using *assms*

proof
  (*induct xs arbitrary*: *ys*)
  case (*St x*)
  then show *?case*
    proof (*cases ys*)
    case (*St x1*)
    then show *?thesis* **by** *simp*
    next
    case (*Cons x21 x22*)
    then show *?thesis* **using** *St.prems* **by** *auto*
    qed
  next
  case (*Cons x1a xs*)
  then show *?case*
    proof (*cases ys*)
    case (*St x1*)
    then show *?thesis* **using** *Cons.prems* **by** *auto*
    next
    case (*Cons x21 x22*)
    then show *?thesis* **using** *Cons.hyps Cons.prems* **by** *auto*
    qed
qed

lemma *interval-intapp-eq-intapp-conv2*:
 $(xs \ominus ys = zs \ominus ts) =$
  $(\exists\ us.\ xs = zs \ominus us \wedge us \ominus ys = ts \vee$
        $xs = zs \wedge ys = ts \vee$
        $xs \ominus us = zs \wedge ys = us \ominus ts)$
proof
 (*induct xs arbitrary*: *ys zs ts*)
case (*St x*)
then show *?case*
 proof (*cases zs*)
 case (*St x1*)
 then show *?thesis* **by** *simp*
 next
 case (*Cons x21 x22*)
 then show *?thesis* **by** *simp*
 qed
next
case (*Cons x1a xs*)
then show *?case*
 proof (*cases zs*)
 case (*St x1*)
 then show *?thesis* **by** *simp*
 next
 case (*Cons x21 x22*)
 then show *?thesis* **by** (*auto simp add*: *Cons.hyps*)
 qed
qed

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

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

lemma *interval-intapp-same-eq*[*iff*, *induct-simp*]:
$(ys \ominus xs = zs \ominus xs) = (ys = zs)$
by *auto*

lemma *interval-suffix1-intapp*:
$suffix\ 1\ (xs \ominus ys) = (case\ xs\ of\ \langle x\rangle \Rightarrow ys \mid x \odot zs \Rightarrow zs \ominus ys)$
by (*cases xs*) *simp-all*

lemma *interval-cons-eq-intapp-conv*:
$(x\odot\ xs = ys \ominus zs) =$
$((\langle x\rangle = ys \wedge xs = zs) \vee (\exists\ ys'.\ x\odot ys' = ys \wedge xs = ys' \ominus zs))$
by (*cases ys*) *simp-all*

lemma *interval-intapp-eq-cons-conv*:
$(ys \ominus zs = x \odot xs) =$
$((\langle x\rangle = ys \wedge zs = xs) \vee (\exists\ ys'.\ ys = x\odot ys' \wedge ys' \ominus zs = xs))$
by (*cases ys*) *auto*

lemma *interval-cons-eq-intappI*:
assumes $x \odot xs1 = ys$
$xs = xs1 \ominus zs$
shows $x \odot xs = ys \ominus zs$
using *assms* **by** *auto*

lemma *interval-intapp-eq-intappI*:
assumes $xs \ominus xs1 = zs$
$ys = xs1 \ominus us$
shows $xs \ominus ys = zs \ominus us$
using *assms* **by** *auto*

lemma *intlen-intapp-gr-zero*:
$intlen\ (xs \ominus ys) > 0$
by *auto*

lemma *interval-intapp-prefix-suffix*:
assumes $i+1 \leq intlen\ xs$
$intlen\ xs > 0$
shows $xs = (prefix\ i\ xs) \ominus (suffix\ (i+1)\ xs)$

using *assms*
proof (*induct xs arbitrary*:*i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
  proof (*cases i*)
  case *0*
  then show *?thesis* **by** *auto*
  next
  case (*Suc nat*)
  then show *?thesis* **using** *Cons.hyps Cons.prems*(*1*) **by** *auto*
  qed
qed

1.2.6 Reverse

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-rev-singleton-conv* [*simp*]:
 (*intrev xs* = ⟨*x*⟩) = (*xs* = ⟨*x*⟩)
by (*metis interval-rev-rev-ident intrev.simps*(*1*))

lemma *interval-single-rev-conv* [*simp*]:
  (⟨*x*⟩ = *intrev xs*) = (⟨*x*⟩ = *xs*)
by (*metis interval-rev-rev-ident intrev.simps*(*1*))

lemma *interval-rev-is-rev-conv* [*iff*]:
 (*intrev xs* = *intrev ys*) = (*xs* = *ys*)
proof
 (*induct xs arbitrary*: *ys*)
 case (*St x*)
 then show *?case* **by** *simp*
 next
 case (*Cons x1a xs*)
 then show *?case*
 using *interval-rev-swap* **by** *force*
qed

lemma *interval-rev-induct* [*case-names St snoc*]:
assumes $\bigwedge$ *y*. *P* ⟨*y*⟩
        $\bigwedge$*x xs*. *P xs* $\Longrightarrow$ *P*(*xs* ⊖ ⟨*x*⟩)
shows   *P xs*

using *assms*
using *interval.induct*[*of* λ *xs*. *P* (*intrev xs*) *intrev xs*]
by *simp*

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

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

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

lemma *interval-intrev-intapp-cons*:
$intrev\ (xs \ominus \langle x\rangle) = x{\odot}intrev\ xs$
by (*cases xs*) *simp-all*

lemma *interval-intlast-intrev*:
intlast (*intrev xs*) = *intfirst xs*
proof (*cases xs*)
case (*St x1*)
then show *?thesis*
by *simp*
next
case (*Cons x21 x22*)
then show *?thesis*
by (*metis interval-intlast-intapp interval-nth-zero interval-nth-zero-intfirst intrev.simps*(*2*))
qed

lemma *interval-intfirst-intrev*:
intfirst (*intrev xs*) = *intlast xs*
proof
(*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
by (*metis interval-intfirst-intapp2 interval-nth-intlen-intlast interval-nth-last intrev.simps*(*2*))
qed

lemma *interval-intrev-nth*:

```
assumes k≤ intlen (intrev xs)
shows    (nth  (intrev xs) k) = (nth  xs ((intlen xs) −k))
using assms
proof
  (induct xs)
  case (St x)
  then show ?case by simp
  next
  case (Cons x1a xs)
  then show ?case
    proof (cases k)
    case 0
    then show ?thesis
     by auto
        (metis Cons.hyps diff-zero interval-intapp-nth interval-intlen-gr-zero)
    next
    case (Suc nat)
    then show ?thesis
     using Cons.hyps Suc-diff-le by (auto simp add: interval-intapp-nth) fastforce
    qed
qed


lemma interval-intrev-prefix:
assumes k≤ intlen xs
shows    intrev( prefix k xs) = suffix ((intlen xs) − k) (intrev xs)
proof
 (induct xs arbitrary: k)
 case (St x)
 then show ?case by simp
 next
 case (Cons x1a xs)
 then show ?case
   proof (cases k)
   case 0
   then show ?thesis
    by auto
      (metis Suc-eq-plus1 add.right-neutral interval-intlast-intapp interval-intlen-intapp
       interval-intrev-intlen interval-suffix-intlast intlen.simps(1))
   next
   case (Suc nat)
   then show ?thesis
    by auto
      (metis Cons.hyps interval-intrev-intlen interval-suffix-intapp2)
   qed
qed


lemma interval-intrev-suffix:
assumes k≤ intlen xs
```

shows *intrev*(*suffix k xs*) = *prefix* ((*intlen xs*) − *k*) (*intrev xs*)
using *assms*
proof
(*induct xs arbitrary*: *k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case* **by** (*simp add*: *interval-intrev-prefix interval-rev-swap*)
qed

lemma *interval-intrev-sub*:
assumes $0 \le i$
$i \le j$
$j \le$ *intlen xs*
shows *intrev* (*sub i j xs*) = *sub* ((*intlen xs*) −*j*) ((*intlen xs*) − *i*) (*intrev xs*)
using *assms*
proof −
have *1*: *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*)
have *2*: *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*
have *3*: *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*
have *4*: *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*)
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

1.2.7 Induction rule

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

lemma *interval-induct-12*:
assumes $\bigwedge x.\ P\ \langle x \rangle$
$\bigwedge x\ y.\ P\ \langle x{,}y \rangle$
$\bigwedge x\ y\ zs.\ P\ (y \odot zs) \Longrightarrow P\ (x \odot y \odot zs)$
shows *P xs*
using *assms*
proof (*induction xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*

by *auto*
  (*metis add-cancel-right-right interval-intlen-cons-1 interval-st-intlen le-add1 le-eq-less-or-eq less-add-same-cancel1*)
qed

lemma *interval-induct2* [*consumes 1*, *case-names St Cons*]:
assumes $intlen\ xs = intlen\ ys$
$(\bigwedge x\ y.\ P\ \langle x\rangle\ \langle y\rangle)$
$(\bigwedge x\ xs\ y\ ys.\ intlen\ xs = intlen\ ys \Longrightarrow P\ xs\ ys \Longrightarrow P\ (x \odot xs)\ (y \odot ys))$
shows $P\ xs\ ys$
using *assms*
proof (*induction xs arbitrary*: *ys*)
case (*St x*)
then show *?case*
by (*metis interval-st-intlen*)
next
case (*Cons x1a xs*)
then show *?case*
by (*metis interval-intlen-cons-1 intlen.simps*(*2*) *nat.simps*(*1*) *plus-1-eq-Suc*)
qed

1.2.8 Map

lemma *map-ext*:
assumes $(\forall x.\ x \in set\ xs \longrightarrow f\ x = g\ x)$
shows $map\ f\ xs = map\ g\ xs$
using *assms*
by (*induct xs*) *simp-all*

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

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

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

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

lemma *intrev-map*:
intrev (*map f xs*) = *map f*(*intrev xs*)
by (*induct xs*) *auto*

lemma *map-eq-conv* [*simp*]:
(*map f xs* = *map g xs*) = (∀ *x*∈ *set xs*. (*f x*) = (*g x*))
by (*induct xs*) *auto*

lemma *map-cong* [*fundef-cong*]:
assumes *xs* = *ys*
(∀ *x*. *x* ∈ *set ys* ⟶ *f x* = *g x*)
shows *map f xs* = *map g ys*
using *assms* **by** *simp*

lemma *map-injective*:
assumes *map f xs* = *map f ys*
inj f
shows *xs* = *ys*
using *assms* **by** (*meson injD interval.inj-map*)

lemma *inj-map-eq-map* [*simp*]:
assumes *inj f*
shows (*map f xs* = *map f ys*) = (*xs* = *ys*)
using *assms* **by** (*blast dest*: *map-injective*)

lemma *inj-mapI*:
assumes *inj f*
shows *inj* (*map f*)
using *assms interval.inj-map* **by** *blast*

lemma *inj-mapD*:
assumes *inj* (*map f*)
shows *inj f*
using *assms* **by** (*metis inj-def interval.map*(*1*) *interval.simps*(*1*))

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

lemma *map-idI*:
assumes (∀ *x*. *x* ∈ *set xs* ⟶ *f x* = *x*)
shows *map f xs* = *xs*
using *assms* **by** (*induct xs*) *auto*

lemma *map-is-state-conv*[*iff*]:
(*map f xs* = ⟨*x*⟩) = (∃ *y*. *xs* = ⟨*y*⟩ ∧ *f y* = *x*)
proof (*cases xs*)
case (*St x1*)
then show *?thesis* **by** *simp*
next

case (*Cons x21 x22*)
then show *?thesis* **by** *simp*
qed

lemma *state-is-map-conv* [*iff*]:
$(\langle x\rangle = map\ f\ xs) = (\exists\ y.\ \langle y\rangle = xs \wedge f\ y = x)$
proof (*cases xs*)
case (*St x1*)
then show *?thesis* **by** *auto*
next
case (*Cons x21 x22*)
then show *?thesis* **by** *auto*
qed

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

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

lemma *ex-map-conv*:
$(\exists xs.\ ys = map\ f\ xs) = (\forall y \in set\ ys.\ \exists x.\ y = f\ x)$
by (*induct ys*) (*auto simp add*: *cons-eq-map-conv*)

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

declare *map.id* [*simp*]

lemma *intfirst-map*:
$intfirst\ (map\ f\ xs) = f\ (intfirst\ xs)$
by (*cases xs*) *simp-all*

lemma *intlast-map*:
$intlast\ (map\ f\ xs) = f\ (intlast\ xs)$
proof (*cases xs rule*: *interval-rev-cases*)
case (*St x*)
then show *?thesis* **by** *simp*
next
case (*snoc ys y*)
then show *?thesis* **by** (*simp add*: *interval-intapp-nth*)
qed

lemma *map-tail*:

shows *map f (suffix 1 xs) = (suffix 1 (map f xs))*
by (*cases xs*) *simp-all*

lemma *map-eq-imp-intlen-eq*:
 assumes *map f xs = map g ys*
 shows *intlen xs = intlen ys*
using *assms*
proof (*induct ys arbitrary*: *xs*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a ys*)
then show *?case* **by** (*metis interval-intlen-map*)
qed

lemma *interval-set-map* [*simp*]:
 set (map f xs) = f'(set xs)
by (*induct xs*) *auto*

lemma *map-inj-on*:
 assumes *map*: *map f xs = map f ys* **and**
 inj: *inj-on f* (*set xs* $\cup$ *set ys*)
 shows *xs =ys*
using *map-eq-imp-intlen-eq*[*OF map*] *assms*
proof (*induction rule*: *interval-induct2*)
case (*St x y*)
then show *?case* **by** *auto*
next
case (*Cons x xs y ys*)
then show *?case*
by (*metis* (*mono-tags*, *hide-lams*) *UnI1 UnI2 inj-on-def interval.inj-map-strong*)
qed

lemma *inj-on-map-eq-map*:
assumes *inj-on f* (*set xs* $\cup$ *set ys*)
shows (*map f xs = map f ys*) = (*xs = ys*)
using *assms* **by** (*blast dest*:*map-inj-on*)

lemma *inj-on-mapI*:
 assumes *inj-on f* ($\bigcup$ (*set ' A*))
 shows *inj-on* (*map f*) *A*
using *assms* **by** (*blast intro*:*inj-onI dest*:*inj-onD map-inj-on*)

lemma *map-prefix*:
 assumes *k* $\leq$ *intlen xs*
 shows *map f (prefix k xs) = prefix k (map f xs)*
using *assms*
proof (*induction xs arbitrary*: *k*)
case (*St x*)
then show *?case* **by** *simp*

next
case (*Cons x1a xs*)
then show *?case*
 proof (*cases k*)
 case *0*
 then show *?thesis* **by** *auto*
 next
 case (*Suc nat*)
 then show *?thesis* **using** *Cons.IH Cons.prems* **by** *auto*
 qed
qed

lemma *map-suffix*:
 assumes $k \leq$ *intlen xs*
 shows *map f* (*suffix k xs*) = *suffix k* (*map f xs*)
using *assms*
proof (*induction xs arbitrary*: *k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
 proof (*cases k*)
 case *0*
 then show *?thesis* **by** *auto*
 next
 case (*Suc nat*)
 then show *?thesis* **using** *Cons.IH Cons.prems* **by** *auto*
 qed
qed

1.2.9 index sequence

lemma *interval-idx-less*:
 assumes *index-sequence x idx*
 $n+k<$*intlen idx*
 shows *nth idx n* < *nth idx* (*Suc*($n+k$))
using *index-sequence-def assms* **by** (*induct k*) *auto*

lemma *interval-idx-less-eq*:
 assumes *index-sequence x l*
 $k \leq j$
 $j \leq$ *intlen l*
 shows *nth l k* $\leq$ *nth l j*
using *assms*
proof (*cases* $k=j$)
 show *index-sequence x l* $\Longrightarrow k \leq j \Longrightarrow j \leq$ *intlen l* $\Longrightarrow k = j \Longrightarrow$ *nth l k* $\leq$ *nth l j*
 by *blast*
 show *index-sequence x l* $\Longrightarrow k \leq j \Longrightarrow j \leq$ *intlen l* $\Longrightarrow k \neq j \Longrightarrow$ *nth l k* $\leq$ *nth l j*

by (*metis Suc-le-lessD interval-idx-less le-SucE le-eq-less-or-eq le-zero-eq ordered-cancel-comm-monoid-diff-class.add-diff-inverse zero-induct*)
qed

lemma *interval-idx-mono*:
assumes *index-sequence x l*
shows *mono* (λ *x*. *nth l x*)
proof –
have *1*: $\forall x\ y.\ x \leq y \longrightarrow nth\ l\ x \leq nth\ l\ y$
proof
fix *x*
show $\forall y{\geq}x.\ nth\ l\ x \leq nth\ l\ y$
proof
fix *y*
show $x \leq y \longrightarrow nth\ l\ x \leq nth\ l\ y$
proof –
have *2*: $x{\leq}\ y \wedge y \leq intlen\ l \Longrightarrow nth\ l\ x \leq nth\ l\ y$
using *assms interval-idx-less-eq* **by** *blast*
have *3*: $x{\leq}\ y\ \wedge x{>}intlen\ l \Longrightarrow nth\ l\ x \leq nth\ l\ y$
using *assms interval-nth-last-stutter*
by (*metis 2 le-cases less-imp-add-positive ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *4*: $x{\leq}\ y\ \wedge x{\leq}intlen\ l \wedge y > intlen\ l \Longrightarrow nth\ l\ x \leq nth\ l\ y$
by (*metis assms interval-idx-less-eq interval-nth-last-stutter less-imp-add-positive order-refl*)
show *?thesis*
using *2 3 4 not-less* **by** *blast*
qed
qed
qed
show *?thesis*
by (*simp add*: *1 monoI*)
qed

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

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

lemma *interval-idx-greater-first*:

assumes *index-sequence x idx*
0 < i
i≤ intlen idx
shows *x < nth idx i*
using *assms*
proof −
have *1*: *nth idx 0 = x*
using *assms* **by** (*simp add*: *index-sequence-def*)
have *2*: ∀ *i*. *0 < i* ∧ *i≤ intlen idx* ⟶ (*nth idx 0*) < (*nth idx i*)
proof
fix *i*
show *0 < i* ∧ *i ≤ intlen idx* ⟶ *nth idx 0 < nth idx i*
by (*meson Suc-leI assms*(*1*) *dual-order.strict-trans1 index-sequence-def interval-idx-less-eq*)
qed
show *?thesis*
using *1 2 assms*(*2*) *assms*(*3*) **by** *blast*
qed

lemma *interval-idx-cons*:
index-sequence y (*x*⊙*ls*) =
(*x=y* ∧ *x<nth ls 0* ∧ *index-sequence* (*nth ls 0*) *ls*)
using *less-Suc-eq-0-disj* **by** (*simp add*: *index-sequence-def*) *auto*

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

lemma *interval-idx-expand*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = (*intlen xs*)
i< (*intlen l*)
shows (*nth l i*) ≤ (*nth l* (*i+1*)) ∧ (*nth l* (*i+1*)) ≤ (*intlen xs*)
using *assms*
by (*metis Suc-eq-plus1 Suc-lessI add.right-neutral interval-idx-less interval-idx-less-last-1*
less-imp-le-nat order-refl)

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-nth-map*)

lemma *interval-idx-shiftm* :
assumes *index-sequence k lsk*
shows *index-sequence 0* (*map* (*shiftm k*) *lsk*) ∧ *k* ≤ (*nth lsk 0*)
using *assms*
proof (*auto simp add*: *index-sequence-def shiftm-def interval-nth-map*)
show ⋀*n*. ∀ *n<intlen lsk*. *nth lsk n < nth lsk* (*Suc n*) ⟹
k = nth lsk 0 ⟹
n < intlen lsk ⟹ *nth lsk n* − *nth lsk 0 < nth lsk* (*Suc n*) − *nth lsk 0*
by (*metis assms diff-less-mono interval-idx-greater-first le-eq-less-or-eq neq0-conv*)
qed

lemma *interval-idx-shiftm-a* :
  assumes *index-sequence 0* (*map* (*shiftm k*) *lsk*)
    $k \leq$ (*nth lsk 0*)
 shows *index-sequence k lsk*
using *assms*
by (*auto simp add*: *index-sequence-def shiftm-def interval-nth-map*)

lemma *interval-idx-shiftm-b* :
 index-sequence k lsk = (*index-sequence 0* (*map* (*shiftm k*) *lsk*) $\wedge$ $k \leq$ (*nth lsk 0*))
using *interval-idx-shiftm interval-idx-shiftm-a* **by** *blast*

lemma *interval-idx-shiftm-c* :
 index-sequence (*nth lsk 0*) *lsk* = *index-sequence 0* (*map* (*shiftm* (*nth lsk 0*)) *lsk*)
using *interval-idx-shiftm-b* **by** *blast*

lemma *interval-lsk-ls* :
  (*index-sequence k* (*lsk*) $\wedge$ *lsk* = *map* (*shift k*) *ls* $\wedge$ *index-sequence 0* (*ls*)) =
   (*index-sequence k* (*lsk*) $\wedge$ *ls* = *map* (*shiftm k*) *lsk* $\wedge$ *index-sequence 0* (*ls*))
proof (*simp add*: *interval-eq-nth-eq index-sequence-def shift-def shiftm-def interval-nth-map*)
  show (*nth lsk 0* = *k* $\wedge$
   ($\forall$ *n*<*intlen lsk*. *nth lsk n* < *nth lsk* (*Suc n*)) $\wedge$
   intlen lsk = *intlen ls* $\wedge$
   ($\forall$ *i*$\leq$*intlen lsk*. *nth lsk i* = *nth ls i* + *k*) $\wedge$
   nth ls 0 = *0* $\wedge$ ($\forall$ *n*<*intlen ls*. *nth ls n* < *nth ls* (*Suc n*))) =
  (*nth lsk 0* = *k* $\wedge$
   ($\forall$ *n*<*intlen lsk*. *nth lsk n* < *nth lsk* (*Suc n*)) $\wedge$
   intlen ls = *intlen lsk* $\wedge$
   ($\forall$ *i*$\leq$*intlen ls*. *nth ls i* = *nth lsk i* − *k*) $\wedge$
   nth ls 0 = *0* $\wedge$ ($\forall$ *n*<*intlen ls*. *nth ls n* < *nth ls* (*Suc n*))) (**is** *?L*=*?R*)
  proof *rule*
   show *?L* $\Longrightarrow$ *?R*
  by (*metis* (*no-types*, *lifting*) *add-diff-cancel-right'*)
   show *?R* $\Longrightarrow$ *?L*
    by (*metis add.commute add-diff-inverse-nat diff-is-0-eq less-eq-Suc-le less-irrefl-nat
     less-nat-zero-code less-or-eq-imp-le old.nat.exhaust*)
  qed
qed

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

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

lemma *interval-idx-bound-0* :
assumes *index-sequence 0 ls*
nth ls (*intlen ls*) = *intlen* (*suffix k xs*)
i≤*intlen ls*
shows *nth ls i* ≤ *intlen* (*suffix k xs*)
using *assms*
by (*metis eq-iff interval-idx-less-last-1 le-neq-implies-less less-imp-le-nat*)

lemma *interval-idx-bound-1*:
(*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*

lemma *interval-idx-less-equal*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen xs*
i ≤ *intlen l*
n ≤ *intlen l*
shows ∀ *j* . *j*≤*n* ⟶ *nth l j* ≤ *nth l n*
using *assms*
using *interval-idx-less-eq* **by** *blast*

lemma *interval-idx-less-than*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen xs*
i ≤ *intlen l*
n ≤ *intlen l*
shows ∀ *j*. *j*>*n* ∧ *j* ≤ *intlen l*⟶ *nth l j* > *nth l n*
by (*meson Suc-leI assms interval-idx-less-equal index-sequence-def less-le-trans*)

lemma *interval-idx-sub*:
assumes *k*≤*n*
n ≤ *intlen l*
index-sequence 0 l
shows *index-sequence* (*nth l k*) (*sub k n l*)
proof −
have *1*: *index-sequence* (*nth l k*) (*sub k n l*) =
((*nth* (*sub k n l*) *0*) = (*nth l k*) ∧
(∀ *i*. *i*<*intlen*(*sub k n l*) ⟶ (*nth* (*sub k n l*) *i*) < (*nth* (*sub k n l*) (*Suc i*))))
using *index-sequence-def* **by** *auto*
have *2*: (*nth* (*sub k n l*) *0*) = (*nth l k*)
using *assms interval-intfirst-sub* **by** *auto*
have *3*: *intlen*(*sub k n l*) = (*n* −*k*)
by (*simp add*: *assms*)
have *4*: (∀ *i*. *i*<*intlen*(*sub k n l*) ⟶ (*nth* (*sub k n l*) *i*) < (*nth* (*sub k n l*) (*Suc i*)))

proof
fix *i*
show $i < intlen\ (sub\ k\ n\ l) \longrightarrow nth\ (sub\ k\ n\ l)\ i < nth\ (sub\ k\ n\ l)\ (Suc\ i)$
proof −
have *41*: $i < (n-k) \longrightarrow nth\ (sub\ k\ n\ l)\ i = nth\ l\ (k+i)$
by (*simp add*: *assms*)
have *42*: $i < (n-k) \longrightarrow nth\ (sub\ k\ n\ l)\ (Suc\ i) = nth\ l\ (k+(Suc\ i))$
by (*simp add*: *assms*)
have *43*: $i < (n-k) \longrightarrow nth\ l\ (k+i) < nth\ l\ (k+(Suc\ i))$
using *assms*
using *index-sequence-def* **by** *auto*
show *?thesis* **using** *3 41 42 43* **by** *auto*
qed
qed
show *?thesis* **by** (*simp add*: *1 2 4*)
qed

lemma *interval-idx-split*:
assumes $n \le intlen\ l$
shows $index\text{-}sequence\ 0\ l =$
$(index\text{-}sequence\ 0\ (prefix\ n\ l) \wedge index\text{-}sequence\ (nth\ l\ n)\ (suffix\ n\ l))$
proof −
have *1*: $index\text{-}sequence\ 0\ l \longrightarrow$
$index\text{-}sequence\ 0\ (prefix\ n\ l) \wedge index\text{-}sequence\ (nth\ l\ n)\ (suffix\ n\ l)$
using *interval-idx-sub* **using** *assms index-sequence-def* **by** *auto*
have *2*: $index\text{-}sequence\ 0\ (prefix\ n\ l) =$
$(\ (nth\ (prefix\ n\ l)\ 0) = 0 \wedge$
$(\forall i.\ i < intlen(prefix\ n\ l) \longrightarrow (nth\ (prefix\ n\ l)\ i) < (nth\ (prefix\ n\ l)\ (Suc\ i))))$
using *index-sequence-def* **by** *blast*
have *3*: $(\ (nth\ (prefix\ n\ l)\ 0) = 0 \wedge$
$(\forall i.\ i < intlen(prefix\ n\ l) \longrightarrow (nth\ (prefix\ n\ l)\ i) < (nth\ (prefix\ n\ l)\ (Suc\ i)))) =$
$(\ (nth\ l\ 0) = 0 \wedge$
$(\forall i.\ i<n \longrightarrow (nth\ l\ i) < (nth\ l\ (Suc\ i))))$
using *assms* **by** *auto*
have *4*: $index\text{-}sequence\ (nth\ l\ n)\ (suffix\ n\ l) =$
$(\ (nth\ (suffix\ n\ l)\ 0) = (nth\ l\ n) \wedge$
$(\forall\ i.\ i<intlen(suffix\ n\ l) \longrightarrow (nth\ (suffix\ n\ l)\ i) < (nth\ (suffix\ n\ l)\ (Suc\ i))))$
using *index-sequence-def* **by** *blast*
have *5*: $(\ (nth\ (suffix\ n\ l)\ 0) = (nth\ l\ n) \wedge$
$(\forall\ i.\ i<intlen(suffix\ n\ l) \longrightarrow (nth\ (suffix\ n\ l)\ i) < (nth\ (suffix\ n\ l)\ (Suc\ i)))) =$
$(\ (nth\ l\ n) = (nth\ l\ n) \wedge$
$(\forall\ i.\ i<intlen\ l - n \longrightarrow (nth\ l\ (i+n)\ ) < (nth\ l\ ((Suc\ i)+n))))$
by (*metis* (*no-types*, *lifting*) *Suc-leI add.commute assms interval-intlast-intfirst*
interval-intlast-prefix interval-nth-suffix interval-nth-zero-intfirst
interval-suffix-length-good less-imp-le-nat)
have *6*: $(\forall\ i.\ i<intlen\ l - n \longrightarrow (nth\ l\ (i+n)\ ) < (nth\ l\ ((Suc\ i)+n))) =$
$(\forall\ i.\ n \le (i+n) \wedge (i+n)<intlen\ l\ \longrightarrow (nth\ l\ (i+n)\ ) < (nth\ l\ ((Suc\ i)+n)))$
by *auto*
have *7*: $(index\text{-}sequence\ 0\ (prefix\ n\ l) \wedge index\text{-}sequence\ (nth\ l\ n)\ (suffix\ n\ l)) \longrightarrow$
$(nth\ l\ 0) = 0 \wedge (\forall i.\ i<n \longrightarrow (nth\ l\ i) < (nth\ l\ (Suc\ i)))$

using *2 3* **by** *blast*
have *8*: (*index-sequence 0* (*prefix n l*) $\wedge$ *index-sequence* (*nth l n*) (*suffix n l*)) $\longrightarrow$
($\forall$ *i*. $n \leq i \wedge i < intlen\ l \longrightarrow (nth\ l\ (i)\ ) < (nth\ l\ ((Suc\ i))))$)
by (*metis* (*no-types*, *lifting*) *4 5 6 add-Suc diff-add*)
have *9*: (*index-sequence 0* (*prefix n l*) $\wedge$ *index-sequence* (*nth l n*) (*suffix n l*)) $\longrightarrow$
$(nth\ l\ 0) = 0 \wedge (\forall i.\ i < intlen\ l \longrightarrow (nth\ l\ i) < (nth\ l\ (Suc\ i)))$
using *7 8 not-le* **by** *blast*
have *10*: (*index-sequence 0* (*prefix n l*) $\wedge$ *index-sequence* (*nth l n*) (*suffix n l*)) $\longrightarrow$
index-sequence 0 l
using *9 index-sequence-def* **by** *blast*
from *10 1* **show** *?thesis* **by** *blast*
qed

lemma *interval-idx-suffixa*:
assumes $n \leq$ *intlen l*
index-sequence (*nth l n*) (*suffix n l*)
shows *index-sequence 0* ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))
using *assms interval-idx-shiftm* **by** *blast*

lemma *interval-idx-greater*:
assumes *index-sequence k l*
shows $(\forall i.\ i \leq intlen\ l \longrightarrow k \leq (nth\ l\ i))$
by (*metis assms eq-iff index-sequence-def interval-idx-greater-first less-imp-le neq0-conv*)

lemma *interval-idx-suffixb*:
assumes $n \leq$ *intlen l*
index-sequence 0 ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))
shows *index-sequence* (*nth l n*) (*suffix n l*)
by (*metis assms interval-idx-shiftm-c interval-intlast-intfirst interval-intlast-prefix*
interval-nth-zero-intfirst)

lemma *interval-idx-suffix*:
assumes $n \leq$ *intlen l*
shows *index-sequence* (*nth l n*) (*suffix n l*) =
index-sequence 0 ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))
using *assms interval-idx-shiftm interval-idx-suffixb* **by** *blast*

lemma *interval-idx-intfirst*:
assumes *index-sequence 0* (*x1a*$\odot$ *l*)
shows *x1a* < *intfirst*(*l*)
by (*metis assms interval-idx-cons interval-nth-zero-intfirst*)

lemma *interval-idx-expand1*:
(*index-sequence x1a* (*x1a* $\odot$ *l*)) = (*x1a* < *intfirst l* $\wedge$ *index-sequence* (*intfirst l*) *l*)
using *interval-nth-zero-intfirst less-Suc-eq-0-disj* **by** (*auto simp add*: *index-sequence-def*)

lemma *interval-idx-intlen-leq-intlast-intfirst*:
assumes *index-sequence* (*intfirst l*) *l*
shows *intlen* (*l*) $\leq$ (*intlast l* $-$ *intfirst l*)

using *assms*
proof
(*induct l*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a l*)
then show *?case*
proof −
have *1*: *intlen* (*x1a* $\odot$ *l*) = *intlen l* +*1*
by *simp*
have *2*: *index-sequence* (*intfirst l*) *l*
using *Cons.prems interval-idx-expand1* **by** *auto*
have *3*: *intlast* (*x1a* $\odot$ *l*) = *intlast l*
by *simp*
have *4*: *intfirst* (*x1a* $\odot$ *l*) = *x1a*
by *simp*
have *5*: *x1a* < *intfirst l*
using *Cons.prems interval-idx-expand1* **by** *auto*
have *6*: *intlen l* ≤ *intlast l* − *intfirst l*
using *2 Cons.hyps* **by** *blast*
have *7*: *intlen l* +*1* ≤ (*intlast l* − *intfirst l*)+*1*
using *6 add-le-cancel-right* **by** *blast*
have *8*: (*intlast l* − *intfirst l*) +*1* ≤ *intlast l* − *x1a*
by (*metis 5 6 Suc-eq-plus1 diff-is-0-eq diff-less-mono diff-less-mono2 interval-nth-intlen-intlast interval-nth-zero-intfirst le-antisym less-eq-Suc-le nat-le-linear not-le not-less0*)
show *?thesis* **using** *7 8* **by** *auto*
qed
qed

lemma *interval-idx-intlen-leq*:
assumes *index-sequence* (*intfirst l*) *l*
intlast(*l*) ≤ *intlen xs*
shows *intlen* (*l*) ≤ *intlen* (*sub* (*intfirst l*) (*intlast l*) *xs*)
proof −
have *1*: *intfirst l* ≤ *intlast l*
using *assms* **by** (*metis eq-iff gr0I index-sequence-def interval-idx-less-last-1 interval-nth-intlen-intlast less-imp-le-nat*)
have *2*: *intlen* (*sub* (*intfirst l*) (*intlast l*) *xs*) = (*intlast l* − *intfirst l*)
using *1 assms interval-intlen-sub* **by** *blast*
have *3*: *intlen* (*l*) ≤ (*intlast l* − *intfirst l*)
using *assms interval-idx-intlen-leq-intlast-intfirst* **by** *blast*
show *?thesis* **using** *2 3* **by** *linarith*
qed

lemma *interval-idx-shiftm-sub-nth*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen xs*

$k \leq n$
$n \leq intlen\ l$
shows $\forall\ j.\ j \leq n{-}k \longrightarrow$
$nth\ (map\ (shiftm\ (nth\ l\ k))\ (sub\ k\ n\ l))\ j = nth\ l\ (k{+}j) - (nth\ l\ k)$
by (*simp add*: *Nat.le-diff-conv2 assms interval-nth-map shiftm-def*)

lemma *interval-idx-shiftm-suffix-nth*:
assumes *index-sequence 0 l*
$(nth\ l\ (intlen\ l)) = intlen\ xs$
$n \leq intlen\ l$
shows $\forall\ j.\ j \leq intlen\ l\ {-}n \longrightarrow$
$nth\ (map\ (shiftm\ (nth\ l\ n))\ (suffix\ n\ l))\ j = nth\ l\ (n{+}j) - (nth\ l\ n)$
using *assms* **by** (*metis interval-nth-map interval-nth-suffix shiftm-def*)

1.2.10 upt

lemma *upt-rec*[*code*]:
$[i..{\leq}j] = (if\ i{<}j\ then\ i \odot [Suc\ i..{\leq}j]\ else\ \langle j\rangle)$
by (*induct j*) *auto*

lemma *upt-conv-st* [*simp*]:
assumes $j < i$
shows $[i..{\leq}j] = \langle j\rangle$
using *assms*
by (*metis Interval.upt.simps*(1) *Interval.upt.simps*(2) *Suc-leD less-Suc-eq-0-disj not-le*)

lemma *upt-same*:
$[i..{\leq}i] = \langle i\rangle$
by (*metis Interval.upt.simps*(1) *Interval.upt.simps*(2) *less-Suc-eq less-Suc-eq-0-disj not-le*)

lemma *upt-eq-st-conv*[*simp*]:
$([i..{\leq}j] = \langle j\rangle) = (j{\leq}i)$
by (*simp add*: *Interval.upt-rec*)

lemma *upt-eq-cons-conv*:
$([i..{\leq}j] = x{\odot}xs) = (i{<}j \wedge i{=}x \wedge [Suc\ i..{\leq}j] = xs)$
using *Interval.upt-rec* **by** (*induct j arbitrary*: *x xs*) *auto*

lemma *upt-suc-append*:
assumes $i \leq j$
shows $[i..{\leq}(Suc\ j)] = [i..{\leq}j]{\ominus}\langle(Suc\ j)\rangle$
using *assms* **by** *simp*

lemma *upt-conv-cons*:
assumes $i{<}j$
shows $[i..{\leq}j] = i \odot [(Suc\ i)..{\leq}j]$
using *assms* **by** (*simp add*: *upt-rec*)

lemma *upt-conv-cons-cons*:

$(m \odot n \odot ns = [m..\leq q]) = (n \odot ns = [(Suc\ m)..\leq q])$
proof (*cases* $m \leq q$)
case *True*
then show *?thesis* **by** (*simp add*: *Interval.upt-rec*)
next
case *False*
then show *?thesis* **by** *auto*
qed

lemma *upt-add-eq-append*:
assumes $i \leq j$
$k>0$
shows $[i..\leq j+k] = [i..\leq j] \ominus [Suc\ j..\leq j+k]$
using *assms*
proof
(*induct k*)
case *0*
then show *?case* **by** *blast*
next
case (*Suc k*)
then show *?case* **using** *Suc-less-eq le-simps*(*2*) **by** *auto*
qed

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

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

lemma *upt-nth*:
assumes $i + k \leq j$
shows $(nth\ [i..\leq j]\ k) = i + k$
using *assms*
proof
(*induct j arbitrary*: *k i*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc j*)

```
  then show ?case
   by (metis Interval.upt.upt-Suc Nat.le-diff-conv2 add.commute add-leD1
         interval-intapp-nth le-SucE upt-length upt-nth-help)
qed

lemma upt-intfirst:
assumes i ≤ j
shows   intfirst [i..≤j] = i
using assms by (simp add: Interval.upt-rec)

lemma upt-intlast:
  intlast [i..≤j] = j
by (metis add-diff-inverse-nat interval-nth-intlen-intlast interval-st-intlen order-refl
         upt-conv-st upt-length upt-nth)

lemma prefix-upt:
assumes i+m ≤ n
shows   prefix m [i..≤n] = [i..≤i+m]
using assms
proof
 (induct m arbitrary: i)
 case 0
 then show ?case by (simp add: upt-nth upt-same)
 next
 case (Suc m)
 then show ?case using Interval.upt-rec by auto
qed

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

lemma map-suc-upt:
  map Suc [m..≤n] = [Suc m..≤Suc n]
proof
 (induct n arbitrary: m)
 case 0
 then show ?case by simp
 next
 case (Suc n)
 then show ?case by simp
qed
```

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

1.2.11 Set

lemma *interval-set-intapp* [*simp*]:
set ($xs \ominus ys$) = (*set xs* $\cup$ *set ys*)
by (*induct xs*) *auto*

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

lemma *interval-hd-in-set* [*simp*]:
$x \in$ *set* ($x \odot xs$)
by *simp*

lemma *interval-set-subset-Cons*:
set xs $\subseteq$ *set* ($x \odot xs$)
by *auto*

lemma *interval-set-ConsD*:
assumes $y \in$ *set* ($x \odot xs$)
shows $y = x \vee y \in$ *set xs*
using *assms* **by** *auto*

lemma *interval-exists-cons*:
($\exists$ *ys* $\in$ *set* (($x \odot xs$)). *P ys*) $\longleftrightarrow$
(*P x* $\wedge$ ($\exists$ *ys* $\in$ *set xs*. *P ys*)) $\vee$ ($\neg$ *P x* $\wedge$ ($\exists$ *ys* $\in$ *set xs*. *P ys*)) $\vee$
(*P x* $\wedge$ ($\forall$ *ys* $\in$ *set xs*. $\neg$ *P ys*))
by *auto*

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

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

lemma *interval-split-interval*:
assumes $x \in set\ xs$
shows $\exists\ ys\ zs.\ xs = ys \ominus (x \odot zs) \lor xs = \langle x\rangle \lor xs = x \odot zs \lor xs = ys \ominus \langle x\rangle$
using *assms*
proof (*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
by (*metis intapp-St interval.inject(2) interval.set-cases interval-intapp-assoc*)
qed

lemma *interval-in-set-conv-decomp*:
$x \in set\ xs =$
$(\exists\ ys\ zs.\ xs = ys \ominus (x \odot zs) \lor xs = \langle x\rangle \lor xs = x \odot zs \lor xs = ys \ominus \langle x\rangle)$
by (*auto elim*: *interval-split-interval*)

lemma *interval-split-interval-first*:
assumes $x \in set\ xs$
shows $\exists\ ys\ zs.\ xs = ys \ominus (x \odot zs) \land x \notin set\ ys \lor xs = \langle x\rangle \lor xs = x \odot zs \lor$
$xs = ys \ominus \langle x\rangle \land x \notin set\ ys$
using *assms*
proof (*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases* $x = x1a$)
case *True*
then show *?thesis* **by** *blast*
next
case *False*
then show *?thesis*
using *Cons interval-cons-eq-intappI*
proof *auto*
show $\bigwedge ys\ zs.$
$x \neq x1a \Longrightarrow$
$(\bigwedge x\ xs1\ ys\ xs\ zs.\ x \odot xs1 = ys \Longrightarrow xs = xs1 \ominus zs \Longrightarrow x \odot xs = ys \ominus zs) \Longrightarrow$
$x \notin interval.set\ ys \Longrightarrow$
$xs = ys \ominus x \odot zs \Longrightarrow$
$\exists ysa.\ (\exists zsa.\ x1a \odot ys \ominus x \odot zs = ysa \ominus x \odot zsa) \land x \notin interval.set\ ysa \lor$
$x1a \odot ys \ominus x \odot zs = ysa \ominus \langle x\rangle \land x \notin interval.set\ ysa$
by (*metis Un-iff empty-iff insert-iff intapp-St interval.simps(15) interval-intapp-assoc interval-set-intapp*)
show $x \neq x1a \Longrightarrow$
$(\bigwedge x\ xs1\ ys\ xs\ zs.\ x \odot xs1 = ys \Longrightarrow xs = xs1 \ominus zs \Longrightarrow x \odot xs = ys \ominus zs) \Longrightarrow$
$xs = \langle x\rangle \Longrightarrow$

$\exists ys.\ (\exists zs.\ \langle x1a,\ x\rangle = ys \ominus x \odot zs) \wedge x \notin interval.set\ ys \vee \langle x1a,\ x\rangle = ys \ominus \langle x\rangle \wedge$
$x \notin interval.set\ ys$
by (*metis insert-absorb insert-iff interval.simps(15) interval-intrev-intapp-cons interval-rev-intapp interval-set-nonempty intrev.simps(1)*)
show $\bigwedge zs.\ x \neq x1a \Longrightarrow$
$(\bigwedge x\ xs1\ ys\ xs\ zs.\ x \odot xs1 = ys \Longrightarrow xs = xs1 \ominus zs \Longrightarrow x \odot xs = ys \ominus zs) \Longrightarrow$
$xs = x \odot zs \Longrightarrow$
$\exists ys.\ (\exists zsa.\ x1a \odot x \odot zs = ys \ominus x \odot zsa) \wedge x \notin interval.set\ ys \vee$
$x1a \odot x \odot zs = ys \ominus \langle x\rangle \wedge x \notin interval.set\ ys$
by (*metis Interval.nth.simps(1) intapp-St interval.set-cases interval-intapp-not-state*)
show $\bigwedge ys.\ x \neq x1a \Longrightarrow$
$(\bigwedge x\ xs1\ ys\ xs\ zs.\ x \odot xs1 = ys \Longrightarrow xs = xs1 \ominus zs \Longrightarrow x \odot xs = ys \ominus zs) \Longrightarrow$
$x \notin interval.set\ ys \Longrightarrow$
$xs = ys \ominus \langle x\rangle \Longrightarrow$
$\exists ysa.\ (\exists zs.\ x1a \odot ys \ominus \langle x\rangle = ysa \ominus x \odot zs) \wedge x \notin interval.set\ ysa \vee$
$x1a \odot ys \ominus \langle x\rangle = ysa \ominus \langle x\rangle \wedge x \notin interval.set\ ysa$
by (*metis insert-iff intapp-Cons interval.simps(16)*)
qed
qed
qed

lemma *in-set-conv-decomp-first*:
$x \in set\ xs =$
$(\exists\ ys\ zs.\ xs = ys \ominus (x \odot zs) \wedge x \notin set\ ys \vee xs = \langle x\rangle \vee xs = x \odot zs \vee$
$xs = ys \ominus \langle x\rangle \wedge x \notin set\ ys)$
by (*auto dest*!: *interval-split-interval-first*)

lemma *interval-split-interval-last*:
assumes $x \in set\ xs$
shows $\exists\ ys\ zs.\ xs = ys \ominus (x \odot zs) \wedge x \notin set\ zs \vee xs = \langle x\rangle \vee xs = x \odot zs \wedge x \notin set\ zs \vee$
$xs = ys \ominus \langle x\rangle$
using *assms*
proof (*induct xs rule*: *interval-rev-induct*)
case (*St y*)
then show *?case* **by** *simp*
next
case (*snoc x1a xs*)
then show *?case* **proof** (*cases* $x = x1a$)
case *True*
then show *?thesis* **by** *blast*
next
case *False*
then show *?thesis* **using** *snoc* **by** *fastforce*
qed
qed

lemma *interval-in-set-conv-decomp-last*:
$x \in set\ xs =$
$(\exists\ ys\ zs.\ xs = ys \ominus (x \odot zs) \wedge x \notin set\ zs \vee xs = \langle x\rangle \vee xs = x \odot zs \wedge x \notin set\ zs \vee$
$xs = ys \ominus \langle x\rangle)$

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

lemma *interval-list-prop*:
assumes $\exists\ x \in set\ xs.\ P\ x$
shows $(\exists\ ys\ x\ zs.\ (xs = ys \ominus (x \odot zs) \vee xs = \langle x\rangle \vee xs = x \odot zs \vee$
$xs = ys \ominus \langle x\rangle) \wedge P\ x)$
using *assms*
proof (*induct xs*)
case (*St x*)
then show *?case* **by** *auto*
next
  case (*Cons x1a xs*)
  then show *?case*
  by (*meson interval-split-interval*)
qed

lemma *interval-split-interval-propE*:
 assumes $\exists\ x\in set\ xs.\ P\ x$
 obtains *ys x zs* **where** $xs = ys \ominus (x \odot zs) \vee xs = \langle x\rangle \vee xs = x \odot zs \vee$
$xs = ys \ominus \langle x\rangle$ **and** $P\ x$
using *interval-list-prop* [*OF assms*] **by** *blast*

lemma *interval-split-interval-first-prop*:
assumes $\exists\ x \in set\ xs.\ P\ x$
 shows $(\exists\ ys\ x\ zs.\ ((xs = ys \ominus (x \odot zs) \wedge (\forall y \in set\ ys.\ \neg\ P\ y)) \vee$
$xs = \langle x\rangle \vee xs = x \odot zs \vee$
$(xs = ys \ominus \langle x\rangle \wedge (\forall y \in set\ ys.\ \neg\ P\ y))) \wedge P\ x$
)
using *assms*
proof (*induct xs*)
case (*St x*)
then show *?case* **by** *auto*
next
  case (*Cons x1a xs*)
  then show *?case* **proof** (*cases P x1a*)
  case *True*
  then show *?thesis* **by** *blast*
  next
  case *False*
  then show *?thesis*
   proof −
    have *1*: $\exists\ x \in set\ xs.\ P\ x$
     using *Cons.prems False* **by** *auto*
    have *2*: $\neg\ P\ x1a$
      by (*simp add*: *False*)
    have *3*: $\exists ys\ x\ zs.$
      $(xs = ys \ominus x \odot zs \wedge (\forall y{\in}interval.set\ ys.\ \neg\ P\ y) \vee$
       $xs = \langle x\rangle \vee xs = x \odot zs \vee xs = ys \ominus \langle x\rangle \wedge (\forall y{\in}interval.set\ ys.\ \neg\ P\ y)) \wedge$
      $P\ x$
     using *1 Cons.hyps* **by** *blast*

obtain *ys x zs* **where** *4*: $(xs = ys \ominus x \odot zs \wedge (\forall y{\in}interval.set\ ys.\ \neg\ P\ y) \vee$
$xs = \langle x\rangle \vee xs = x \odot zs \vee xs = ys \ominus \langle x\rangle \wedge (\forall y{\in}interval.set\ ys.\ \neg\ P\ y)) \wedge$
$P\ x$
using *3* **by** *auto*
have *5*: $(xs = ys \ominus x \odot zs \wedge (\forall y{\in}interval.set\ ys.\ \neg\ P\ y)) \longrightarrow$
$(x1a \odot xs = (x1a \odot ys) \ominus x \odot zs \wedge (\forall y{\in}interval.set\ (x1a{\odot}ys).\ \neg\ P\ y))$
$\vee\ (x1a{\odot}\ xs) = \langle x\rangle \vee (x1a{\odot}\ xs) = x \odot zs \vee$
$x1a \odot xs = (x1a{\odot}ys) \ominus \langle x\rangle \wedge (\forall y{\in}interval.set\ (x1a{\odot}ys).\ \neg\ P\ y)$
using *False* **by** *auto*
have *6*: $xs = ys \ominus \langle x\rangle \wedge (\forall y{\in}interval.set\ ys.\ \neg\ P\ y) \longrightarrow$
$(x1a \odot xs = (x1a \odot ys) \ominus x \odot zs \wedge (\forall y{\in}interval.set\ (x1a{\odot}ys).\ \neg\ P\ y)) \vee$
$x1a \odot xs = (x1a{\odot}ys) \ominus \langle x\rangle \wedge (\forall y{\in}interval.set\ (x1a{\odot}ys).\ \neg\ P\ y) \vee$
$(x1a{\odot}\ xs) = \langle x\rangle \vee (x1a{\odot}\ xs) = x \odot zs$
using *False* **by** *auto*
have *7*: *P x*
using *4* **by** *auto*
have *8*: $xs = \langle x\rangle \longrightarrow$
$(x1a \odot xs = (x1a \odot ys) \ominus x \odot zs \wedge (\forall y{\in}interval.set\ (x1a{\odot}ys).\ \neg\ P\ y)) \vee$
$x1a \odot xs = \langle x1a\rangle \ominus \langle x\rangle \wedge (\forall y{\in}interval.set\ (\langle x1a\rangle).\ \neg\ P\ y) \vee$
$(x1a{\odot}\ xs) = \langle x\rangle \vee (x1a{\odot}\ xs) = x \odot zs$

by (*simp add*: *False*)
have *9*: $xs = x \odot zs \longrightarrow$
$(x1a \odot xs = (\langle x1a\rangle) \ominus x \odot zs \wedge (\forall y{\in}interval.set\ (\langle x1a\rangle).\ \neg\ P\ y)) \vee$
$x1a \odot xs = (x1a{\odot}ys) \ominus \langle x\rangle \wedge (\forall y{\in}interval.set\ (x1a{\odot}ys).\ \neg\ P\ y) \vee$
$(x1a{\odot}\ xs) = \langle x\rangle \vee (x1a{\odot}\ xs) = x \odot zs$
by (*simp add*: *False*)
show *?thesis*
using *4 5 6 8 9* **by** *blast*
qed
qed
qed

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

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

lemma *interval-split-interval-last-prop*:
assumes $\exists\ x \in set\ xs.\ P\ x$

shows $(\exists\ ys\ x\ zs.\ ((xs = ys \ominus (x \odot zs) \wedge (\forall y \in set\ zs.\ \neg\ P\ y)) \vee$
$xs = \langle x\rangle \vee xs = x \odot zs \wedge (\forall y \in set\ zs.\ \neg\ P\ y) \vee$
$(xs = ys \ominus \langle x\rangle)) \wedge P\ x$
)
using *assms*
proof (*induct xs rule*: *interval-rev-induct*)
case (*St y*)
then show *?case* **by** *auto*
next
case (*snoc x1a xs*)
then show *?case* **proof** (*cases P x1a*)
case *True*
then show *?thesis* **by** *blast*
next
case *False*
then show *?thesis*
proof −
have *1*: $\exists\ x \in set\ xs.\ P\ x$
using *False snoc.prems* **by** *auto*
have *2*: $\neg\ P\ x1a$
by (*simp add*: *False*)
have *3*: $\exists ys\ x\ zs.$
$(xs = ys \ominus x \odot zs \wedge (\forall y{\in}interval.set\ zs.\ \neg\ P\ y) \vee$
$xs = \langle x\rangle \vee xs = x \odot zs \wedge (\forall y{\in}interval.set\ zs.\ \neg\ P\ y) \vee xs = ys \ominus \langle x\rangle) \wedge$
$P\ x$
using *1 snoc.hyps* **by** *blast*
obtain *ys x zs* **where** *4*: $(xs = ys \ominus x \odot zs \wedge (\forall y{\in}interval.set\ zs.\ \neg\ P\ y) \vee$
$xs = \langle x\rangle \vee xs = x \odot zs \wedge (\forall y{\in}interval.set\ zs.\ \neg\ P\ y) \vee xs = ys \ominus \langle x\rangle) \wedge$
$P\ x$
using *3* **by** *auto*
have *5*: $P\ x$
using *4* **by** *auto*
have *6*: $(xs = ys \ominus x \odot zs \wedge (\forall y{\in}interval.set\ zs.\ \neg\ P\ y)) \longrightarrow$
$(xs \ominus \langle x1a\rangle = ys \ominus x \odot (zs \ominus \langle x1a\rangle) \wedge (\forall y{\in}interval.set\ (zs \ominus \langle x1a\rangle).\ \neg\ P\ y) \vee$
$xs \ominus \langle x1a\rangle = \langle x\rangle \vee$
$xs \ominus \langle x1a\rangle = x \odot (zs \ominus \langle x1a\rangle) \wedge (\forall y{\in}interval.set\ (zs{\ominus}\ \langle x1a\rangle).\ \neg\ P\ y) \vee$
$xs \ominus \langle x1a\rangle = ys \ominus \langle x\rangle)$

by (*simp add*: *False*)
have *7*: $xs = \langle x\rangle \longrightarrow$
$(xs \ominus \langle x1a\rangle = ys \ominus x \odot (\ \langle x1a\rangle) \wedge (\forall y{\in}interval.set\ (\ \langle x1a\rangle).\ \neg\ P\ y) \vee$
$xs \ominus \langle x1a\rangle = \langle x\rangle \vee$
$xs \ominus \langle x1a\rangle = x \odot (\ \langle x1a\rangle) \wedge (\forall y{\in}interval.set\ (\ \langle x1a\rangle).\ \neg\ P\ y) \vee$
$xs \ominus \langle x1a\rangle = ys \ominus \langle x\rangle)$

using *False* **by** *auto*
have *8*: $xs = x \odot zs \wedge (\forall y{\in}interval.set\ zs.\ \neg\ P\ y) \longrightarrow$
$(xs \ominus \langle x1a\rangle = ys \ominus x \odot (zs \ominus \langle x1a\rangle) \wedge (\forall y{\in}interval.set\ (zs \ominus \langle x1a\rangle).\ \neg\ P\ y) \vee$
$xs \ominus \langle x1a\rangle = \langle x\rangle \vee$
$xs \ominus \langle x1a\rangle = x \odot (zs \ominus \langle x1a\rangle) \wedge (\forall y{\in}interval.set\ (zs{\ominus}\ \langle x1a\rangle).\ \neg\ P\ y) \vee$

```
          xs ⊖ ⟨x1a⟩ = ys ⊖ ⟨x⟩)

        by (simp add: False)
    have 9: xs = ys ⊖ ⟨x⟩ ⟶
          (xs ⊖ ⟨x1a⟩ = ys ⊖ x ⊙ ( ⟨x1a⟩) ∧ (∀ y∈interval.set ( ⟨x1a⟩). ¬ P y) ∨
          xs ⊖ ⟨x1a⟩ = ⟨x⟩ ∨
          xs ⊖ ⟨x1a⟩ = x ⊙ ( ⟨x1a⟩) ∧ (∀ y∈interval.set ( ⟨x1a⟩). ¬ P y) ∨
          xs ⊖ ⟨x1a⟩ = ys ⊖ ⟨x⟩)

      by (simp add: False)
    show ?thesis
    using 4 6 7 8 9 by (metis (full-types))
   qed
  qed
qed

lemma interval-split-interval-last-propE:
 assumes ∃ x ∈ set xs. P x
 obtains ys x zs where ((xs = ys ⊖ (x ⊙ zs) ∧ (∀ y ∈ set zs. ¬ P y)) ∨
        xs = ⟨x⟩ ∨ xs = x ⊙ zs ∧ (∀ y ∈ set zs. ¬ P y) ∨
        (xs = ys ⊖ ⟨x⟩)) and P x
 using interval-split-interval-last-prop [OF assms] by blast

lemma interval-split-interval-last-prop-iff:
 (∃ x ∈ set xs. P x) ⟷
  (∃ ys x zs. ((xs = ys ⊖ (x ⊙ zs) ∧ (∀ y ∈ set zs. ¬ P y)) ∨
        xs = ⟨x⟩ ∨ xs = x ⊙ zs ∧ (∀ y ∈ set zs. ¬ P y) ∨
        (xs = ys ⊖ ⟨x⟩)) ∧ P x
  )
by (rule, erule interval-split-interval-last-prop, auto)

lemma interval-nth-and-set:
 x ∈ set xs = (∃ i≤intlen xs. (nth xs i) =x)
proof (induct xs)
case (St x)
then show ?case by auto
next
case (Cons x1a xs)
then show ?case
by (metis Suc-le-mono insert-iff interval.simps(16) interval-intlen-cons interval-intlen-gr-zero
   interval-nth-Suc interval-nth-cons interval-nth-zero intlen.simps(2) le-diff-conv
   neq0-conv plus-1-eq-Suc)
qed

lemma interval-card-intlen:
 card (set xs) ≤ intlen xs+1
proof (induct xs)
case (St x)
then show ?case by simp
next
```

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

lemma *set-nth*:
set xs = { (*nth xs k*) | *k*. *k*≤ *intlen xs*}
proof (*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof *auto*
show *set xs* = {*nth xs k* |*k*. *k* ≤ *intlen xs*} ⟹
∃ *k*. *x1a* = (*case k of 0* ⇒ *x1a* | *Suc x* ⇒ *nth xs x*) ∧ *k* ≤ *Suc* (*intlen xs*)
by *force*
show ⋀*k*. *set xs* = {*nth xs k* |*k*. *k* ≤ *intlen xs*} ⟹
k ≤ *intlen xs* ⟹
∃ *ka*. *nth xs k* = (*case ka of 0* ⇒ *x1a* | *Suc x* ⇒ *nth xs x*) ∧ *ka* ≤ *Suc* (*intlen xs*)
by *force*
show ⋀*k*. *set xs* = {*nth xs k* |*k*. *k* ≤ *intlen xs*} ⟹
(*case k of 0* ⇒ *x1a* | *Suc x* ⇒ *nth xs x*) ≠ *x1a* ⟹
k ≤ *Suc* (*intlen xs*) ⟹
∃ *ka*. (*case k of 0* ⇒ *x1a* | *Suc x* ⇒ *nth xs x*) = *nth xs ka* ∧ *ka* ≤ *intlen xs*
by (*metis* (*mono-tags*, *lifting*) *Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv*)
qed
qed

lemma *prefix-set*:
assumes *k*≤ *intlen xs*
shows *set* (*prefix k xs*) = {(*nth xs i*) |*i*. *i*≤*k*}
proof −
have *1*: *set* (*prefix k xs*) = { (*nth* (*prefix k xs*) *i*) | *i*. *i*≤ *intlen* (*prefix k xs*)}
by (*simp add*: *set-nth*)
have *2*: *k*≤ *intlen xs*
using *assms* **by** *auto*
have *3*: { (*nth* (*prefix k xs*) *i*) | *i*. *i*≤ *intlen* (*prefix k xs*)} =
{ (*nth xs i*) | *i*. *i*≤ *k*}
using *2* **by** (*metis interval-nth-prefix interval-prefix-length-good*)
show *?thesis* **using** *1 3* **by** *auto*
qed

lemma *suffix-set*:
assumes *k*≤ *intlen xs*
shows *set* (*suffix k xs*) = {(*nth xs* (*k*+*i*)) |*i*. *i*≤*intlen xs* − *k*}
proof −
have *1*: *set* (*suffix k xs*) = { (*nth* (*suffix k xs*) *i*) | *i*. *i*≤ *intlen* (*suffix k xs*)}
by (*simp add*: *set-nth*)
have *2*: *k* ≤ *intlen xs*
using *assms* **by** *auto*

```
 have 3: { (nth (suffix k xs) i) | i. i≤ intlen (suffix k xs)} =
         { (nth xs (k+i)) | i. i≤ intlen xs − k}
   using 2 by (metis interval-nth-suffix interval-suffix-length-good)
 show ?thesis  using 1 3 by auto
qed

lemma suffix-set-a:
assumes k≤ intlen xs
 shows    set (suffix k xs) = {(nth xs i) |i. k≤ i ∧ i≤intlen xs }
proof −
 have 1: set (suffix k xs) = { (nth xs (k+i)) | i. i≤ intlen xs − k}
   using assms suffix-set by blast
 have 2: ∀ x ∈ { (nth xs (k+i)) | i. i≤ intlen xs − k} .
           x ∈ {(nth xs i) |i. k≤ i ∧ i≤intlen xs }
    using assms by auto
 have 3: ∀ x ∈ {(nth xs i) |i. k≤ i ∧ i≤intlen xs } .
           x ∈ { (nth xs (k+i)) | i. i≤ intlen xs − k}
     using nat-le-iff-add by force
 have 4: { (nth xs (k+i)) | i. i≤ intlen xs − k} =
         {(nth xs i) |i. k≤ i ∧ i≤intlen xs }
  using 2 3 by blast
 show ?thesis by (simp add: 1 4)
qed

lemma sub-interval-set:
 assumes k ≤ n
         n ≤ intlen xs
 shows    set (sub k n xs) = { (nth xs (k+i)) | i. i ≤ n−k }
proof −
 have 1:  set (sub k n xs) = { (nth (sub k n xs) i) | i. i ≤ intlen (sub k n xs) }
    by (simp add: set-nth)
 have 2:  k≤ n
   using assms by auto
 have 3:  n ≤ intlen xs
   using assms by auto
 have 4:  { (nth (sub k n xs) i) | i. i ≤ intlen (sub k n xs) } =
          { (nth xs (k+i) ) | i. i ≤ n−k }
   using 2 3 by force
 show ?thesis by (simp add: 1 4)
qed


lemma sub-interval-set-a:
 assumes k ≤ n
         n ≤ intlen xs
 shows    set (sub k n xs) = { (nth xs i) | i. k ≤ i ∧ i ≤ n }
proof −
 have 1: set (sub k n xs) = { (nth xs (k+i)) | i. i ≤ n−k }
   using assms sub-interval-set by blast
 have 2: ∀ x ∈ { (nth xs (k+i)) | i. i ≤ n−k } .
```

$x \in \{ (nth\ xs\ i) \mid i.\ k \leq i \wedge i \leq n \}$
using *assms* **by** *auto*
have *3*: $\forall\ x \in \{ (nth\ xs\ i) \mid i.\ k \leq i \wedge i \leq n \}.$
$x \in \{ (nth\ xs\ (k+i)) \mid i.\ i \leq n-k \}$
using *assms* **using** *le-Suc-ex* **by** *auto fastforce*
have *4*: $\{ (nth\ xs\ (k+i)) \mid i.\ i \leq n-k \} = \{ (nth\ xs\ i) \mid i.\ k \leq i \wedge i \leq n \}$
using *2 3* **by** *blast*
show *?thesis* **by** (*simp add*: *1 4*)
qed

lemma *nth-set*:
assumes $k \leq$ *intlen xs*
shows (*nth xs k*) $\in$ *set xs*
using *assms*
by (*meson interval-nth-and-set*)

end

2 Finite ITL Semantics

theory *Semantics*
imports *Interval HOL−TLA.Intensional*
begin

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

2.1 Types of Formulas

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

type-synonym ($'a$,$'b$) *formfun* = $'a$ *interval* $\Rightarrow$ $'b$
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. Note chopstar is a derived operator, i.e., it is defined recursively in terms of chop.

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

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

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

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

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

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

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

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

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

-chop-d ⇌ *CONST chop-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*)

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* $\Rightarrow$ *lift* ((*dm* -) [*88*] *87*)

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

translations
-empty-d $\rightleftharpoons$ *CONST empty-d*
-dm-d $\rightleftharpoons$ *CONST dm-d*

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

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

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

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

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

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

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

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

primrec *power-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *nat* $\Rightarrow$ *'a formula*
where *pow-0* : (*power-d F 0*) = *LIFT*(*empty*)
| *pow-Suc*: (*power-d F* (*Suc n*)) = *LIFT*((*F*);(*power-d F n*))

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

-ifthenelse-d :: [*lift*,*lift*,*lift*] ⇒ *lift* ((*if* $_i$ *- then - else -*) [*88*,*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*)
-power-d :: [*lift*,*nat*] ⇒ *lift* ((*power - -*) [*88*,*88*] *87*)

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

definition *len-d* :: *nat* ⇒ (*'a*::*world*) *formula*
where *len-d n* ≡ *LIFT*(*power skip n*)

definition *powerstar-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *powerstar-d F* ≡ *LIFT*(∃ *k*. *power F k*)

syntax
-len-d :: *nat* ⇒ *lift* ((*len -*) [*88*] *87*)
-powerstar-d :: *lift* ⇒ *lift* ((*powerstar -*) [*85*] *85*)

syntax (*ASCII*)
-len-d :: *nat* ⇒ *lift* ((*len -*) [*88*] *87*)
-powerstar-d :: *lift* ⇒ *lift* ((*powerstar -*) [*85*] *85*)

translations
-len-d ⇌ *CONST len-d*
-powerstar-d ⇌ *CONST powerstar-d*

definition *chopstar-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *chopstar-d F* ≡ *LIFT*(*powerstar* (*F* ∧ *more*))

syntax
-chopstar-d :: *lift* ⇒ *lift* ((*-**) [*85*] *85*)

syntax (*ASCII*)

-chopstar-d :: *lift* ⇒ *lift* ((*chopstar* -) [*85*] *85*)

translations
-chopstar-d ⇌ *CONST chopstar-d*

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

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

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

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

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

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

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

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

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

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

definition *prev-assign-d* :: ('*a*::*world*,'*b*) *stfun* ⇒ ('*a*,'*b*) *formfun* ⇒ '*a formula*
where *prev-assign-d v e* ≡ *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 above operators have the expected semantics.

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

lemma *chop-defs* :

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

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

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

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

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

lemma *da-defs* :
$(w \models da\ F) = (\exists\ n\ na.\ n{+}na \le intlen\ w \wedge ((sub\ n\ (n{+}na)\ w) \models F))$
proof (*auto simp add*: *Semantics.da-d-def chop-defs*)
show $\bigwedge n\ na.$
$n \le intlen\ w \Longrightarrow$
$na \le intlen\ w - n \Longrightarrow F\ (prefix\ na\ (suffix\ n\ w)) \Longrightarrow$
$\exists n\ na.\ n + na \le intlen\ w \wedge F\ (sub\ n\ (n + na)\ w)$
by (*metis Interval.sub-def Nat.le-diff-conv2 add.commute add-diff-cancel-left'*)
show $\bigwedge n\ na.$
$n + na \le intlen\ w \Longrightarrow F\ (sub\ n\ (n + na)\ w) \Longrightarrow$
$\exists n \le intlen\ w.\ \exists na \le intlen\ w - n.\ F\ (prefix\ na\ (suffix\ n\ w))$
by (*metis Interval.sub-def add-leD1 interval-intlen-sub interval-pref-intlen-bound interval-suffix-length le-add1*)
qed

lemma *ba-defs* :
$(w \models ba\ F) = (\forall\ n\ na.\ \ na{+}n \le intlen\ w \longrightarrow ((sub\ n\ (n{+}na)\ w) \models F))$
by (*auto simp add*: *ba-d-def da-defs*)

lemma *next-defs* :
$(w \models \bigcirc\ F) = (intlen\ w > 0 \wedge ((suffix\ 1\ w) \models F)\ )$
using *Suc-le-eq min.absorb1* **by** (*simp add*: *next-d-def chop-defs skip-defs*) *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*

neq0-conv zero-neq-one)

lemma *wprev-defs* :
 $(w \models wprev\ F) = (intlen\ w = 0 \lor ((prefix\ ((intlen\ w)-1)\ w) \models F))$
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* :
 $(w \models init\ F) = ((prefix\ 0\ w) \models F)$
using *min.absorb1* **by** (*simp add*: *init-d-def empty-defs chop-defs*) *force*

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

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

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

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

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

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

lemma *len-defs* :
 $(w \models len\ n) = (intlen\ w = n)$
proof
 (*induct n arbitrary*: *w*)
 case *0*
 then show *?case* **by** (*simp add*: *len-d-def empty-defs*)
 next

case (*Suc n*)
then show *?case* **by** (*simp add*: *len-d-def chop-defs skip-defs*) *fastforce*
qed

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

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

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

lemma *prev-assign-defs* :
assumes $intlen\ s > 0$
shows $(s \models v =: e) = v\ (Interval.nth\ s\ ((intlen\ s) - 1)) = e\ s$
using *assms* **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* :
$(s \models v\ gets\ e) = (\forall\ i < intlen\ s.\ v\ (Interval.nth\ s\ (Suc\ i)) = e\ (sub\ i\ (i{+}1)\ s)\ )$
using *Suc-le-eq min.absorb1 add-le-imp-le-diff interval-prefix-suffix-intlen-good*
by (*auto simp add*: *gets-d-def keep-d-def ba-defs skip-defs sub-def temporal-assign-defs*)
fastforce

lemma *stable-defs-helpa*:
assumes $(\forall i < intlen\ s.\ v\ (Interval.nth\ s\ (Suc\ i)) = v\ (Interval.nth\ s\ i))$
$i \leq intlen\ s$
shows $(v\ (Interval.nth\ s\ i) = v\ (Interval.nth\ s\ 0))$
using *assms*

```
proof (induct s arbitrary:i)
case (St x)
then show ?case by simp
next
case (Cons x1a s)
then show ?case
  proof (cases i)
  case 0
  then show ?thesis by blast
  next
  case (Suc nat)
  then show ?thesis
     by (metis Cons.hyps Cons.prems(1) Cons.prems(2) Suc-le-mono Suc-mono interval-nth-Suc
         interval-nth-zero intlen.simps(2) plus-1-eq-Suc zero-less-Suc)
  qed
qed


lemma stable-defs-helpb:
assumes (∀ i≤intlen s. v (Interval.nth s i) = v (Interval.nth s 0))
        i < intlen s
 shows  v (Interval.nth s (Suc i)) = v (Interval.nth s i)
using assms
proof (induct s arbitrary:i)
case (St x)
then show ?case by simp
next
case (Cons x1a s)
then show ?case
   proof (cases i)
   case 0
   then show ?thesis using Suc-leI Cons.prems(1) Cons.prems(2) by blast
   next
   case (Suc nat)
   then show ?thesis using Cons.prems(1) Cons.prems(2) Suc-leI less-imp-le-nat by presburger
   qed
qed

lemma stable-defs-help:
 (∀ i<intlen s. v (Interval.nth s (Suc i)) = v (Interval.nth s i)) ⟷
  (∀ i≤intlen s. v (Interval.nth s i) = v (Interval.nth s 0))
proof −
 have 1: (∀ i<intlen s. v (Interval.nth s (Suc i)) = v (Interval.nth s i)) ⟶
         (∀ i≤intlen s. v (Interval.nth s i) = v (Interval.nth s 0))
  using stable-defs-helpa by auto
 have 2: (∀ i≤intlen s. v (Interval.nth s i) = v (Interval.nth s 0)) ⟶
         (∀ i<intlen s. v (Interval.nth s (Suc i)) = v (Interval.nth s i))
   using stable-defs-helpb by blast
 show ?thesis using 1 2 by blast
qed
```

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

lemma *padded-defs* :
$(s \models padded\ v) = ((\forall\ \ i{<}\ intlen\ s.\ (v\ (nth\ s\ i)) = (v\ (nth\ s\ 0))) \vee intlen\ s = 0)$
proof (*simp add*: *padded-d-def stable-defs chop-defs skip-defs empty-defs*)
show $((\exists\, n{\leq}intlen\ s.$
$(\forall\, i.\ i \leq n \wedge i \leq intlen\ s \longrightarrow v\ (nth\ s\ i) = v\ (nth\ s\ 0)) \wedge intlen\ s - n = Suc\ 0) \vee$
$intlen\ s = 0) =$
$((\forall\, i{<}intlen\ s.\ v\ (Interval.nth\ s\ i) = v\ (Interval.nth\ s\ 0)) \vee intlen\ s = 0)$
proof *rule+*
show $\bigwedge i.\ (\exists\, n{\leq}intlen\ s.$
$(\forall\, i.\ i \leq n \wedge i \leq intlen\ s \longrightarrow v\ (nth\ s\ i) = v\ (nth\ s\ 0)) \wedge intlen\ s - n = Suc\ 0) \vee$
$intlen\ s = 0 \Longrightarrow$
$i < intlen\ s \Longrightarrow v\ (nth\ s\ i) = v\ (nth\ s\ 0)$
by (*metis One-nat-def Suc-leI Suc-le-mono le-add-diff-inverse2 less-imp-le-nat not-less-zero plus-1-eq-Suc*)
show $(\forall\, i{<}intlen\ s.\ v\ (nth\ s\ i) = v\ (nth\ s\ 0)) \vee intlen\ s = 0 \Longrightarrow$
$(\exists\, n{\leq}intlen\ s.$
$(\forall\, i.\ i \leq n \wedge i \leq intlen\ s \longrightarrow v\ (nth\ s\ i) = v\ (nth\ s\ 0)) \wedge intlen\ s - n = Suc\ 0) \vee$
$intlen\ s = 0$
by (*metis Suc-leI Suc-pred diff-diff-cancel diff-le-self gr-zeroI le-imp-less-Suc*)
qed
qed

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

2.5 Soundness of Finite ITL Axioms

2.5.1 ChopAssoc

lemma *ChopAssocSemHelpa*:
assumes $(\exists\, i\ ia\ .\ i{\leq}intlen\ \sigma \wedge ia \leq intlen\ \sigma - i \wedge (prefix\ i\ \sigma \models f) \wedge$
$(prefix\ ia\ (suffix\ i\ \sigma) \models g) \wedge (suffix\ (ia + i)\ \sigma \models h))$
shows $(\exists\, j\ ja\ .\ j \leq intlen\ \sigma \wedge ja \leq j \wedge (prefix\ ja\ (prefix\ j\ \sigma) \models f) \wedge$
$(suffix\ ja\ (prefix\ j\ \sigma) \models g) \wedge (suffix\ j\ \sigma \models h))$
proof −
have *1*: $(\exists\, i\ ia\ .\ i{\leq}intlen\ \sigma \wedge ia \leq intlen\ \sigma - i \wedge (prefix\ i\ \sigma \models f) \wedge$
$(prefix\ ia\ (suffix\ i\ \sigma) \models g) \wedge (suffix\ (ia + i)\ \sigma \models h))$
using *assms* **by** *auto*
obtain *i ia* **where** *2*: $i{\leq}intlen\ \sigma \wedge ia \leq intlen\ \sigma - i \wedge (prefix\ i\ \sigma \models f) \wedge$
$(prefix\ ia\ (suffix\ i\ \sigma) \models g) \wedge (suffix\ (ia + i)\ \sigma \models h)$
using *1* **by** *auto*
have *3*: $(suffix\ (ia{+}i)\ \sigma \models h)$
using *2* **by** *auto*

have *4*: *ia* +*i* ≤*intlen* σ
  using *2 Nat.le-diff-conv2* **by** *blast*
have *5*: *i* ≤ *ia*+*i*
  by *simp*
have *6*: (*suffix i* (*prefix* (*ia* +*i*) σ) ⊨ *g*)
  using *2 4 interval-suffix-prefix-swap* **by** *force*
have *7*: (*prefix i* (*prefix* (*ia* +*i*) σ) ⊨ *f*)
  by (*simp add*: *2 add.commute*)
show *?thesis* **using** *2 4 5 6 7* **by** *blast*
qed

lemma *ChopAssocSemHelpb*:
assumes (∃ *j ja* . *j* ≤*intlen* σ ∧ *ja* ≤ *j* ∧ (*prefix ja* (*prefix j* σ) ⊨ *f*) ∧
  (*suffix ja* (*prefix j* σ) ⊨ *g*) ∧ (*suffix j* σ ⊨ *h*))
shows (∃ *i ia* . *i*≤*intlen* σ ∧ *ia* ≤ *intlen* σ −*i* ∧ (*prefix i* σ ⊨ *f*) ∧
  (*prefix ia* (*suffix i* σ) ⊨ *g*) ∧ (*suffix* (*ia* + *i*) σ ⊨ *h*))
proof −
have *1*: (∃ *j ja* . *j* ≤*intlen* σ ∧ *ja* ≤ *j* ∧ (*prefix ja* (*prefix j* σ) ⊨ *f*) ∧
  (*suffix ja* (*prefix j* σ) ⊨ *g*) ∧ (*suffix j* σ ⊨ *h*))
 using *assms* **by** *auto*
obtain *j ja* **where** *2*: *j* ≤*intlen* σ ∧ *ja* ≤ *j* ∧ (*prefix ja* (*prefix j* σ) ⊨ *f*) ∧
  (*suffix ja* (*prefix j* σ) ⊨ *g*) ∧ (*suffix j* σ ⊨ *h*)
using *1* **by** *auto*
have *3*: *ja*≤*intlen* σ
 using *2 le-trans* **by** *blast*
have *4*: *j* −*ja* ≤ *intlen* σ −*ja*
 by (*simp add*: *2 diff-le-mono*)
have *5*: (*prefix ja* σ ⊨ *f*)
  by (*metis 2 interval-pref-pref-3 le-add-diff-inverse*)
have *6*: (*prefix* (*j* −*ja*) (*suffix ja* σ) ⊨ *g*)
 by (*simp add*: *2 interval-suffix-prefix-swap*)
have *7*: (*suffix* ((*j* −*ja*) + *ja*) σ ⊨ *h*)
 by (*simp add*: *2*)
show *?thesis* **using** *3 4 5 6 7* **by** *blast*
qed

lemma *ChopAssocSemHelp*:
(∃ *i ia* . *i*≤*intlen* σ ∧ *ia* ≤ *intlen* σ −*i* ∧ (*prefix i* σ ⊨ *f*) ∧
  (*prefix ia* (*suffix i* σ) ⊨ *g*) ∧ (*suffix* (*ia* + *i*) σ ⊨ *h*)) =
 (∃ *j ja* . *j* ≤*intlen* σ ∧ *ja* ≤ *j* ∧ (*prefix ja* (*prefix j* σ) ⊨ *f*) ∧
  (*suffix ja* (*prefix j* σ) ⊨ *g*) ∧ (*suffix j* σ ⊨ *h*))
using *ChopAssocSemHelpa*[*of* σ *f g h*]
   ChopAssocSemHelpb[*of* σ *f g h*] **by** *auto*

lemma *ChopAssocSemHelp2*:
 (σ ⊨ *f* ; (*g* ; *h*)) = (σ ⊨ (*f*;*g*);*h*)
proof −
have (σ ⊨ *f* ; (*g* ; *h*)) =
   ((∃ *i*≤*intlen* σ. (*prefix i* σ ⊨ *f*) ∧ (∃ *ia*≤*intlen* (*suffix i* σ).
    (*prefix ia* (*suffix i* σ) ⊨ *g*) ∧ (*suffix* (*ia* + *i*) σ ⊨ *h*))))

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

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

2.5.2 OrChopImp

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

2.5.3 ChopOrImp

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

2.5.4 EmptyChop

lemma *EmptyChopSem*:
$(\sigma \models empty\ ;\ f = f\ )$
using *min.absorb1* **by** (*simp add*: *empty-defs chop-defs*) *force*

2.5.5 ChopEmpty

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

2.5.6 StateImpBi

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

2.5.7 NextImpNotNextNot

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

2.5.8 BiBoxChopImpChop

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

2.5.9 BoxInduct

lemma *box-induct-help-1* :
assumes $(\sigma \models f)$
  $(\forall i.\ Suc\ 0 \leq intlen\ \sigma - i \longrightarrow$
  $i \leq intlen\ \sigma \longrightarrow (suffix\ i\ \sigma \models f) \longrightarrow (suffix\ (Suc\ i)\ \sigma \models f))$
shows $(\forall\ j.\ j \leq intlen\ \sigma \longrightarrow (suffix\ j\ \sigma \models f))$
proof
  fix *j*
  show $j \leq intlen\ \sigma \longrightarrow (suffix\ j\ \sigma \models f)$
   using *assms*
   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*:
  $(\sigma \models \Box\ (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box\ f)$
proof –
 have *1*: $(\sigma \models \Box\ (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box\ f) =$
  $((\forall n \leq intlen\ \sigma.\ f\ (suffix\ n\ \sigma) \longrightarrow intlen\ \sigma = n \vee f\ (suffix\ (Suc\ n)\ \sigma)) \wedge f\ \sigma \longrightarrow$
  $(\forall n \leq intlen\ \sigma.\ f\ (suffix\ n\ \sigma)))$
   by (*simp add*: *always-defs wnext-defs*)
 from *1* **show** *?thesis* **using** *box-induct-help-1*
  by (*metis One-nat-def diff-self-eq-0 not-one-le-zero*)
qed

2.5.10 ChopStarEqv

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

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

lemma *powersem1*:
$(\sigma \models (\exists\ k.\ power\ f\ k) = (\ empty \vee (\exists\ k.\ power\ f\ (Suc\ k))))$
proof *auto*
show $\bigwedge x.\ \sigma \models (power\ f\ x) \implies \forall k.\ \neg(\sigma \models (f;power\ f\ k)\ ) \implies \sigma \models empty$
by (*metis not0-implies-Suc pow-0 pow-Suc*)
show $\sigma \models empty \implies \exists x.\ \sigma \models (power\ f\ x)$
by (*metis pow-0*)
show $\bigwedge k.\ \sigma \models (f;power\ f\ k) \implies \exists x.\ \sigma \models (power\ f\ x)$
by (*metis pow-Suc*)
qed

lemma *powersem*:
$\vdash (\exists\ k.\ power\ f\ k) = (\ \ empty \vee (f);(\exists\ k.\ (power\ f\ k)))$
proof −
have *1*: $\vdash (\exists\ k.\ power\ f\ k) = (\ empty \vee (\exists\ k.\ power\ f\ (Suc\ k)))$
using *powersem1* **by** *blast*
have *2*: $\vdash (\exists\ k.\ power\ f\ (Suc\ k)) = (\exists\ k.\ (f);power\ f\ k)$
by *simp*
have *3*: $\vdash (\exists\ k.\ (f);(power\ f\ k)) = (f);(\exists\ k.\ (power\ f\ k))$
using *ChopExist* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *PowerstarEqvSem*:
$(\sigma \models (powerstar\ f) = (empty \vee\ f;(powerstar\ f)\ ))$
proof −
have *1*: $(\sigma \models (powerstar\ f)) =$
$(\ \sigma \models (\exists\ k.\ power\ f\ k))$
by (*simp add*: *powerstar-d-def*)
have *2*: $(\ \sigma \models (\exists\ k.\ power\ f\ k)) =$
$(\sigma \models (empty \vee f;(\exists\ k.\ (power\ f\ k))))$
using *powersem* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** (*simp add*: *powerstar-d-def*)
qed

lemma *ChopstarEqvSem*:
$(\sigma \models f^\star = (empty \vee\ (f \wedge more);\ f^\star)\ )$
by (*metis PowerstarEqvSem chopstar-d-def*)

2.6 Quantification over State (Flexible) Variables

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

Intensional theory.

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 ≡ (λs. (∃ x. $s \models F\ x$))

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

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

definition *forall-state-d* :: (*'a statefun* ⇒ *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*

end

3 Fuse operator

theory *Fuse*
imports *Semantics*
begin

This theory introduces the fuse operator.

3.1 Definitions

primrec *fuse* :: *'a interval* ⇒ *'a interval* ⇒ *'a interval*
 where *fuse-St* : *fuse* ⟨*x*⟩ *ys* = *ys*
  | *fuse-Cons* : *fuse* (*x* ⊙ *xs*) *ys* = *x* ⊙ (*fuse xs ys*)

primrec *lfuse* :: *'a interval interval* ⇒ *'a interval*
where *lfuse-St* : *lfuse* ⟨*xs*⟩ = *xs*
  | *lfuse-Cons* : *lfuse* (*x* ⊙ *xs*) = *fuse x* (*lfuse xs*)

primrec *lastfirst* :: *'a interval interval* ⇒ *bool*
 where *lastfirst* ⟨*xs*⟩ = *True*
  | *lastfirst* (*xs* ⊙ *xxs*) =
   (((*intlast xs*) = (*intfirst* (*intfirst xxs*))) ∧ (*lastfirst xxs*))

3.2 Lemmas

lemma *interval-fuse-intlen* :
 assumes *intlast xs* = *intfirst ys*
 shows *intlen* (*fuse xs ys*) = (*intlen xs*) + (*intlen ys*)
using *assms* **by** (*induct xs*) *simp-all*

lemma *interval-fuse-intlen-a*:
 intlen(*fuse xs ys*) = *intlen xs* + *intlen ys*
proof
 (*induct xs arbitrary*: *ys*)
 case (*St x*)
 then show *?case* **by** *simp*
 next
 case (*Cons x1a xs*)
 then show *?case* **by** *simp*
qed

lemma *interval-fuse-nth*:
assumes *i*≤ *intlen* (*fuse xs ys*)
  intlast xs = *intfirst ys*
shows (*i* ≤ *intlen xs* ⟶ *nth* (*fuse xs ys*) *i* = *nth xs i*)
  ∧
  (*intlen xs* ≤ *i* ∧ *i*≤ *intlen* (*fuse xs ys*) ⟶ *nth* (*fuse xs ys*) *i* = *nth ys* (*i* − *intlen xs*))

using *assms*
proof
 (*induct xs arbitrary*: *i*)
 case (*St x*)
 then show *?case* **by** *simp*
 next
 case (*Cons x1a xs*)
 then show *?case*
 using *less-Suc-eq-0-disj less-Suc-eq-le* **by** *fastforce*
qed

lemma *interval-fuse-nth-a*:
assumes $j \leq$ *intlen ys*
intlast xs $=$ *intfirst ys*
shows *nth* (*fuse xs ys*) (*intlen xs* $+j$) $=$ (*nth ys j*)
using *assms*
by (*simp add*: *interval-fuse-intlen-a interval-fuse-nth*)

lemma *interval-fuse-leftneutral* :
fuse (*St* (*intfirst xs*)) *xs* $=$ *xs*
by *simp*

lemma *interval-fuse-rightneutral* :
fuse xs (*St* (*intlast xs*)) $=$ *xs*
by (*induct xs*) *simp-all*

lemma *interval-intfirst-fuse* :
assumes *intlast xs* $=$ *intfirst ys*
shows *intfirst* (*fuse xs ys*) $=$ *intfirst xs*
using *assms* **by** (*induct xs*) *simp-all*

lemma *interval-intlast-fuse* :
assumes *intlast xs* $=$ *intfirst ys*
shows *intlast* (*fuse xs ys*) $=$ *intlast ys*
using *assms* **by** (*induct xs*) *simp-all*

lemma *interval-FusionAssoc* :
assumes (*intlast xs*) $=$ (*intfirst ys*)
(*intlast ys*) $=$ (*intfirst zs*)
shows (*fuse xs* (*fuse ys zs*)) $=$ (*fuse* (*fuse xs ys*) *zs*)
using *assms* **by** (*induct xs*) *simp-all*

lemma *interval-intlast-intfirst*:
(*intlast* (*prefix i xs*)) $=$ (*intfirst* (*suffix i xs*))
using *interval-intlast-intfirst* **by** *blast*

lemma *interval-prefix-fuse* :
assumes *intlast xs* $=$ *intfirst ys*
shows (*prefix* (*intlen xs*) (*fuse xs ys*)) $=$ *xs*
using *assms* **by** (*induct xs arbitrary*: *ys*) *simp-all*

lemma *interval-suffix-fuse* :
assumes *intlast xs* $=$ *intfirst ys*
shows (*suffix* (*intlen xs*) (*fuse xs ys*)) $=$ *ys*
using *assms* **by** (*induct xs arbitrary*: *ys*) *simp-all*

lemma *interval-fuse-prefix-suffix-intlen* :
assumes $n \leq$ *intlen xs*
shows *intlen* (*fuse* (*prefix n xs*) (*suffix n xs*)) $=$ *intlen xs*

using *assms*
by (*metis interval-fuse-intlen-a interval-prefix-length-good interval-suffix-length-good le-add-diff-inverse*)

lemma *interval-fuse-prefix-suffix-nth* :
assumes $n \leq$ *intlen xs*
$i \leq$ *intlen xs*
shows *nth* (*fuse* (*prefix n xs*) (*suffix n xs*)) $i =$ *nth xs i*
using *assms*
by (*metis interval-fuse-prefix-suffix-intlen interval-fuse-nth interval-intlast-intfirst interval-nth-prefix interval-nth-suffix interval-prefix-length-good le-cases nat-add-left-cancel-le ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)

lemma *interval-fuse-prefix-suffix*:
assumes $n \leq$ *intlen xs*
shows *fuse* (*prefix n xs*) (*suffix n xs*) = *xs*
using *assms*
by (*simp add*: *interval-fuse-prefix-suffix-intlen interval-fuse-prefix-suffix-nth interval-eq-nth-eq*)

lemma *interval-chop-fuse-1* :
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = fuse\ \sigma 1\ \sigma 2\ \wedge$
$(\sigma 1\ \models f) \wedge (\sigma 2\ \models g) \wedge$
$(intlast\ \sigma 1 = intfirst\ \sigma 2)) =$
$(\exists\ i.\ 0 \leq i \wedge i \leq intlen\ \sigma \wedge (\ prefix\ i\ \sigma\ \models f) \wedge (suffix\ i\ \sigma\ \models g))$
by (*metis interval-fuse-intlen interval-fuse-prefix-suffix interval-intlast-intfirst interval-intlen-gr-zero interval-prefix-fuse interval-suffix-fuse le-add-same-cancel1*)

lemma *interval-chop-fuse-2* :
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = fuse\ \sigma 1\ \sigma 2\ \wedge$
$(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge$
$(intlast\ \sigma 1 = intfirst\ \sigma 2)) =$
$(\exists\ i \leq intlen\ \sigma.\ (prefix\ i\ \sigma) \in X \wedge (suffix\ i\ \sigma) \in Y)$
by (*metis interval-fuse-intlen interval-fuse-prefix-suffix interval-intlast-intfirst interval-prefix-fuse interval-suffix-fuse le-add1*)

lemma *interval-chop-fuse*:
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = fuse\ \sigma 1\ \sigma 2\ \wedge$
$(\sigma 1\ \models f) \wedge (\sigma 2\ \models g) \wedge$
$(intlast\ \sigma 1 = intfirst\ \sigma 2)) =$
$(\sigma\ \models f;g)$
by (*metis chop-defs interval-fuse-intlen-a interval-fuse-prefix-suffix interval-intlast-intfirst interval-prefix-fuse interval-suffix-fuse le-add1*)

lemma *interval-sub-fuse*:
assumes $k \leq n$
$n \leq m$
$m \leq$ *intlen xs*
shows *fuse* (*sub k n xs*) (*sub n m xs*) = (*sub k m xs*)
proof −
have *1*: *intlast*(*sub k n xs*) = (*nth xs n*)

using *assms interval-intlast-sub le-trans less-imp-le-nat* **by** *blast*
have *2*: *intfirst*(*sub n m xs*) = (*nth xs n*)
using *assms interval-intfirst-sub less-imp-le-nat* **by** *blast*
have *3*: *intlen*(*fuse* (*sub k n xs*) (*sub n m xs*)) = *intlen*(*sub k m xs*)
by (*metis Nat.add-diff-assoc2 assms*(*1*) *assms*(*2*) *assms*(*3*) *interval-fuse-intlen-a interval-intlen-sub le-trans ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *4*: $(\forall\ i.\ i \leq$ *intlen*(*sub k m xs*) $\longrightarrow$
(*nth* (*fuse* (*sub k n xs*) (*sub n m xs*)) *i*) = (*nth xs* (*k*+*i*)))
proof
fix *i*
show $i \leq$ *intlen* (*sub k m xs*) $\longrightarrow$ *nth* (*fuse* (*sub k n xs*) (*sub n m xs*)) *i* = *nth xs* (*k* + *i*)
proof −
have *41*: *intlen* (*sub k m xs*) = (*m*−*k*)
using *assms interval-intlen-sub le-trans less-imp-le-nat* **by** *metis*
have *42*: $i \leq$ *intlen* (*sub k m xs*) $\longrightarrow$ *nth* (*fuse* (*sub k n xs*) (*sub n m xs*)) *i* =
(*if* $i \leq$ *intlen*(*sub k n xs*) *then* (*nth* (*sub k n xs*) *i*)
else (*nth* (*sub n m xs*) (*i*−*intlen*(*sub k n xs*))))
by (*metis 1 2 3 interval-fuse-nth le-cases*)
have *43*: $i \leq (m-k) \longrightarrow$ *nth* (*fuse* (*sub k n xs*) (*sub n m xs*)) *i* =
(*if* $i \leq (n-k)$ *then* (*nth xs* (*k*+*i*)) *else* (*nth xs* (*n*+(*i*−(*n*−*k*)))))
using *42 Nat.le-diff-conv2 assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *auto*
have *44*: $i \leq (m-k) \longrightarrow$ *nth* (*fuse* (*sub k n xs*) (*sub n m xs*)) *i* =
(*nth xs* (*k*+*i*))
by (*simp add*: *43 add.commute assms less-imp-le-nat*)
show *?thesis*
by (*simp add*: *41 44*)
qed
qed
have *5*: $(\forall\ i.\ i \leq$ *intlen*(*sub k m xs*) $\longrightarrow$ (*nth* (*sub k m xs*) *i*) = (*nth xs* (*k*+*i*)))
using *assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *auto*
show *?thesis*
by (*simp add*: *3 4 5 interval-eq-nth-eq*)
qed

lemma *interval-sub-fuse-idx*:
assumes *index-sequence 0 l*
nth l (*intlen l*) = *intlen* σ
(*Suc i*) < *intlen l*
shows *fuse* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) (*sub* (*nth l* (*Suc i*)) (*nth l* (*intlen l*)) σ) =
(*sub* (*nth l i*) (*nth l* (*intlen l*)) σ)
proof −
have *1*: *intlast*(*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) = (*nth* σ (*nth l* (*Suc i*)))
by (*metis assms interval-idx-less-equal interval-idx-less-last-1 interval-intlast-sub le-add2 less-imp-le-nat plus-1-eq-Suc*)
have *2*: *intfirst*(*sub* (*nth l* (*Suc i*)) (*nth l* (*intlen l*)) σ) = (*nth* σ (*nth l* (*Suc i*)))
by (*metis assms dual-order.strict-iff-order eq-imp-le interval-idx-less-last-1 interval-intfirst-sub*)
have *3*: (*nth l i*) < (*nth l* (*Suc i*))
using *Suc-lessD assms interval-idx-less-than less-imp-le-nat* **by** *blast*
have *4*: (*nth l* (*Suc i*)) < (*nth l* (*intlen l*))

    using *assms interval-idx-less-last-1* **by** *blast*
  show *?thesis* **using** *3 4 assms* **by** (*simp add*: *interval-sub-fuse*)
qed

lemma *interval-idx-fuse-intfirst-intlast*:
  assumes *index-sequence 0 l1*
        index-sequence 0 l2
        nth l1 (*intlen l1*) = *cp*
        nth l2 (*intlen l2*) = *intlen* σ −*cp*
        cp $\leq$ *intlen* σ
        l = *fuse l1* (*map* (*shift cp*) *l2*)
  shows *intlast l1* = *intfirst* (*map* (*shift cp*) *l2*)
using *assms*
  by (*metis Interval.shift-def add.left-neutral index-sequence-def*
        interval-nth-intlen-intlast interval-nth-map interval-nth-zero-intfirst)

lemma *interval-idx-fuse-nth-cp*:
  assumes *index-sequence 0 l1*
        index-sequence 0 l2
        nth l1 (*intlen l1*) = *cp*
        nth l2 (*intlen l2*) = *intlen* σ −*cp*
        cp $\leq$ *intlen* σ
        l = *fuse l1* (*map* (*shift cp*) *l2*)
        i $\leq$ *intlen l2*
  shows *nth l* (*intlen l1* + *i*) = *cp* + *nth l2 i*
proof −
  have *1*: *intlast l1* = *intfirst* (*map* (*shift cp*) *l2*)
    using *assms interval-idx-fuse-intfirst-intlast* **by** *blast*
  have *2*: *nth l* (*intlen l1* + *i*) = *nth* (*map* (*shift cp*) *l2*) *i*
    using *assms* **by** (*metis 1 interval-fuse-nth-a interval-intlen-map*)
  have *3*: *nth* (*map* (*shift cp*) *l2*) *i* = *nth l2 i* +*cp*
    by (*simp add*: *Interval.shift-def interval-nth-map*)
   show *?thesis* **using** *2 3* **by** *auto*
qed

lemma *interval-idx-fuse-idx*:
  assumes *index-sequence 0 l1*
        index-sequence 0 l2
        nth l1 (*intlen l1*) = *cp*
        nth l2 (*intlen l2*) = *intlen* σ −*cp*
        cp $\leq$ *intlen* σ
        l = *fuse l1* (*map* (*shift cp*) *l2*)
        i $\leq$ *intlen l2*
  shows *index-sequence 0 l*
proof −
  have *1*: *intlast l1* = *intfirst* (*map* (*shift cp*) *l2*)
    using *assms interval-idx-fuse-intfirst-intlast* **by** *blast*
  have *2*: *nth* (*fuse l1* (*map* (*shift cp*) *l2*)) *0* = *nth l1 0*

    using *1 interval-fuse-nth* **by** *blast*
  have *3*: *intlen* (*fuse l1* (*map* (*shift cp*) *l2*)) = *intlen l1* + *intlen l2*
    by (*simp add*: *interval-fuse-intlen-a*)
  have *4*: ∀ *i*. *0*≤*i* ∧ *i*≤*intlen l1* ⟶
          nth (*fuse l1* (*map* (*shift cp*) *l2*)) *i* =
          nth l1 i
    by (*metis 1 interval-nth-prefix interval-prefix-fuse*)
  have *5*: ∀ *i*. *intlen l1* ≤*i* ∧ *i*≤*intlen l1* + *intlen l2* ⟶
          nth (*fuse l1* (*map* (*shift cp*) *l2*)) *i* =
          cp + *nth l2* (*i* −*intlen l1*)
    by (*metis* (*no-types*, *lifting*) *1 3 Interval*.*shift-def add*.*commute interval-fuse-nth*
      interval-nth-map)
  have *6*: ∀ *i*. *0*≤*i* ∧ *i*<*intlen l1*+*intlen l2* ⟶
           nth (*fuse l1* (*map* (*shift cp*) *l2*)) *i* < *nth* (*fuse l1* (*map* (*shift cp*) *l2*)) (*Suc i*)
    using *assms* **by** (*metis 1 3 4 add*.*right-neutral index-sequence-def interval-idx-link*
      interval-idx-split interval-intlen-gr-zero interval-prefix-fuse interval-suffix-fuse
      le-add1)
  have *7*: *index-sequence 0 l* =
         ((*nth l1 0*) = *0* ∧
         (∀ *i*. *0*≤*i* ∧ *i*<*intlen l1*+*intlen l2* ⟶
           nth (*fuse l1* (*map* (*shift cp*) *l2*)) *i* < *nth* (*fuse l1* (*map* (*shift cp*) *l2*)) (*Suc i*)))
    by (*simp add*: *2 3 assms index-sequence-def*)
  from *7 6 2* **show** *?thesis*
  using *assms index-sequence-def* **by** *auto*
qed

lemma *interval-idx-fuse-intlen*:
  assumes *index-sequence 0 l1*
        index-sequence 0 l2
        nth l1 (*intlen l1*) = *cp*
        nth l2 (*intlen l2*) = *intlen σ* −*cp*
        cp ≤ *intlen σ*
        l = *fuse l1* (*map* (*shift cp*) *l2*)
        i ≤ *intlen l2*
  shows   *nth l* (*intlen l*) = *intlen σ*
using *assms interval-idx-fuse-nth-cp*[*of l1 l2 cp σ*] **by** (*simp add*: *interval-fuse-intlen-a*)

lemma *interval-intfirst-lfuse-intfirst*:
assumes *lastfirst* (*xs* ⊙ *xxs*)
shows *intfirst*(*lfuse xxs*) = *intfirst*(*intfirst xxs*)
using *assms*
proof
  (*induct xxs*)
  case (*St x*)
  then show *?case* **by** *simp*
  next
  case (*Cons x1a xxs*)
  then show *?case*
      proof (*cases x1a*)

case (*St x1*)
then show *?thesis* **using** *Cons.hyps Cons.prems* **by** *auto*
next
case (*Cons x21 x22*)
then show *?thesis* **by** *simp*
qed
qed

lemma *interval-intfirst-lfuse*:
assumes *lastfirst* (*xs* $\odot$ *xxs*)
shows (*intfirst* (*lfuse* (*xs* $\odot$ *xxs*))) = (*intfirst xs*)
proof −
have *1*: *lastfirst* (*xs* $\odot$ *xxs*)
using *assms* **by** *auto*
have *2*: (((*intlast xs*) = (*intfirst* (*intfirst xxs*))) $\wedge$ (*lastfirst xxs*))
using *1* **by** *simp*
have *3*: (*intlast xs*) = (*intfirst* (*intfirst xxs*))
using *2* **by** *auto*
have *4*: *intfirst* (*lfuse* (*xs* $\odot$ *xxs*)) = *intfirst*(*fuse xs* (*lfuse xxs*))
by *simp*
have *5*: *intfirst*(*lfuse xxs*) = *intfirst*(*intfirst xxs*)
using *assms interval-intfirst-lfuse-intfirst* **by** *blast*
have *6*: *intfirst*(*fuse xs* (*lfuse xxs*)) = *intfirst xs*
by (*metis 3 5 interval-intfirst-fuse*)
show *?thesis* **using** *6* **by** *auto*
qed

lemma *interval-lastfirst-lfuse-intlast*:
assumes *lastfirst xxs*
shows *intlast*(*lfuse xxs*) = *intlast*(*intlast xxs*)
using *assms*
proof
(*induct xxs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xxs*)
then show *?case*
by (*metis interval-fuse-intlen-a interval-fuse-nth-a interval-intfirst-lfuse-intfirst interval-nth-intlen-intlast interval-nth-last lastfirst.simps*(*2*) *lfuse-Cons order.order-iff-strict*)
qed

lemma *interval-lastfirst-lfuse*:
assumes *lastfirst xxs*
shows *intfirst* (*lfuse xxs*) = *intfirst*(*intfirst*(*xxs*))
using *assms*
proof
(*cases xxs*)
case (*St x1*)

then show *?thesis* **by** *simp*
next
case (*Cons x21 x22*)
then show *?thesis*
using *assms interval-intfirst-lfuse* **by** *auto*
qed

lemma *interval-lfuse-intlen* :
assumes *lastfirst xxs*
shows *intlen* (*lfuse xxs*) = ($\sum$ *k*::*nat*= *0*..(*intlen xxs*). *intlen*(*nth xxs k*))
using *assms*
proof
(*induct xxs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xxs*)
then show *?case* **proof** −
have *1* : *intlen* (*lfuse* (*x1a* ⊙ *xxs*)) = *intlen* (*fuse x1a* (*lfuse xxs*))
by *simp*
have *2* : *lastfirst* (*x1a* ⊙ *xxs*) **using** *Cons*.*prems* **by** *auto*
have *3* : *intlast x1a* = *intfirst*(*lfuse xxs*)
using *Cons*.*prems interval-intfirst-lfuse-intfirst* **by** *fastforce*
have *4* : *intlen* (*fuse x1a* (*lfuse xxs*)) = (*intlen x1a*) + *intlen*(*lfuse xxs*)
using *3 interval-fuse-intlen* **by** *blast*
have *5*: (*intlen x1a*) + *intlen*(*lfuse xxs*) =
(*intlen x1a*) + ($\sum$ *k*::*nat*= *0*..(*intlen xxs*). *intlen*(*nth xxs k*))
using *Cons*.*hyps Cons*.*prems* **by** *auto*
have *6*: ($\sum$ *k* = *0*..*intlen* (*x1a* ⊙ *xxs*). *intlen* (*Interval*.*nth* (*x1a* ⊙ *xxs*) *k*)) =
(*intlen*(*nth* (*x1a* ⊙ *xxs*) *0*)) +
($\sum$ *k* = *1*..*1*+*intlen* (*xxs*). *intlen* (*Interval*.*nth* (*x1a* ⊙ *xxs*) *k*))
by (*simp add*: *sum*.*atLeast-Suc-atMost*)
have *7*: (*intlen*(*nth* (*x1a* ⊙ *xxs*) *0*)) = *intlen*(*x1a*)
by *simp*
have *8*: ($\sum$ *k* = *1*..*1*+*intlen* (*xxs*). *intlen* (*Interval*.*nth* (*x1a* ⊙ *xxs*) *k*)) =
($\sum$ *k* = *0*..*intlen* (*xxs*). *intlen* (*Interval*.*nth* (*x1a* ⊙ *xxs*) (*k*+*1*)))
by (*metis* (*mono-tags*, *lifting*) *Nat*.*add-0-right add*.*commute sum*.*cong sum*.*shift-bounds-cl-nat-ivl*)
have *9*: ($\sum$ *k* = *0*..*intlen* (*xxs*). *intlen* (*Interval*.*nth* (*x1a* ⊙ *xxs*) (*k*+*1*))) =
($\sum$ *k* = *0*..*intlen* (*xxs*). *intlen* (*Interval*.*nth* (*xxs*) (*k*)))
by *auto*
have *10*: (*intlen x1a*) + ($\sum$ *k*::*nat*= *0*..(*intlen xxs*). *intlen*(*nth xxs k*)) =
($\sum$ *k* = *0*..*intlen* (*x1a* ⊙ *xxs*). *intlen* (*Interval*.*nth* (*x1a* ⊙ *xxs*) *k*))
using *6 7 8 9* **by** *linarith*
show *?thesis*
by (*simp add*: *10 4 5*)
qed
qed

lemma *interval-idx-fuse*:
assumes *intlast l1* = *intfirst l2*

shows
(*index-sequence* (*intfirst l1*) (*fuse l1 l2*)) =
(*index-sequence* (*intfirst l1*) *l1* ∧ *index-sequence* (*intfirst l2*) *l2*)
using *assms*
proof
(*induct l1 arbitrary*: *l2*)
case (*St x*)
then show *?case* **by** (*simp add*: *index-sequence-def*)
next
case (*Cons x1a l1*)
then show *?case*
using *interval-idx-expand1 interval-intfirst-fuse* **by** *force*
qed

lemma *interval-idx-lfuse-help1*:
assumes (∀ *k*. *k*<*intlen* (*lfuse l*) ⟶
nth (*fuse x1a* (*lfuse l*)) (*intlen x1a* + *k*) <
nth (*fuse x1a* (*lfuse l*)) (*intlen x1a* + *Suc k*))

intlen x1a ≤ *n*
n< *intlen x1a* +*intlen* (*lfuse l*)
shows *nth* (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*)
using *assms*
by (*metis add-Suc-right add-less-imp-less-left le-Suc-ex*)

lemma *interval-idx-lfuse*:
assumes *lastfirst l*
shows (*index-sequence* (*intfirst* (*lfuse l*)) (*lfuse l*)) =
(∀ *i*≤*intlen l*. *index-sequence* (*intfirst* (*nth l i*)) (*nth l i*))
using *assms*
proof
(*induct l*)
case (*St x*)
then show *?case* **by** (*simp add*: *index-sequence-def*)
next
case (*Cons x1a l*)
then show *?case*
proof −
have *0*: *lastfirst l*
using *Cons.prems lastfirst.simps*(*2*) **by** *blast*
have *1*: *index-sequence* (*intfirst* (*lfuse* (*x1a* ⊙ *l*))) (*lfuse* (*x1a* ⊙ *l*)) =
(*nth* (*fuse x1a* (*lfuse l*)) *0* = *intfirst* (*fuse x1a* (*lfuse l*)) ∧
(∀ *n*<*intlen* (*fuse x1a* (*lfuse l*)).
nth (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*)))
by (*simp add*: *index-sequence-def*)
have *2*: (*nth* (*fuse x1a* (*lfuse l*)) *0*) = *intfirst* (*fuse x1a* (*lfuse l*))
by *simp*
have *3*: *intlen* (*fuse x1a* (*lfuse l*)) = *intlen x1a* + *intlen*(*lfuse l*)
by (*simp add*: *interval-fuse-intlen-a*)
have *4*: (∀ *n*<*intlen* (*fuse x1a* (*lfuse l*)).

nth (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*)) =
(∀ *n*<*intlen x1a* + *intlen* (*lfuse l*).
nth (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*))
by (*simp add*: *3*)
have *5*: (∀ *n*<*intlen x1a* + *intlen* (*lfuse l*).
nth (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*)) =
((∀ *n*<*intlen x1a*. *nth* (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*)) ∧
(∀ *n*. *0* ≤ *n*−*intlen x1a* ∧ *n* − *intlen x1a*<*intlen* (*lfuse l*) ⟶
nth (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*)))
by *auto*
(*metis add*.*commute less-diff-conv2 not-less*)
have *6*: (∀ *n*<*intlen x1a*. *nth* (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*)) =
index-sequence (*intfirst x1a*) *x1a*
by (*simp add*: *index-sequence-def*)
(*metis Cons*.*prems Suc-leI interval-intfirst-lfuse-intfirst interval-nth-prefix*
interval-prefix-fuse lastfirst.*simps*(*2*) *le-simps*(*1*))
have *7*: (∀ *n*. *intlen x1a* ≤ *n* ∧ *n* < *intlen x1a* + *intlen* (*lfuse l*) ⟶
nth (*fuse x1a* (*lfuse l*)) *n* < *nth* (*fuse x1a* (*lfuse l*)) (*Suc n*)) =
(∀ *k*. *k*<*intlen* (*lfuse l*) ⟶
nth (*fuse x1a* (*lfuse l*)) (*intlen x1a* + *k*) <
nth (*fuse x1a* (*lfuse l*)) (*intlen x1a* + *Suc k*))
using *interval-idx-lfuse-help1* **by** *auto*
have *8*: (∀ *k*. *k*<*intlen* (*lfuse l*) ⟶
nth (*fuse x1a* (*lfuse l*)) (*intlen x1a* + *k*) <
nth (*fuse x1a* (*lfuse l*)) (*intlen x1a* + *Suc k*)) =
(∀ *k*. *k*<*intlen* (*lfuse l*) ⟶
nth ((*lfuse l*)) (*k*) <
nth ((*lfuse l*)) (*Suc k*)) (**is** *?L*=*?R*)
proof
show *?L* ⟹ *?R*
by (*metis Cons*.*prems Suc-leI add-Suc-right interval-fuse-nth-a*
interval-intfirst-lfuse-intfirst lastfirst.*simps*(*2*) *le-simps*(*1*))
show *?R* ⟹ *?L*
by (*metis Cons*.*prems Suc-leI add-Suc-right interval-fuse-nth-a*
interval-intfirst-lfuse-intfirst lastfirst.*simps*(*2*) *less-imp-le-nat*)
qed
have *9*: (∀ *k*. *k*<*intlen* (*lfuse l*) ⟶
nth ((*lfuse l*)) (*k*) <
nth ((*lfuse l*)) (*Suc k*)) = *index-sequence* (*intfirst* (*lfuse l*)) (*lfuse l*)
by (*simp add*: *index-sequence-def*)
have *91*: *index-sequence* (*intfirst* (*lfuse* (*x1a* ⊙ *l*))) (*lfuse* (*x1a* ⊙ *l*)) =
(*index-sequence* (*intfirst x1a*) *x1a* ∧ *index-sequence* (*intfirst* (*lfuse l*)) (*lfuse l*))
using *1 2 4 5 6 7 8 9*
by (*metis Cons*.*prems interval-idx-fuse interval-intfirst-lfuse*
interval-intfirst-lfuse-intfirst lastfirst.*simps*(*2*) *lfuse-Cons*)
have *10*: *index-sequence* (*intfirst* (*lfuse l*)) (*lfuse l*) =
(∀ *i*≤*intlen l*. *index-sequence* (*intfirst* (*nth l i*)) (*nth l i*))
using *0 Cons*.*hyps* **by** *blast*
have *11*: (∀ *i*≤ *intlen*(*x1a*⊙*l*). *index-sequence* (*intfirst* (*nth* (*x1a*⊙*l*) *i*)) (*nth* (*x1a*⊙*l*) *i*)) =
(∀ *i*≤ *1* +*intlen l*. *index-sequence* (*intfirst* (*nth* (*x1a*⊙*l*) *i*)) (*nth* (*x1a*⊙*l*) *i*))

by *auto*
have *12*: ($\forall$ *i*$\leq$ *1* +*intlen l*. *index-sequence* (*intfirst* (*nth* (*x1a*$\odot$*l*) *i*)) (*nth* (*x1a*$\odot$*l*) *i*)) =
(*index-sequence* (*intfirst* (*x1a*)) (*x1a*) $\wedge$
($\forall$ *i*. *1* $\leq$ *i* $\wedge$ *i*$\leq$ *1* +*intlen l* $\longrightarrow$
index-sequence (*intfirst* (*nth* (*x1a* $\odot$ *l*) *i*)) (*nth* (*x1a* $\odot$ *l*) *i*)))
by (*metis One-nat-def Suc-leI interval-intlen-gr-zero interval-nth-zero*
interval-prefix-length-good intlen.simps(*2*) *order.strict-iff-order*)
have *13*: ($\forall$ *i*. *1* $\leq$ *i* $\wedge$ *i*$\leq$ *1* +*intlen l* $\longrightarrow$
index-sequence (*intfirst* (*nth* (*x1a* $\odot$ *l*) *i*)) (*nth* (*x1a* $\odot$ *l*) *i*)) =
($\forall$ *j*. *j*$\leq$ *intlen l* $\longrightarrow$
index-sequence (*intfirst* (*nth* (*x1a* $\odot$ *l*) (*1*+*j*))) (*nth* (*x1a* $\odot$ *l*) (*1*+*j*)))
by *auto*
(*simp add*: *Nitpick.case-nat-unfold*)
have *14*: ($\forall$ *j*. *j*$\leq$ *intlen l* $\longrightarrow$
index-sequence (*intfirst* (*nth* (*x1a* $\odot$ *l*) (*1*+*j*))) (*nth* (*x1a* $\odot$ *l*) (*1*+*j*))) =
($\forall$ *j*. *j*$\leq$ *intlen l* $\longrightarrow$
index-sequence (*intfirst* (*nth* (*l*) (*j*))) (*nth* (*l*) (*j*)))
by *simp*
show *?thesis*
using *10 12 13 91* **by** *auto*
qed
qed

lemma *interval-lfuse-intlen-a*:
assumes *lastfirst xxs*
shows ($\forall$ *i*. *i* $\leq$*intlen* (*xxs*) $\longrightarrow$
($\forall$ *j*$\leq$*intlen* (*nth* (*xxs*) (*i*)) . *j*$\leq$ *intlen*(*lfuse xxs*)))
using *assms*
proof (*induct xxs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xxs*)
then show *?case*
proof −
have *0*: *intlast x1a* = *intfirst*(*intfirst xxs*)
using *Cons.prems lastfirst.simps*(*2*) **by** *blast*
have *1*: ($\forall$ *i*. *i* $\leq$ *intlen* (*x1a* $\odot$ *xxs*) $\longrightarrow$
($\forall$ *j*$\leq$*intlen* (*nth* (*x1a* $\odot$ *xxs*) *i*). *j* $\leq$ *intlen* (*lfuse* (*x1a* $\odot$ *xxs*))))
=
(($\forall$ *j*$\leq$*intlen* (*Interval.nth* (*x1a* $\odot$ *xxs*) *0*). *j* $\leq$ *intlen* (*lfuse* (*x1a* $\odot$ *xxs*))) $\wedge$
($\forall$ *i*. *1* $\leq$ *i* $\wedge$ *i*−*1* $\leq$ *intlen* (*xxs*) $\longrightarrow$
($\forall$ *j*$\leq$*intlen* (*nth* (*x1a* $\odot$ *xxs*) *i*). *j* $\leq$ *intlen* (*lfuse* (*x1a* $\odot$ *xxs*)))))
by *auto*
(*metis One-nat-def add.commute le-diff-conv le-zero-eq not-less-eq-eq old.nat.simps*(*4*)
plus-1-eq-Suc)
have *2*: ($\forall$ *j*$\leq$*intlen* (*Interval.nth* (*x1a* $\odot$ *xxs*) *0*). *j* $\leq$ *intlen* (*lfuse* (*x1a* $\odot$ *xxs*))) =
($\forall$ *j*$\leq$*intlen* (*x1a*). *j* $\leq$ *intlen* (*fuse x1a* (*lfuse xxs*)))

by *simp*
have *3*: $(\forall j \le intlen\ (x1a).\ j \le intlen\ (fuse\ x1a\ (lfuse\ xxs))) =$
$(\forall j \le intlen\ (x1a).\ j \le intlen\ (x1a) + intlen\ (lfuse\ xxs))$
by (*simp add*: *interval-fuse-intlen-a*)
have *4*: $(\forall j \le intlen\ (x1a).\ j \le intlen\ (x1a) + intlen\ (lfuse\ xxs))$
by *linarith*
have *5*: $(\forall i.\ 1 \le i \wedge i{-}1 \le intlen\ (xxs) \longrightarrow$
$(\forall j \le intlen\ (nth\ (x1a \odot xxs)\ i).\ j \le intlen\ (lfuse\ (x1a \odot xxs))))$
$=$
$(\forall i.\ 0 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall j \le intlen\ (nth\ (x1a \odot xxs)\ (Suc\ i)).\ j \le intlen\ (lfuse\ (x1a \odot xxs))))$
by (*metis add-diff-cancel-left′ interval-intlen-gr-zero interval-prefix-length-good le-add1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
have *6*: $(\forall i.\ 0 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall j \le intlen\ (nth\ (x1a \odot xxs)\ (Suc\ i)).\ j \le intlen\ (lfuse\ (x1a \odot xxs)))) =$
$(\forall i.\ 0 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall j \le intlen\ (nth\ (\ xxs)\ (i)).\ j \le intlen\ x1a + intlen\ (lfuse\ (\ xxs))))$
by (*simp add*: *interval-fuse-intlen-a*)
have *7*: $\forall i.\ 0 \le i \wedge i \le intlen\ xxs \longrightarrow (\forall j \le intlen\ (Interval.nth\ xxs\ i).\ j \le intlen\ (lfuse\ xxs))$
using *Cons.hyps Cons.prems lastfirst.simps*(*2*) **by** *blast*
have *8*: $(\forall i.\ 0 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall j \le intlen\ (nth\ (\ xxs)\ (i)).\ j \le intlen\ x1a + intlen\ (lfuse\ (\ xxs))))$
by (*simp add*: *7 trans-le-add2*)
show *?thesis* **using** *1 3 5 6 8* **by** *auto*
qed
qed

lemma *interval-lfuse-split*:
assumes *lastfirst xxs* $\wedge$ $(\forall\ j \le intlen\ (xxs).\ intlen(nth\ (xxs)\ j) > 0)$
shows $(\forall i \le\ intlen(xxs).$
$(\forall ia < intlen\ ((nth\ (xxs)\ i)).$
$f\ \ (sub\ (nth\ (nth\ (xxs)\ i)\ ia)$
$(nth\ (nth\ (xxs)\ i)\ (Suc\ ia))$
$\sigma))) =$
$(\forall j < intlen\ (lfuse\ (xxs)).$
$f\ \ (sub\ (nth\ (lfuse\ (xxs))\ j)$
$(nth\ (lfuse\ (xxs))\ (Suc\ j))$
$\sigma))$
using *assms*
proof (*induct xxs*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a xxs*)
then show *?case*
proof −
have *1*: $(\forall i \le intlen\ (x1a \odot xxs).$
$\forall ia < intlen\ (nth\ (x1a \odot xxs)\ i).$
$f\ \ (sub\ (nth\ (nth\ (x1a \odot xxs)\ i)\ ia)\ (nth\ (nth\ (x1a \odot xxs)\ i)\ (Suc\ ia))\ \sigma)) =$

$(\forall\, i \le intlen\ (xxs)+1.$
$\quad \forall\, ia < intlen\ (nth\ (x1a \odot xxs)\ i).$
$\quad\ f\ (sub\ (nth\ (nth\ (x1a \odot xxs)\ i)\ ia)\ (nth\ (nth\ (x1a \odot xxs)\ i)\ (Suc\ ia))\ \sigma))$
by *simp*
have *2*: ... =
$(\ (\forall\, ia < intlen\ (nth\ (x1a \odot xxs)\ 0).$
$\quad f\ (sub\ (nth\ (nth\ (x1a \odot xxs)\ 0)\ ia)\ (nth\ (nth\ (x1a \odot xxs)\ 0)\ (Suc\ ia))\ \sigma))$
$\quad \wedge$
$(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs)+1 \longrightarrow$
$\ (\forall\, ia < intlen\ (nth\ (x1a \odot xxs)\ i).$
$\ \ f\ (sub\ (nth\ (nth\ (x1a \odot xxs)\ i)\ ia)\ (nth\ (Interval.nth\ (x1a \odot xxs)\ i)\ (Suc\ ia))\ \sigma))))$
by (*metis One-nat-def Suc-leI add-nonneg-nonneg gr-zeroI interval-intlen-gr-zero zero-le-one*)
have *3*: $(\forall\, ia < intlen\ (nth\ (x1a \odot xxs)\ 0).$
$\quad f\ (sub\ (nth\ (nth\ (x1a \odot xxs)\ 0)\ ia)\ (nth\ (nth\ (x1a \odot xxs)\ 0)\ (Suc\ ia))\ \sigma)) =$
$(\forall\, ia < intlen\ x1a.$
$\quad f\ (sub\ (nth\ (x1a)\ ia)\ (nth\ (x1a)\ (Suc\ ia))\ \sigma))$
by *simp*
have *4*: $(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs)+1 \longrightarrow$
$\quad (\forall\, ia < intlen\ (nth\ (x1a \odot xxs)\ i).$
$\quad f\ (sub\ (nth\ (nth\ (x1a \odot xxs)\ i)\ ia)\ (nth\ (nth\ (x1a \odot xxs)\ i)\ (Suc\ ia))\ \sigma))) =$
$(\forall\, i.\ 0 \le i-1 \wedge i-1 \le intlen\ (xxs) \longrightarrow$
$\quad (\forall\, ia < intlen\ (nth\ (x1a \odot xxs)\ ((i-1)+1)).$
$\quad f\ (sub\ (nth\ (nth\ (x1a \odot xxs)\ ((i-1)+1))\ ia)$
$\qquad (nth\ (nth\ (x1a \odot xxs)\ ((i-1)+1))\ (Suc\ ia))\ \sigma)))$
by (*auto simp add*: *Nitpick.case-nat-unfold*)
have *5*: ... =
$(\forall\, i.\ 0 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$\quad (\forall\, ia < intlen\ (nth\ (x1a \odot xxs)\ ((i)+1)).$
$\quad f\ (sub\ (nth\ (nth\ (x1a \odot xxs)\ ((i)+1))\ ia)$
$\qquad (nth\ (nth\ (x1a \odot xxs)\ ((i)+1))\ (Suc\ ia))\ \sigma)))$
using *4* **by** *auto*
have *6*: ... =
$(\forall\, i.\ 0 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$\quad (\forall\, ia < intlen\ (nth\ (xxs)\ ((i))).$
$\quad f\ (sub\ (nth\ (nth\ (xxs)\ ((i)))\ ia)\ (nth\ (nth\ (xxs)\ ((i)))\ (Suc\ ia))\ \sigma)))$

by *simp*
have *7*: *lastfirst xxs*
using *Cons.prems lastfirst.simps*(*2*) **by** *blast*
have *8*: *intlast x1a* = *intfirst*(*intfirst xxs*)
using *Cons.prems lastfirst.simps*(*2*) **by** *blast*
have *9*: $(\forall\ j \le intlen\ (x1a \odot xxs).\ intlen(nth\ (x1a \odot xxs)\ j) > 0)$
using *Cons.prems* **by** *blast*
have *10*: $intlen(nth\ (x1a \odot xxs)\ 0) > 0$
using *Cons.prems* **by** *blast*
have *11*: $(\forall j.\ 1 \le j \wedge j \le intlen\ (xxs)+1 \longrightarrow intlen(nth\ (x1a \odot xxs)\ j) > 0)$
using *Cons.prems* **by** *auto*
have *12*: $(\forall j.\ 0 \le j-1 \wedge j-1 \le intlen\ (xxs) \longrightarrow intlen(nth\ (x1a \odot xxs)\ ((j-1)+1)) > 0)$
using *Cons.prems* **by** *auto*
have *13*: $(\forall j.\ j \le intlen\ (xxs) \longrightarrow intlen(nth\ (x1a \odot xxs)\ ((j)+1)) > 0)$

```
  using Cons.prems by auto
have 14: (∀j.  j ≤ intlen (xxs) ⟶ intlen(nth (xxs) ((j))) > 0)
  using 13 by simp
have 15: (∀i.  i≤intlen (xxs)⟶
          (∀ia<intlen (nth (xxs) ((i))).
           f (sub (nth (nth (xxs) ((i))) ia) (nth (nth (xxs) ((i))) (Suc ia)) σ))) =
         (∀j<intlen (lfuse (xxs)).
             f (sub (nth (lfuse (xxs)) j)
                    (nth (lfuse (xxs)) (Suc j))
                    σ))
    by (simp add: 14 7 Cons.hyps)
have 16: (∀j<intlen (lfuse (x1a ⊙ xxs)).
           f (sub (nth (lfuse (x1a ⊙ xxs)) j) (nth (lfuse (x1a ⊙ xxs)) (Suc j)) σ)) =
         (∀j<intlen (fuse x1a (lfuse (xxs))).
           f (sub (nth (lfuse (x1a ⊙ xxs)) j) (nth (lfuse (x1a ⊙ xxs)) (Suc j)) σ))
          by simp
have 17:  ... =
          (∀j<intlen x1a + intlen ((lfuse (xxs))).
           f (sub (nth (lfuse (x1a ⊙ xxs)) j) (nth (lfuse (x1a ⊙ xxs)) (Suc j)) σ))
  by (simp add: interval-fuse-intlen-a)
have 18:  ... =
          ((∀j<intlen x1a.
           f (sub (nth (lfuse (x1a ⊙ xxs)) j) (nth (lfuse (x1a ⊙ xxs)) (Suc j)) σ)) ∧
          (∀j. intlen x1a ≤ j ∧ j<intlen x1a + intlen ((lfuse (xxs))) ⟶
           f (sub (nth (lfuse (x1a ⊙ xxs)) j) (nth (lfuse (x1a ⊙ xxs)) (Suc j)) σ)))
   using le-add1 less-le-trans not-less by blast
have 19: (∀j<intlen x1a.
           f (sub (nth (lfuse (x1a ⊙ xxs)) j) (nth (lfuse (x1a ⊙ xxs)) (Suc j)) σ)) =
         (∀j<intlen x1a.
           f (sub (nth (fuse x1a (lfuse (xxs))) j) (nth (fuse x1a (lfuse (xxs))) (Suc j)) σ))
       by simp
have 20:  ... =
          (∀j<intlen x1a.
           f (sub (nth (x1a) j) (nth (x1a) (Suc j)) σ))
     by (metis Cons.prems Suc-leI  interval-intfirst-lfuse-intfirst
        interval-nth-prefix interval-prefix-fuse lastfirst.simps(2) less-imp-le-nat)
have 21: (∀j. intlen x1a ≤ j ∧ j<intlen x1a + intlen ((lfuse (xxs))) ⟶
           f (sub (nth (lfuse (x1a ⊙ xxs)) j) (nth (lfuse (x1a ⊙ xxs)) (Suc j)) σ)) =
         (∀j. intlen x1a ≤ j ∧ j<intlen x1a + intlen ((lfuse (xxs))) ⟶
           f (sub (nth (fuse x1a (lfuse (xxs))) j) (nth (fuse x1a (lfuse (xxs))) (Suc j)) σ))
   by simp
have 22:  ... =
          (∀j. intlen x1a ≤ j ∧ j<intlen x1a + intlen ((lfuse (xxs))) ⟶
           f (sub (nth ((lfuse (xxs))) (j −intlen x1a))
                  (nth (fuse x1a (lfuse (xxs))) ((j)+1)) σ))  (is ?L=?R)
        proof rule
         show ?L⟹?R
           by auto
          (metis 8 Cons.prems interval-fuse-intlen-a interval-fuse-nth
           interval-intfirst-lfuse-intfirst less-imp-le-nat)
```

```
        show ?R ⟹ ?L
         by auto
        (metis 8 Cons.prems interval-fuse-intlen-a interval-fuse-nth
         interval-intfirst-lfuse-intfirst less-imp-le-nat)
       qed
  have 23: ... =
        (∀ j. intlen x1a ≤ j ∧ j<intlen x1a + intlen ((lfuse (xxs))) ⟶
         f (sub (nth ((lfuse (xxs))) (j −intlen x1a))
               (nth ( (lfuse (xxs))) (((Suc j)−intlen x1a))) σ)) (is ?L=?R)
       proof
        show ?L⟹?R
         by auto
           (metis 8 Cons.prems Suc-leI interval-fuse-intlen-a interval-fuse-nth
            interval-intfirst-lfuse-intfirst le-SucI)
        show ?R ⟹?L
         by auto
           (metis 8 Cons.prems Suc-leI interval-fuse-intlen-a interval-fuse-nth
            interval-intfirst-lfuse-intfirst le-SucI)
       qed
  have 24: ... =
        (∀ j. 0 ≤ j  ∧ j < intlen ((lfuse (xxs))) ⟶
         f (sub (nth ((lfuse (xxs))) (j) ) (nth ( (lfuse (xxs))) (((Suc j) ))) σ))
       by (rule interval-shift-index-to-zero-b)
  show ?thesis
  using 15 17 18 2 20 22 23 24 4 5 by auto
 qed
qed


end
```

4 Infinite Intervals

```
theory InfiniteInterval
 imports
   Interval
begin
```

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

4.1 Definitions

```
type-synonym
 'a infinterval = nat ⇒ 'a
```

type-synonym
infiniteindex = *nat infinterval*

definition
conc :: $['a\ interval, 'a\ infinterval] \Rightarrow 'a\ infinterval$
where *conc w x* = $(\lambda n.\ if\ n \leq intlen\ w\ then\ nth\ w\ n\ else\ x\ (n\ -\ intlen\ w\ -1))$

definition
isuffix :: $[nat, 'a\ infinterval] \Rightarrow 'a\ infinterval$
where *isuffix k x* = $(\lambda n.\ x\ (k+n))$

definition
subinterval :: $'a\ infinterval \Rightarrow nat \Rightarrow nat \Rightarrow 'a\ interval$
where
subinterval w i j = *map w* $[i..\leq j]$

definition
iprefix :: $nat \Rightarrow 'a\ infinterval \Rightarrow 'a\ interval$
where
iprefix n w $\equiv$ *subinterval w 0 n*

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

4.2 Lemmas

4.2.1 isuffix

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

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

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

4.2.2 iprefix

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

lemma *iprefix-nth*:
assumes $k \leq m$
shows $(nth\ (iprefix\ m\ x)\ k) = (x\ k)$
using *assms*

by (*simp add*: *interval-nth-map iprefix-def subinterval-def upt-nth*)

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

4.2.3 subinterval

lemma *subinterval-length*:
intlen (*subinterval x i j*) = *j* −*i*
by (*simp add*: *subinterval-def upt-length*)

lemma *subinterval-nth*:
assumes *i*+*k* ≤ *j*
shows *nth* (*subinterval x i j*) *k* = *x* (*i*+*k*)
unfolding *subinterval-def*
using *assms* **by** (*simp add*: *interval-nth-map upt-nth*)

lemma *iprefix-isuffix*:
iprefix n (*isuffix k x*) = *subinterval x k* (*n*+*k*)
proof −
have *0*: *iprefix n* (*isuffix k x*) = *map* (λ*n*. *x* (*k* + *n*)) [*0*..≤*n*]
by (*simp add*: *iprefix-def isuffix-def subinterval-def*)
have *1*: [*k*..≤*n*+*k*] = (*map* (λ*i*. *i*+*k*) [*0*..≤*n*])
using *map-add-upt* **by** *simp*
hence *2*: *map x* [*k*..≤*n*+*k*] = *map x* (*map* (λ*i*. *i*+*k*) [*0*..≤*n*])
by *simp*
have *3*: *map x* (*map* (λ*i*. *i*+*k*) [*0*..≤*n*]) = *map* (*x* ∘ (λ*i*. *i*+*k*)) [*0*..≤*n*]
by *simp*
have *4*: (*x* ∘ (λ*i*. *i*+*k*)) = (λ*n*. *x* (*k* + *n*)) **by** (*metis add*.*commute comp-apply*)
hence *5*: *map* (*x* ∘ (λ*i*. *i*+*k*)) [*0*..≤*n*] = *map* (λ*n*. *x* (*k* + *n*)) [*0*..≤*n*]
by *simp*
have *6*: *subinterval x k* (*n*+*k*) = *map x* [*k*..≤*n*+*k*]
by (*simp add*: *subinterval-def*)
from *0 2 3 5 6* **show** *?thesis* **by** *auto*
qed

lemma *subinterval-sub-isuffix*:
assumes *i* < *j*
shows (*subinterval xs* (*i*+*k*) (*j*+*k*)) = (*subinterval* (*isuffix k xs*) *i j*)
proof −
have *1*: (*subinterval xs* (*i*+*k*) (*j*+*k*)) =
iprefix (*j*−*i*) (*isuffix* (*i*+*k*) *xs*)
by (*simp add*: *iprefix-isuffix assms less-imp-le-nat*)
have *2*: *iprefix* (*j*−*i*) (*isuffix* (*i*+*k*) *xs*) =
iprefix (*j*−*i*) (*isuffix* (*i*) (*isuffix k xs*))
by (*simp add*: *isuffix-isuffix add*.*commute*)
have *3*: *iprefix* (*j*−*i*) (*isuffix* (*i*) (*isuffix k xs*)) =
(*subinterval* (*isuffix k xs*) *i j*)
by (*simp add*: *iprefix-isuffix assms less-imp-le-nat*)

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

lemma *subinterval-sub-isuffix-iidx*:
assumes *infinite-index-sequence 0 lsk* $\wedge$ *n>0*
shows (*subinterval* σ ((*lsk i*) +*n*) ((*lsk* (*Suc i*))+*n*)) =
(*subinterval* (*isuffix n* σ) (*lsk i*) (*lsk* (*Suc i*)))
using *assms* **by** (*simp add*: *infinite-index-sequence-def subinterval-sub-isuffix*)

lemma *interval-pref-ipref-3-intlen*:
intlen (*prefix i* (*iprefix* (*i*+*k*) *xs*)) = *intlen* (*iprefix i xs*)
by (*simp add*: *iprefix-length*)

lemma *interval-pref-ipref-3-nth*:
(*nth* (*prefix i* (*iprefix* (*i*+*k*) *xs*)) *m*) = (*nth* (*iprefix i xs*) *m*)
proof −
obtain *nn* :: *'a interval* $\Rightarrow$ *'a interval* $\Rightarrow$ *nat* **where**
$\forall$ *i ia*. (*i* $\neq$ *ia* $\vee$ *intlen i* = *intlen ia* $\wedge$
($\forall$ *n*. $\neg$ *n* $\leq$ *intlen i* $\vee$ *nth i n* = *nth ia n*)) $\wedge$
(*i* = *ia* $\vee$ *intlen i* $\neq$ *intlen ia* $\vee$ *nn ia i* $\leq$ *intlen i* $\wedge$ *nth i* (*nn ia i*) $\neq$ *nth ia* (*nn ia i*))
by (*metis* (*no-types*) *interval-eq-nth-eq*)
moreover
{ **assume** *nth* (*iprefix i xs*) (*nn* (*prefix i* (*iprefix* (*i* + *k*) *xs*)) (*iprefix i xs*)) $\neq$
nth (*prefix i* (*iprefix* (*i* + *k*) *xs*)) (*nn* (*prefix i* (*iprefix* (*i* + *k*) *xs*)) (*iprefix i xs*))
have $\neg$ *nn* (*prefix i* (*iprefix* (*i* + *k*) *xs*)) (*iprefix i xs*) $\leq$ *intlen* (*iprefix i xs*) $\vee$
nth (*iprefix i xs*) (*nn* (*prefix i* (*iprefix* (*i* + *k*) *xs*)) (*iprefix i xs*)) =
nth (*prefix i* (*iprefix* (*i* + *k*) *xs*)) (*nn* (*prefix i* (*iprefix* (*i* + *k*) *xs*)) (*iprefix i xs*))
by (*simp add*: *iprefix-length iprefix-nth*) }
ultimately show *?thesis*
by (*metis interval-pref-ipref-3-intlen*)
qed

lemma *interval-pref-ipref-3* [*simp*]:
(*prefix i* (*iprefix* (*i*+*k*) *xs*)) = *iprefix i xs*
by (*meson interval-eq-nth-eq interval-pref-ipref-3-intlen interval-pref-ipref-3-nth*)

lemma *interval-iprefix-isuffix-swap-intlen*:
intlen (*iprefix ia* (*isuffix i xs*)) = *intlen* (*suffix i* (*iprefix* (*ia*+*i*) *xs*))
by (*simp add*: *iprefix-length*)

lemma *interval-iprefix-isuffix-swap-nth*:
assumes *m*$\leq$*ia*
shows (*nth* (*iprefix ia* (*isuffix i xs*)) *m*) = (*nth* (*suffix i* (*iprefix* (*ia*+*i*) *xs*)) *m*)
using *assms* **by** (*simp add*: *iprefix-length iprefix-nth isuffix-def*)

lemma *interval-iprefix-isuffix-swap*:
iprefix ia (*isuffix i xs*) = *suffix i* (*iprefix* (*ia*+*i*) *xs*)
by (*simp add*: *interval-eq-nth-eq interval-iprefix-isuffix-swap-nth iprefix-length*)

4.2.4 Conc

lemma *conc-empty-zero*:
 (*conc* $\langle s\rangle$ *x*) *0* = *s*
unfolding *conc-def* **by** *auto*

lemma *conc-empty-suc*:
 (*conc* $\langle s\rangle$ *x*) (*Suc i*) = *x i*
unfolding *conc-def* **by** *auto*

lemma *conc-conc*:
 conc x (*conc y w*) = *conc* (*x* $\ominus$ *y*) *w* (**is** *?lhs* = *?rhs*)
proof
 fix *n*
 have *x*: *n* $\leq$ *intlen x* $\longrightarrow$ *?lhs n* = *?rhs n*
  by (*simp add*: *conc-def interval-intapp-nth*)
 have *y*: *n* > *intlen x* $\wedge$ *n* $\leq$ (*intlen x*) + (*intlen y*) $\longrightarrow$ *?lhs n* = *?rhs n*
  by (*simp add*: *conc-def interval-intapp-nth*) *arith*
 have *w*: *n* > (*intlen x*) + (*intlen y*) $\longrightarrow$ *?lhs n* = *?rhs n*
  by (*simp add*: *conc-def interval-intapp-nth*) *arith*
 from *x y w* **show** *?lhs n* = *?rhs n* **using** *not-less* **by** *blast*
qed

lemma *conc-iprefix-isuffix-help*:
 x a = *conc* (*iprefix n x*) (*isuffix* (*Suc n*) *x*) *a*
proof (*induct n*)
case *0*
then show *?case*
by (*simp add*: *conc-def iprefix-length iprefix-nth isuffix-nth*)
next
case (*Suc n*)
then show *?case*
 by (*simp add*: *conc-def iprefix-length iprefix-nth*)
 (*metis isuffix-nth le-SucI not-less-eq-eq ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
qed

lemma *conc-iprefix-isuffix*:
 x = *conc* (*iprefix n x*) (*isuffix* (*Suc n*) *x*)

proof
 (*rule ext*)
 show $\bigwedge$*xa*. *x xa* = *conc* (*iprefix n x*) (*isuffix* (*Suc n*) *x*) *xa*
 by (*simp add*: *conc-iprefix-isuffix-help*)
qed

4.2.5 Infinite index sequence

lemma *iidx-1*:
 l = (*conc* $\langle$(*l 0*)$\rangle$ (λ*x*. (*l* (*x*+*1*))))
  (**is** *?lhs* = *?rhs*)
proof

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

lemma *iidx-2*:
infinite-index-sequence 0 (*conc* $\langle(l\ 0)\rangle$ ($\lambda x.$ (*l* (x+*1*)))) =
((*l 0*) = *0* $\wedge$ (*l 0*) < (*l 1*) $\wedge$
infinite-index-sequence (*l 1*) ($\lambda x.$ (*l* (x+*1*))))
by (*auto simp add*: *infinite-index-sequence-def conc-def*)
(*metis Suc-diff-Suc add-diff-cancel-left' gr0I plus-1-eq-Suc zero-less-diff*)

lemma *iidx-less-plus*:
assumes *infinite-index-sequence n ls*
shows (*ls i*) < (*ls* (*Suc* (*i*+*k*)))
using *assms*
by (*simp add*: *infinite-index-sequence-def lift-Suc-mono-less*)

lemma *iidx-greater*:
assumes *infinite-index-sequence n ls*
shows *i*>*0* $\longrightarrow$ *n* < *ls i*
using *assms*
proof (*induct i*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc i*)
then show *?case* **by** (*metis iidx-less-plus infinite-index-sequence-def less-imp-Suc-add*)
qed

lemma *iidx-3*:
assumes *infinite-index-sequence n ls*
shows *infinite-index-sequence 0* ((*shiftm n*)$\circ$*ls*)
using *assms*
by (*simp add*: *infinite-index-sequence-def shiftm-def*)
(*metis One-nat-def add-Suc assms diff-less-mono iidx-less-plus
less-le neq0-conv plus-1-eq-Suc zero-less-diff*)

lemma *iidx-4*:
(*infinite-index-sequence* (*x*+*k*) ((*shift k*)$\circ$*ls*))=
infinite-index-sequence x ls
by (*simp add*: *shift-def infinite-index-sequence-def*)

lemma *iidx-5*:
assumes (*infinite-index-sequence k* (*lsk*) $\wedge$ *ls* = (*shiftm k*)$\circ$*lsk*)
shows *infinite-index-sequence 0 ls*

using *assms*
by (*simp add*: *infinite-index-sequence-def shiftm-def*)
  (*metis add-less-same-cancel1 diff-less-mono lift-Suc-mono-less-iff not-add-less1 not-le-imp-less*)

lemma *iidx-ext*:
  $((xs :: infiniteindex) = ys) = (\forall i.\ xs\ i = ys\ i)$
by *auto*

lemma *iidx-6*:
 $(infinite\text{-}index\text{-}sequence\ k\ lsk \wedge lsk = (shift\ k) \circ ls \wedge infinite\text{-}index\text{-}sequence\ 0\ ls) =$
  $(infinite\text{-}index\text{-}sequence\ k\ lsk \wedge ls = (shiftm\ k) \circ lsk \wedge infinite\text{-}index\text{-}sequence\ 0\ ls)$
proof (*simp add*: *iidx-ext infinite-index-sequence-def shift-def shiftm-def*)
  show $(lsk\ 0 = k \wedge (\forall n.\ lsk\ n < lsk\ (Suc\ n)) \wedge$
      $(\forall i.\ lsk\ i = ls\ i + k) \wedge ls\ 0 = 0 \wedge (\forall n.\ ls\ n < ls\ (Suc\ n))) =$
    $(lsk\ 0 = k \wedge (\forall n.\ lsk\ n < lsk\ (Suc\ n)) \wedge$
      $(\forall i.\ ls\ i = lsk\ i - k) \wedge ls\ 0 = 0 \wedge (\forall n.\ ls\ n < ls\ (Suc\ n)))$
   proof
    show $lsk\ 0 = k \wedge (\forall n.\ lsk\ n < lsk\ (Suc\ n)) \wedge$
      $(\forall i.\ lsk\ i = ls\ i + k) \wedge ls\ 0 = 0 \wedge (\forall n.\ ls\ n < ls\ (Suc\ n)) \Longrightarrow$
     $lsk\ 0 = k \wedge (\forall n.\ lsk\ n < lsk\ (Suc\ n)) \wedge$
      $(\forall i.\ ls\ i = lsk\ i - k) \wedge ls\ 0 = 0 \wedge (\forall n.\ ls\ n < ls\ (Suc\ n))$
    by *auto*
    show $lsk\ 0 = k \wedge (\forall n.\ lsk\ n < lsk\ (Suc\ n)) \wedge$
      $(\forall i.\ ls\ i = lsk\ i - k) \wedge ls\ 0 = 0 \wedge (\forall n.\ ls\ n < ls\ (Suc\ n)) \Longrightarrow$
     $lsk\ 0 = k \wedge (\forall n.\ lsk\ n < lsk\ (Suc\ n)) \wedge$
      $(\forall i.\ lsk\ i = ls\ i + k) \wedge ls\ 0 = 0 \wedge (\forall n.\ ls\ n < ls\ (Suc\ n))$
    by (*metis add.commute add-diff-inverse-nat add-less-same-cancel1 lift-Suc-mono-less-iff*
     not-add-less1)
   qed
qed

lemma *iidx-7*:
  $(infinite\text{-}index\text{-}sequence\ k\ lsk \wedge ls = (shiftm\ k) \circ lsk) =$
   $(infinite\text{-}index\text{-}sequence\ k\ lsk \wedge ls = (shiftm\ k) \circ lsk \wedge$
    $infinite\text{-}index\text{-}sequence\ 0\ ls)$
using *iidx-5* **by** *blast*

lemma *iidx-8*:
  $(lsk = (shift\ k) \circ ls \wedge infinite\text{-}index\text{-}sequence\ 0\ ls) =$
   $(lsk = (shift\ k) \circ ls \wedge infinite\text{-}index\text{-}sequence\ k\ lsk \wedge$
    $infinite\text{-}index\text{-}sequence\ 0\ ls)$
by (*metis Interval.shift-def add-left-imp-eq diff-is-0-eq' interval-idx-shift-mono*
     le-add-diff-inverse2 mono-def iidx-4 rel-simps(*46*))

lemma *iidx-0-a*:
assumes *infinite-index-sequence 0 l*
 shows $(l\ 0) = 0 \wedge (l\ 0) < (l\ 1) \wedge infinite\text{-}index\text{-}sequence\ (l\ 1)\ (\lambda x.\ l(x+1))$
using *assms* **by** (*simp add*: *infinite-index-sequence-def*) *metis*

lemma *iidx-0-b*:
assumes $x = 0$
$x < (l\ 0)$
infinite-index-sequence $(l\ 0)\ l$
shows *infinite-index-sequence* 0 $(conc\ \langle x\rangle\ l)$
using *assms diff-Suc-less infinite-index-sequence-def*
by (*simp add*: *conc-def lift-Suc-mono-less*)

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

end

5 Old vs new definition of Chopstar

theory *AltChopstarSem*
imports *Semantics*
begin

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

5.1 Definition

definition *chopstar-d-old* :: $('a{::}world)\ formula \Rightarrow 'a\ formula$
where $\mathit{chopstar\text{-}d\text{-}old}\ F \equiv \lambda s.\ (\exists\ (l{::}index).\ \mathit{index\text{-}sequence}\ 0\ l\ \wedge (nth\ l\ (intlen\ l)) = (intlen\ s) \wedge$
$(\forall i.\ (0 \le i \wedge i < (intlen\ l)) \longrightarrow$
$((sub\ (nth\ l\ i)\ (nth\ l\ (i+1))\ s) \models F)$
)
)

syntax
-chopstar-d-old :: *lift* ⇒ *lift* ((*chopstarold* -) [*85*] *85*)

syntax (*ASCII*)
-chopstar-d-old :: *lift* ⇒ *lift* ((*chopstarold* -) [*85*] *85*)

translations
-chopstar-d-old ⇌ *CONST chopstar-d-old*

5.2 Lemmas

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

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

lemma chopstar-help-2:
 (∀ i. (0<i ∧ i< 1+(intlen ls)) ⟶
                              ((sub (nth ls (i−1)) (nth ls (i)) σ) ⊨ f)
                          ) =
  (∀ i. (0≤i ∧ i< (intlen ls)) ⟶
                              ((sub (nth ls (i)) (nth ls ((i)+1)) σ) ⊨ 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:
  (∃ (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)
                  ))
        )
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)
                          )
            )
 by (metis interval-intlen-cons-1 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 ... =
```

($\exists$ *x ls* . (*intlen ls*) = *n* $\wedge$ *index-sequence 0* (*x*$\odot$*ls*) $\wedge$
(*nth* (*x*$\odot$*ls*) (*intlen* (*x*$\odot$*ls*))) = (*intlen* σ) $\wedge$
($\forall$ *i*. (*0*$\leq$*i* $\wedge$ *i*< (*intlen* (*x*$\odot$*ls*))) $\longrightarrow$
((*sub* (*nth* (*x*$\odot$*ls*) *i*) (*nth* (*x*$\odot$*ls*) (*i*+*1*)) σ) $\models$ *f*)
)
)
by *auto*
also have ... =
($\exists$ *x ls* . (*intlen ls*) = *n* $\wedge$ *x* =*0* $\wedge$ *index-sequence 0* (*x* $\odot$ *ls*) $\wedge$
(*nth* (*ls*) (*intlen* (*ls*))) = (*intlen* σ) $\wedge$
(($\forall$ *i*. (*0*$\leq$*i* $\wedge$ *i*< (*intlen* (*x*$\odot$*ls*))) $\longrightarrow$
((*sub* (*nth* (*x*$\odot$*ls*) *i*) (*nth* (*x*$\odot$*ls*) (*i*+*1*)) σ) $\models$ *f*))
)
)
by (*simp add*: *index-sequence-def*)
also have ... =
($\exists$ *x ls* . (*intlen ls*) = *n* $\wedge$ *x* =*0* $\wedge$ *index-sequence* (*nth ls 0*) (*ls*) $\wedge$
(*nth* (*ls*) (*intlen* (*ls*))) = (*intlen* σ) $\wedge$
(*x* < (*nth ls 0*) $\wedge$
(($\forall$ *i*. (*0*$\leq$*i* $\wedge$ *i*< (*intlen* (*x*$\odot$*ls*))) $\longrightarrow$
((*sub* (*nth* (*x*$\odot$*ls*) *i*) (*nth* (*x*$\odot$*ls*) (*i*+*1*)) σ) $\models$ *f*))
)
)
)
using *interval-idx-cons* **by** *auto*
also have ... =
($\exists$ *x ls* . (*intlen ls*) = *n* $\wedge$ *x* =*0* $\wedge$ *index-sequence* (*nth ls 0*) (*ls*) $\wedge$
(*nth* (*ls*) (*intlen* (*ls*))) = (*intlen* σ) $\wedge$
(*x* < (*nth ls 0*) $\wedge$
((*sub* (*nth* (*x*$\odot$*ls*) *0*) (*nth* (*x*$\odot$*ls*) (*1*)) σ) $\models$ *f*)
$\wedge$
(($\forall$ *i*. (*0*<*i* $\wedge$ *i*< *1*+ (*intlen* (*ls*))) $\longrightarrow$
((*sub* (*nth* (*x*$\odot$*ls*) *i*) (*nth* (*x*$\odot$*ls*) (*i*+*1*)) σ) $\models$ *f*))
)
)
)
by (*metis* (*no-types*, *lifting*) *One-nat-def add.right-neutral add-Suc add-Suc-right add-cancel-right-left interval-intlen-cons not-gr-zero zero-le zero-less-Suc*)
also have ... =
($\exists$ *x ls* . (*intlen ls*) = *n* $\wedge$ *x* =*0* $\wedge$ *index-sequence* (*nth ls 0*) (*ls*) $\wedge$
(*nth* (*ls*) (*intlen* (*ls*))) = (*intlen* σ) $\wedge$
(*x* < (*nth ls 0*) $\wedge$ (*nth* (*x*$\odot$*ls*) *0*) = *x* $\wedge$ (*nth* (*x*$\odot$*ls*) (*1*)) = (*nth ls 0*) $\wedge$
((*sub* (*nth* (*x*$\odot$*ls*) *0*) (*nth* (*x*$\odot$*ls*) (*1*)) σ) $\models$ *f*)
$\wedge$
(($\forall$ *i*. (*0*<*i* $\wedge$ *i*< *1*+ (*intlen* (*ls*))) $\longrightarrow$
((*sub* (*nth* (*x*$\odot$*ls*) *i*) (*nth* (*x*$\odot$*ls*) (*i*+*1*)) σ) $\models$ *f*))
)
)
)
by *auto*

also have ... =

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

by *auto*

also have ... =

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

by *auto*

also have ... =

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

using *interval-nth-cons* **by** (*metis* (*no-types*, *lifting*))

also have ... =

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

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

also have ... =

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

((*sub* (*nth ls* (*i*)) (*nth ls* ((*i*)+*1*)) σ) ⊨ *f*)
)
)
by *simp*
also have ... =
(∃ *lsk* . (*intlen lsk*) = *n* ∧ (*nth lsk 0*) ≤ *intlen* σ ∧ (*nth lsk 0*) > *0* ∧
((*sub 0* (*nth lsk 0*) σ) ⊨ *f*) ∧
index-sequence (*nth lsk 0*) (*lsk*) ∧
(*nth* (*lsk*) (*intlen* (*lsk*))) = (*intlen* σ) ∧
(∀ *i*. (*0*≤*i* ∧ *i*< (*intlen lsk*)) ⟶
((*sub* (*nth lsk* (*i*)) (*nth lsk* ((*i*)+*1*)) σ) ⊨ *f*)
)
)
by (*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 ... =
(∃ *k lsk*. (*intlen lsk*) = *n* ∧ (*nth lsk 0*) ≤ *intlen* σ ∧
(*nth lsk 0*) > *0* ∧ *k*=(*nth lsk 0*) ∧
(*sub 0* (*nth lsk 0*) σ ⊨ *f*) ∧
index-sequence (*nth lsk 0*) (*lsk*) ∧
(*nth* (*lsk*) (*intlen* (*lsk*))) = (*intlen* (σ)) ∧
(∀ *i*. (*0*≤*i* ∧ *i*< (*intlen lsk*)) ⟶
((*sub* ((*nth lsk* (*i*))) ((*nth lsk* ((*i*)+*1*))) (σ)) ⊨ *f*)
)
)
by *auto*
also have ... =
(∃ *k lsk*. (*intlen lsk*) = *n* ∧ *0* ≤*k* ∧ *k* ≤ *intlen* σ ∧ *k* > *0* ∧ *k*=(*nth lsk 0*) ∧
(*sub 0 k* σ ⊨ *f*) ∧
(*index-sequence k* (*lsk*) ∧
(*nth* (*lsk*) (*intlen* (*lsk*))) = ((*intlen* (*suffix k* σ))+*k*) ∧
(∀ *i*. (*0*≤*i* ∧ *i*< (*intlen lsk*)) ⟶
((*sub* ((*nth lsk* (*i*))) ((*nth lsk* ((*i*)+*1*))) (σ)) ⊨ *f*)
))
)
by *auto*
also have ... =
(∃ *k lsk*. (*intlen lsk*) = *n* ∧ *0* ≤*k* ∧ *k* ≤ *intlen* σ ∧ *k* > *0* ∧
(*sub 0 k* σ ⊨ *f*) ∧
(*index-sequence k* (*lsk*) ∧
(*nth* (*lsk*) (*intlen* (*lsk*))) = ((*intlen* (*suffix k* σ))+*k*) ∧
(∀ *i*. (*0*≤*i* ∧ *i*< (*intlen lsk*)) ⟶
((*sub* ((*nth lsk* (*i*))) ((*nth lsk* ((*i*)+*1*))) (σ)) ⊨ *f*)
))
)
using *index-sequence-def* **by** *auto*
also have ... =
(∃ *k*. *0* ≤*k* ∧ *k* ≤ *intlen* σ ∧ *k* > *0* ∧
(*sub 0 k* σ ⊨ *f*) ∧
(∃ *ls lsk*. (*intlen lsk*) = *n* ∧ *index-sequence k* (*lsk*) ∧

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

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

$(\forall\, i.\ (0 \le i \wedge i < (intlen\ l)) \longrightarrow$
$((sub\ (nth\ l\ i)\ (nth\ l\ (i+1))\ \sigma) \models f)$
$)$
$) =$
$(\exists\ k.\ 0 \le k \wedge k \le intlen\ \sigma \wedge k > 0 \wedge (sub\ 0\ k\ \sigma \models f) \wedge$
$(\exists\ ls.\ (intlen\ ls) = n \wedge index\text{-}sequence\ 0\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ (suffix\ k\ \sigma)) \wedge$
$(\forall\, i.\ (0 \le i \wedge i < (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i))\ (nth\ ls\ ((i)+1))\ (suffix\ k\ \sigma)) \models f)$
$)$
$)$
$)$
.
qed

lemma *chop-power-eqv-sem*:
$(\ (\sigma \models (\exists\ n.\ (power\ (f \wedge more)\ n)))) =$
$((\sigma \models empty) \vee\ (\ (\sigma \models (f \wedge more);(\exists\ n.\ (power\ (f \wedge more)\ n)))))$
using *ChopstarEqvSem powerstar-d-def chopstar-d-def*
by (*metis* (*mono-tags*, *lifting*) *unl-lift2*)

lemma *chopstar-eqv-power-chop-help*:
$(\ \sigma \models power\ (f \wedge more)\ n) =$
$(\exists\ (l{::}index).\ intlen(l) = n \wedge\ index\text{-}sequence\ 0\ l\ \wedge$
$(nth\ l\ (intlen\ l)) = (intlen\ (\ \sigma)) \wedge$
$(\forall\, i.\ (0 \le i \wedge i < (intlen\ l)) \longrightarrow$
$((sub\ (nth\ l\ i)\ (nth\ l\ (i+1))\ (\sigma)) \models f)$
$)$
$)$
proof
(*induct n arbitrary*: σ)
case *0*
then show *?case* **using** *index-sequence-def chopstar-help-1 empty-defs*
by (*metis* (*mono-tags*, *lifting*) *intlen.simps*(*1*) *pow-0*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: $(\sigma \models power\ (f \wedge more)\ (Suc\ n)) = (\sigma \models ((f \wedge more);(power\ (f \wedge more)\ n)))$
by *simp*
have *2*: $(\sigma \models ((f \wedge more);(power\ (f \wedge more)\ n))) =$
$(\exists\ k.\ 0 \le k \wedge k \le intlen\ (\sigma) \wedge k > 0 \wedge$
$(prefix\ k\ (\sigma) \models f) \wedge$
$(suffix\ k\ (\sigma) \models power\ (f \wedge more)\ n)$
$)$

by (*simp add*: *more-defs chop-defs*) *auto*
have *3*: $(\exists\ k.\ 0 \le k \wedge k \le intlen\ (\sigma) \wedge k > 0 \wedge$
$(prefix\ k\ (\sigma) \models f) \wedge$
$(suffix\ k\ (\sigma) \models power\ (f \wedge more)\ n)$

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

lemma *chopstar-eqv-power-chop*:

$(\sigma \models chopstarold\ f) = (\ (\sigma \models (\exists\ k.\ power\ (f \wedge more)\ k)))$
by (*simp add*: *chopstar-d-old-def chopstar-eqv-power-chop-help*)

lemma *OldChopstarEqvSem*:
$(\sigma \models (chopstarold\ f = \ (empty \vee \ (f \wedge more);\ (chopstarold\ f)))\ )$
proof −
have *1*: $(\sigma \models chopstarold\ f) = (\ (\sigma \models (\exists\ k.\ power\ (f \wedge more)\ k)))$
using *chopstar-eqv-power-chop* **by** *simp*
have *2*: $(\ (\sigma \models (\exists\ k.\ power\ (f \wedge more)\ k))) =$
$(\ (\sigma \models empty) \vee (\sigma \models (f \wedge more);(\exists\, n.\ power\ (f \wedge more)\ n)))$
using *chop-power-eqv-sem* **by** *simp*
have *3*: $(\sigma \models (f \wedge more);(\exists\, n.\ power\ (f \wedge more)\ n)) =$
$(\ \exists\, n \le intlen\ \sigma.\ ((prefix\ n\ \sigma) \models f\ \wedge more) \wedge$
$((suffix\ n\ \sigma) \models (\exists\, x.\ (power\ (f \wedge more)\ x)\ )))$
by (*simp add*: *chop-defs*)
have *4*: $(\ \exists\, n \le intlen\ \sigma.\ ((prefix\ n\ \sigma) \models f\ \wedge more) \wedge$
$((suffix\ n\ \sigma) \models (\exists\, x.\ (power\ (f \wedge more)\ x)\ ))) =$
$(\ \exists\, n \le intlen\ \sigma.\ ((prefix\ n\ \sigma) \models f\ \wedge more) \wedge ((suffix\ n\ \sigma) \models\ chopstarold\ f\ ))$
by (*simp add*: *chopstar-eqv-power-chop*)
have *5*: $(\ \exists\, n \le intlen\ \sigma.\ ((prefix\ n\ \sigma) \models f\ \wedge more) \wedge ((suffix\ n\ \sigma) \models\ chopstarold\ f\ )) =$
$(\sigma \models\ (f \wedge more);\ (chopstarold\ f)\ )$
by (*simp add*: *chop-defs*)
have *6*: $(\ (\sigma \models empty) \vee (\sigma \models (f \wedge more);(\exists\, n.\ power\ (f \wedge more)\ n))) =$
$(\ \sigma \models (empty \vee \ (f \wedge more);\ (chopstarold\ f)))$
using *3 4 5* **by** *auto*
show *?thesis* **using** *1 2 6* **by** *auto*
qed

lemma *OldChopstarEqvChopstar*:
$\vdash (chopstarold\ f) = f^{\star}$
by (*simp add*: *Valid-def chopstar-d-def chopstar-eqv-power-chop powerstar-d-def*)

end

6 Finite ITL: Axioms and Rules

theory *ITL*
imports
Semantics
begin

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

6.1 Rules

lemma *MP* :
assumes $\vdash f \longrightarrow g$

$\vdash f$
shows $\vdash g$
using *assms*(1) *assms*(2) **by** *fastforce*

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

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

6.2 Axioms

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

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

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

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

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

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

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

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

lemma *BoxInduct* :
$\vdash \Box\ (f \longrightarrow wnext\ f) \land f \longrightarrow \Box\ f$

using *BoxInductSem Valid-def* **by** *blast*

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

6.3 Additional Lemmas

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

lemma *int-eq-true*:
 assumes $\vdash P$
 shows $\vdash P = \#True$
  using *assms* **by** *auto*

lemma *int-eq*:
 assumes $\vdash X = Y$
 shows $X = Y$
  using *assms* **by** (*auto simp*: *inteq-reflection*)

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

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

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

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

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

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

6.4 Quantification

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

lemma *EExE*:
assumes $\bigwedge x.\ \vdash F\ x \longrightarrow G$
shows $\vdash (\exists\exists\ x.\ F\ x) \longrightarrow G$
using *assms* **by** (*metis* (*mono-tags*, *lifting*) *Valid-def exist-state-d-def unl-lift2*)

lemma *EExVal*:
$(w \models (\exists\exists\ x.\ F\ x)) =$
$(\exists\ x\ (val :: 'a\ interval).\ (\ (\ val = (map\ x\ w) \land\ (w \models F\ x))))$
by (*simp add*: *exist-state-d-def*)

lemma *AAxDef*:
$\vdash (\forall\forall\ x.\ F\ x) = (\neg(\exists\exists\ x.\ \neg(F\ x)))$
by (*simp add*: *Valid-def forall-state-d-def exist-state-d-def*)

lemma *ExEqvRule*:
assumes $\bigwedge\ x.\ \vdash (f\ x) = (g\ x)$
shows $\vdash (\exists\ x.\ f\ x) = (\exists\ x.\ g\ x)$
using *assms* **by** *fastforce*

6.5 Lemmas about *current-val*

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

lemma *current-fun1*: $\vdash \$(f{<}x{>}) = f\ {<}\$x{>}$
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*]

6.6 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*]

6.7 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*]

6.8 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*]

6.9 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*)

end

7 Finite ITL theorems

theory *Theorems*
imports
ITL
begin

We give the proofs of a list of Finite ITL theorems. These proofs and theorems were from [8].

7.1 Propositional reasoning

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

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

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

lemma *Prop02*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
$\vdash$ *f1* $\longrightarrow$ *g*
shows $\vdash$ *f* $\vee$ *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*)

7.2 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*, *fastforce*)

7.3 Basic Theorems

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

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

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

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

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

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

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

lemma *RightChopEqvChop*:
assumes $\vdash g = g1$
shows $\vdash (f;g) = (f;g1)$
using *assms RightChopImpChop*[*of g g1 f*] *RightChopImpChop*[*of g1 g f*]
by *fastforce*

lemma *ChopOrEqv*:
$\vdash f;(g \vee g1) = (f;g \vee f;g1)$
proof $-$
have *1*: $\vdash g \longrightarrow g \vee g1$ **by** *auto*
hence *2*: $\vdash f;g \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
have *3*: $\vdash g1 \longrightarrow g \vee g1$ **by** *auto*
hence *4*: $\vdash f;g1 \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
from *2 4* **show** *?thesis* **by** (*meson ChopOrImp Prop02 Prop11*)
qed

lemma *OrChopEqv*:
$\vdash (f \vee f1);g = (f;g \vee f1;g)$
proof $-$
have *1*: $\vdash f \longrightarrow f \vee f1$ **by** *auto*
hence *2*: $\vdash f;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash f1 \longrightarrow f \vee f1$ **by** *auto*
hence *4*: $\vdash f1;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
from *2 4* **show** *?thesis*
by (*meson OrChopImp int-iffI Prop02*)
qed

lemma *OrChopImpRule*:
assumes $\vdash f \longrightarrow f1 \vee f2$

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

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

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

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

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

lemma *NextImpDist*:
$\vdash \bigcirc (f \longrightarrow g) \longrightarrow \bigcirc f \longrightarrow \bigcirc g$
proof −
have *1*: $\vdash (\neg (f \longrightarrow g)) = (f \wedge \neg g)$ **by** *auto*
hence *2*: $\vdash skip;(\neg (f \longrightarrow g)) = skip;(f \wedge \neg g)$ **by** (*rule RightChopEqvChop*)
have *3*: $\vdash f \longrightarrow g \vee (f \wedge \neg g)$ **by** *auto*
hence *4*: $\vdash skip;f \longrightarrow (skip;g) \vee (skip;(f \wedge \neg g))$ **by** (*rule ChopOrImpRule*)
hence *5*: $\vdash \neg (skip;(f \wedge \neg g)) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** *auto*
have *6*: $\vdash \neg (skip;(\neg(f \longrightarrow g))) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **using** *2 5* **by** *fastforce*
hence *7*: $\vdash \neg (\bigcirc(\neg(f \longrightarrow g))) \longrightarrow (\bigcirc f) \longrightarrow (\bigcirc g)$ **by** (*simp add*: *next-d-def*)
have *8*: $\vdash \bigcirc(f \longrightarrow g) \longrightarrow \neg (\bigcirc(\neg(f \longrightarrow g)))$ **by** (*rule NextImpNotNextNot*)
from *7 8* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

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

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

lemma *BoxElim*:
$\vdash \Box f \longrightarrow f$
proof −
have *1*: $\vdash \neg f \longrightarrow \Diamond (\neg f)$ **by** (*rule NowImpDiamond*)
hence *2*: $\vdash \neg (\Diamond (\neg f)) \longrightarrow f$ **by** *auto*
from *2* **show** *?thesis* **by** (*metis always-d-def*)
qed

lemma *NextDiamondImpDiamond*:
$\vdash \bigcirc (\Diamond f) \longrightarrow \Diamond f$
proof −
have *1*: $\vdash skip;(\#True;f) = ((skip;\#True);f)$ **by** (*rule ChopAssoc*)
hence *2*: $\vdash (skip;\#True);f = skip;(\#True;f)$ **by** *auto*
hence *3*: $\vdash (skip;\#True);f = \bigcirc(\Diamond f)$ **by** (*simp add*: *next-d-def sometimes-d-def*)
have *4*: $\vdash (skip;\#True);f \longrightarrow \Diamond f$ **by** (*rule ChopImpDiamond*)
from *3 4* **show** *?thesis* **by** *fastforce*

qed

lemma *BoxImpNowAndWeakNext*:
$\vdash \Box\, f \longrightarrow (f \wedge \mathit{wnext}\ (\ \Box\ f)\ )$
proof –
have *1*: $\vdash \neg\, f \longrightarrow \Diamond\ (\neg\ f)$ **by** (*rule NowImpDiamond*)
hence *2*: $\vdash \neg\ (\Diamond\ (\neg\ f)) \longrightarrow f$ **by** *auto*
hence *3*: $\vdash \Box\ f \longrightarrow f$ **by** (*metis always-d-def*)
have *4*: $\vdash \bigcirc\ (\ \Diamond\ (\neg\ f)) \longrightarrow \Diamond\ (\ \neg\ f\ )$ **by** (*rule NextDiamondImpDiamond*)
have *5*: $\vdash \neg\ \neg\ (\Diamond\ (\neg\ f)) \longrightarrow \Diamond(\ \neg\ f\ )$ **by** *auto*
hence *6*: $\vdash \bigcirc\ (\ \neg\ \neg\ (\Diamond\ (\neg\ f))\ ) \longrightarrow \bigcirc\ (\Diamond(\ \neg\ f\ ))$ **by** (*rule NextImpNext*)
have *7*: $\vdash \bigcirc\ (\ \neg\ \neg\ (\Diamond\ (\neg\ f))\ ) \longrightarrow \Diamond\ (\ \neg\ f\ )$ **using** *4 6* **by** *auto*
hence *8*: $\vdash \bigcirc\ (\ \neg(\ \Box\ f)) \longrightarrow \Diamond\ (\ \neg\ f\ )$ **by** (*simp add*: *always-d-def*)
hence *9*: $\vdash \neg\ (\Diamond\ (\ \neg\ f\ )) \longrightarrow \neg\ (\ \bigcirc\ (\ \neg(\ \Box\ f)))$ **by** *auto*
hence *10*: $\vdash \Box f \longrightarrow \mathit{wnext}\ (\ \Box\ f\ )$ **by** (*simp add*: *always-d-def wnext-d-def*)
from *3 10* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpBoxRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \Box\ f \longrightarrow \Box\ g$
proof –
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ g \longrightarrow \neg\ f$ **by** *auto*
hence *3*: $\vdash \Box(\neg\ g \longrightarrow \neg\ f)$ **by** (*rule BoxGen*)
have *4*: $\vdash \Box(\neg\ g \longrightarrow \neg\ f) \longrightarrow (\#\mathit{True};(\neg g)) \longrightarrow (\#\mathit{True};(\neg f))$ **by** (*rule BoxChopImpChop*)
have *5*: $\vdash (\#\mathit{True};(\neg g)) \longrightarrow (\#\mathit{True};(\neg f))$ **using** *3 4 MP* **by** *blast*
hence *6*: $\vdash \Diamond\ (\neg g) \longrightarrow \Diamond(\neg f)$ **by** (*simp add*: *sometimes-d-def*)
hence *7*: $\vdash \neg\ (\Diamond(\neg f)\ ) \longrightarrow \neg(\ \Diamond\ (\neg g))$ **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *always-d-def*)
qed

lemma *BoxImpDist*:
$\vdash \Box(f \longrightarrow g) \longrightarrow \Box\ f \longrightarrow \Box\ g$
proof –
have *1*: $\vdash (f \longrightarrow g) \longrightarrow (\neg\ g \longrightarrow \neg\ f)$ **by** *auto*
hence *2*: $\vdash \Box(f \longrightarrow g) \longrightarrow \Box(\neg\ g \longrightarrow \neg\ f)$ **by** (*rule BoxImpBoxRule*)
have *3*: $\vdash \Box((\neg\ g) \longrightarrow \neg\ f) \longrightarrow (\#\mathit{True};\ (\neg\ g)) \longrightarrow (\#\mathit{True};\ (\neg\ f))$ **by** (*rule BoxChopImpChop*)
have *4*: $\vdash \Box(f \longrightarrow g) \longrightarrow (\#\mathit{True};\ (\neg\ g)) \longrightarrow (\#\mathit{True};\ (\neg\ f))$
using *2 3 lift-imp-trans* **by** *blast*
hence *5*: $\vdash \Box(f \longrightarrow g) \longrightarrow \Diamond(\neg\ g) \longrightarrow \Diamond(\neg\ f)$ **by** (*simp add*: *sometimes-d-def*)
hence *6*: $\vdash \Box(f \longrightarrow g) \longrightarrow \neg(\ \Diamond(\neg\ f)) \longrightarrow \neg(\ \Diamond(\neg\ g))$ **by** *auto*
from *6* **show** *?thesis* **by** (*simp add*: *always-d-def*)
qed

lemma *DiamondEmpty*:
$\vdash \Diamond\ \mathit{empty}$
proof –
have *1*: $\vdash \#\mathit{True}$ **by** *auto*
have *2*: $\vdash \#\mathit{True};\ \mathit{empty} = \#\mathit{True}$ **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*: $\vdash \Box\ (\neg\ g) = ((\neg\ g) \wedge wnext(\Box\ (\neg\ g)))$
  using *BoxEqvAndWnextBox* **by** *metis*
have *7*: $\vdash \bigcirc\ f \wedge \Box\ (\neg\ g) \longrightarrow\ \bigcirc\ f \wedge wnext(\Box\ (\neg\ g))$
  using *6* **by** *auto*
have *8*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow \bigcirc\ f \wedge wnext(\Box\ (\neg\ g))$
  using *5 7* **using** *lift-imp-trans* **by** *blast*
hence *9*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow more \wedge wnext\ f \wedge wnext(\Box\ (\neg\ g))$
  by (*auto simp*: *always-defs more-defs next-defs wnext-defs*)
hence *10*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ f \wedge wnext(\Box\ (\neg\ g))$
  by *auto*
hence *11*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ (f \wedge \Box\ (\neg\ g))$
  by (*auto simp*: *wnext-defs always-defs next-defs*)
hence *12*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ (f \wedge \Box\ (\neg\ g)))$
  by (*rule BoxGen*)
have *13*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ (f \wedge \Box\ (\neg\ g))) \wedge f \wedge (\Box\ (\neg\ g)) \longrightarrow \Box(f \wedge (\Box\ (\neg\ g)))$
  by (*rule BoxInduct*)
hence *14*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow wnext\ (f \wedge \Box\ (\neg\ g))) \longrightarrow ((f \wedge (\Box\ (\neg\ g))) \longrightarrow \Box(f \wedge (\Box\ (\neg\ g))))$
  by *fastforce*
have *15*: $\vdash ((f \wedge (\Box\ (\neg\ g))) \longrightarrow \Box(f \wedge (\Box\ (\neg\ g))))$
  using *12 14 MP* **by** *blast*
have *16*: $\vdash \Box(f \wedge (\Box\ (\neg\ g))) \longrightarrow (f \wedge (\Box\ (\neg\ g)))$
  by (*rule BoxElim*)
have *17*: $\vdash \Box(f \wedge (\Box\ (\neg\ g))) = (f \wedge (\Box\ (\neg\ g)))$
  using *16 15* **by** *fastforce*
have *18*: $\vdash (f \wedge (\Box\ (\neg\ g))) \longrightarrow more$
  using *9* **by** *auto*
hence *19*: $\vdash \Box(f \wedge (\Box\ (\neg\ g))) \longrightarrow \Box\ more$
  by (*rule ImpBoxRule*)
have *20*: $\vdash \neg(\Box\ more)$
  by (*auto simp*: *always-defs more-defs*)
have *21*: $\vdash \neg(f \wedge (\Box\ (\neg\ g)))$
  using *17 19 20* **by** *fastforce*
hence *22*: $\vdash \neg\ f \vee \neg\ (\Box\ (\neg\ g))$
  by *auto*
have *23*: $\vdash (\neg\ (\Box\ (\neg\ g))) = \Diamond\ g$
  by (*auto simp*: *always-d-def*)
from *22 23* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondIntroB*:
assumes $\vdash (f \wedge \neg\ g) \longrightarrow \bigcirc\ (f \wedge \neg\ g)$
shows $\vdash f \longrightarrow \Diamond\ g$
proof −
have *1*: $\vdash (f \wedge \neg\ g) \longrightarrow \bigcirc\ (f \wedge \neg g)$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg(f \wedge \neg\ g)$ **by** (*rule NextLoop*)
hence *3*: $\vdash f \longrightarrow g$ **by** *auto*
have *4*: $\vdash g \longrightarrow \Diamond\ g$ **by** (*rule NowImpDiamond*)
from *3 4* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NextContra* :
assumes $\vdash (f \wedge \neg\ g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$
shows $\vdash f \longrightarrow g$
proof –
have *1*: $\vdash (f \wedge \neg\ g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg(f \longrightarrow g) \longrightarrow \bigcirc(\neg(f \longrightarrow g))$ **by** (*auto simp*: *next-defs Valid-def*)
hence *3*: $\vdash \neg\neg(f \longrightarrow g)$ **by** (*rule NextLoop*)
from *3* **show** *?thesis* **by** *auto*
qed

lemma *DiamondDiamondEqvDiamond*:
$\vdash \Diamond(\Diamond f) = \Diamond f$
proof –
have *1*: $\vdash$ *#True*;*#True* = *#True* **by** (*auto simp*: *chop-defs*)
hence *2*: $\vdash$ (*#True*;*#True*);*f* = *#True*;*f* **using** *LeftChopEqvChop* **by** *blast*
have *3*: $\vdash$ (*#True*;*#True*);*f* = *#True*;(*#True*;*f*) **using** *ChopAssoc* **by** *fastforce*
from *2 3* **show** *?thesis* **by** (*metis inteq-reflection sometimes-d-def*)
qed

lemma *WeakNextDiamondInduct*:
assumes $\vdash$ *wnext* $(\Diamond f) \longrightarrow f$
shows $\vdash f$
proof –
have *1*: $\vdash$ *wnext* $(\Diamond f) \longrightarrow f$ **using** *assms* **by** *blast*
hence *2*: $\vdash \neg f \longrightarrow \neg($ *wnext* $(\Diamond f))$ **by** *fastforce*
hence *3*: $\vdash \neg f \longrightarrow \bigcirc(\neg(\Diamond f))$ **by** (*simp add*: *wnext-d-def*)
have *4*: $\vdash f \longrightarrow \Diamond f$ **by** (*rule NowImpDiamond*)
hence *5*: $\vdash \neg(\Diamond f) \longrightarrow \neg f$ **by** *auto*
have *6*: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **using** *3 5* **using** *NextImpNext lift-imp-trans* **by** *blast*
hence *7*: $\vdash \neg\neg f$ **by** (*rule NextLoop*)
from *7* **show** *?thesis* **by** *auto*
qed

lemma *EmptyNextInducta*:
assumes $\vdash$ *empty* $\longrightarrow f$
$\vdash \bigcirc f \longrightarrow f$
shows $\vdash f$
proof –
have *1*: $\vdash$ *empty* $\longrightarrow f$ **using** *assms* **by** *auto*
have *2*: $\vdash \bigcirc f \longrightarrow f$ **using** *assms* **by** *blast*
have *3*: $\vdash$ (*empty* $\vee \bigcirc f) \longrightarrow f$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash$ *wnext f* = (*empty* $\vee \bigcirc f$) **by** (*rule WnextEqvEmptyOrNext*)
hence *5*: $\vdash$ *wnext f* $\longrightarrow f$ **using** *3* **by** *fastforce*
hence *6*: $\vdash \neg f \longrightarrow \neg$ (*wnext f*) **by** *auto*
hence *7*: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **by** (*auto simp*: *wnext-d-def*)
hence *8*: $\vdash \neg\neg f$ **by** (*rule NextLoop*)
from *8* **show** *?thesis* **by** *auto*
qed

lemma *EmptyNextInductb*:
assumes $\vdash$ *empty* $\wedge$ *f* $\longrightarrow$ *g*
$\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$
shows $\vdash f \longrightarrow g$
proof –
have *1*: $\vdash$ *empty* $\wedge$ *f* $\longrightarrow$ *g* **using** *assms* **by** *auto*
have *2*: $\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *assms* **by** *blast*
have *3*: $\vdash$ (*empty* $\vee \bigcirc(f \longrightarrow g)) \wedge f \longrightarrow g$ **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *wnext* $(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*
hence *5*: $\vdash$ *wnext* $(f \longrightarrow g) \longrightarrow (f \longrightarrow g)$ **by** *fastforce*
hence *6*: $\vdash \neg (f \longrightarrow g) \longrightarrow \neg$ (*wnext* $(f \longrightarrow g)$) **by** *fastforce*
hence *7*: $\vdash \neg (f \longrightarrow g) \longrightarrow \bigcirc (\neg(f \longrightarrow g))$ **by** (*simp add*: *wnext-d-def*)
hence *8*: $\vdash \neg \neg (f \longrightarrow g)$ **by** (*rule NextLoop*)
from *8* **show** *?thesis* **by** *auto*
qed

lemma *FinImpFin*:
assumes $\vdash f \longrightarrow g$
shows $\vdash$ *fin* *f* $\longrightarrow$ *fin* *g*
using *ImpBoxRule*[*of LIFT* (*empty* $\longrightarrow$ *f*) *LIFT* (*empty* $\longrightarrow$ *g*)] *assms*
fin-d-def [*of f*] *fin-d-def* [*of g*] **by** *fastforce*

lemma *FinEqvFin*:
assumes $\vdash f = g$
shows $\vdash$ *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*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*empty* = *f*) = (*empty* = *g*) **by** *auto*
hence *3*: $\vdash \Box$(*empty* = *f*) = $\Box$ (*empty* = *g*) **by** (*rule BoxEqvBox*)
from *3* **show** *?thesis* **by** (*simp add*: *halt-d-def*)

qed

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

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

lemma *BiImpBiRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash$ *bi* $f \longrightarrow$ *bi* g
proof $-$
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ g \longrightarrow \neg\ f$ **by** *auto*
hence *3*: $\vdash$ *di* $(\neg\ g) \longrightarrow$ *di* $(\neg\ f)$ **by** (*rule DiImpDi*)
hence *4*: $\vdash \neg$ (*di* $(\neg\ f)) \longrightarrow \neg$ (*di* $(\neg\ g))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

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

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

lemma *LeftChopChopImpChopRule*:

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

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

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

lemma *StateAndChopImport*:
$\vdash$ (*init* $w) \wedge (f;\, g) \longrightarrow (($*init* $w) \wedge f);\, g$
proof −
have *1*: $\vdash$ (*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

7.4 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: ⊢ f; g = (((init w) ∧ f1); g)  by (rule LeftChopEqvChop)
 have  3: ⊢ ((init w) ∧ f1); g = ((init w) ∧ (f1; g))  by (rule StateAndChop)
 from 2 3 show ?thesis by fastforce
qed

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

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

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

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

lemma IfChopEqvRule:
 assumes ⊢ f =  if_i  (init w)  then  f1  else  f2
```

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

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

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

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

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

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

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

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

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

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

lemma *StateAndEmptyImpBoxState*:
$\vdash (init\ w) \wedge empty \longrightarrow \Box\, (init\ w)$
by (*simp add*: *init-defs empty-defs always-defs Valid-def*)

lemma *BoxEqvAndBox*:
$\vdash \Box\, f = (f \wedge \Box\, f)$
by (*simp add*: *always-defs Valid-def*) *fastforce*

lemma *NotBoxImpNotOrNotNextBox*:
$\vdash \neg(\Box f) \longrightarrow \neg f \vee \neg(\bigcirc (\Box f))$
proof −
 have *1*: $\vdash f \wedge (\bigcirc (\Box f)) \longrightarrow \Box f$
  using *BoxEqvAndEmptyOrNextBox* **by** *fastforce*
 hence *2*: $\vdash \neg(\Box f) \longrightarrow \neg(f \wedge (\bigcirc (\Box f)))$ **by** *fastforce*
 have *3*: $\vdash (\neg(f \wedge (\bigcirc (\Box f)))) = (\neg f \vee \neg(\bigcirc (\Box f)))$ **by** *auto*
 from *2 3* **show** *?thesis* **by** *auto*
qed

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

lemma *NotBoxStateImpBoxYieldsNotBox*:

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

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

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

7.5 Properties of Da and Ba

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

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

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

lemma *DiamondNotEqvNotBox*:

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

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

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

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

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

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

lemma *BoxStateEqvBaBoxState*:
$\vdash\ \ \Box\ (init\ w) =\ \ ba\ (\Box\ (init\ w))$
proof −
have *1*: $\vdash (init\ w) = bi\ \ (init\ w)$ **by** (*rule StateEqvBi*)
hence *2*: $\vdash \Box\ (init\ w) = \Box\ (bi\ \ (init\ w))$ **by** (*rule BoxEqvBox*)

have *3*: $\vdash \Box$ (*bi* (*init w*)) = *bi*($\Box$ (*init w*)) **by** (*rule BtBiEqvBiBt*)
have *4*: $\vdash \Box$ (*init w*) = $\Box$($\Box$ (*init w*)) **by** (*rule BoxEqvBoxBox*)
hence *5*: $\vdash$ *bi*($\Box$ (*init w*)) = *bi* ($\Box$($\Box$ (*init w*))) **by** (*rule BiEqvBi*)
have *6*: $\vdash$ *ba*($\Box$ (*init w*)) = *bi*($\Box$($\Box$ (*init w*))) **by** (*rule BaEqvBiBt*)
from *2 3 5 6* **show** *?thesis* **by** *fastforce*
qed

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

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

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

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

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

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

lemma *ChopAndCommute*:
$\vdash \quad f;\ (g \wedge g1) = f;\ (g1 \wedge g)$
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*:
$\vdash\ f;\ g \wedge \Box\ h \longrightarrow f;\ (g \wedge h)$
proof −
have *1*: $\vdash \Box\ h \wedge f;\ g \longrightarrow f;\ (h \wedge g)$ **by** (*rule BoxAndChopImport*)
have *2*: $\vdash f;\ (h \wedge g) = f;\ (g \wedge h)$ **by** (*rule ChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

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

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

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

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

qed

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

lemma *NotChopEqvYieldsNot*:
$\vdash \ \ (\neg\ (f;\ g)) = f\ \mathit{yields}\ \ (\neg\ \ g)$
proof –
 have *1*: $\vdash g = (\neg\ \neg\ \ g)$ **by** *auto*
 hence *2*: $\vdash f;\ g = f;\ (\neg\ \neg\ \ g)$ **by** (*rule RightChopEqvChop*)
 hence *3*: $\vdash (\neg\ (f;\ g)) = (\neg\ (f;\ (\neg\ \neg\ \ g)))$ **by** *auto*
 from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *NotDiFalse*:
$\vdash \ \ \neg\ (\ \mathit{di}\ \ \#\mathit{False})$
proof –
 have *1*: $\vdash (\mathit{init}\ \#\mathit{True}) \longrightarrow \mathit{bi}\ (\mathit{init}\ \#\mathit{True})$ **by** (*rule StateImpBi*)
 hence *2*: $\vdash \#\mathit{True} \longrightarrow \mathit{bi}\ \#\mathit{True}$ **by** (*auto simp*: *bi-defs*)
 have *3*: $\vdash \#\mathit{True}$ **by** *auto*
 have *4*: $\vdash \mathit{bi}\ \#\mathit{True}$ **using** *2 3 MP* **by** *auto*
 hence *5*: $\vdash \neg\ (\ \mathit{di}\ (\neg\ \#\mathit{True}))$ **by** (*simp add*: *bi-d-def*)
 have *6*: $\vdash (\neg\ \#\mathit{True}) = \ \#\mathit{False}$ **by** *auto*
 hence *7*: $\vdash \ \mathit{di}\ (\neg\ \#\mathit{True}) = \ \mathit{di}\ \ \#\mathit{False}$ **by** (*rule DiEqvDi*)
 from *5 7* **show** *?thesis* **by** *auto*
qed

lemma *StateAndEmptyChop*:
$\vdash \ \ ((\mathit{init}\ w) \wedge\ \ \mathit{empty}\ );\ f = ((\mathit{init}\ w) \wedge f)$
proof –
 have *1*: $\vdash ((\mathit{init}\ w) \wedge\ \ \mathit{empty}\ );\ f = ((\mathit{init}\ w) \wedge\ \ \mathit{empty}\ ;\ f)$ **by** (*rule StateAndChop*)
 have *2*: $\vdash \ \mathit{empty}\ ;\ f = f$ **by** (*rule EmptyChop*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndNextChop*:
$\vdash \ \ ((\mathit{init}\ w) \wedge \bigcirc\ f);\ g = ((\mathit{init}\ w) \wedge \bigcirc(f;\ g))$
proof –
 have *1*: $\vdash ((\mathit{init}\ w) \wedge \bigcirc\ f);\ g = ((\mathit{init}\ w) \wedge (\bigcirc\ f);\ g)$ **by** (*rule StateAndChop*)
 have *2*: $\vdash (\bigcirc\ f);\ g = \bigcirc(f;\ g)$ **by** (*rule NextChop*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NextAndEqvNextAndNext*:
$\vdash \bigcirc\ (f \wedge g) = (\bigcirc\ f \wedge \bigcirc\ g)$

by (*auto simp*: *next-defs*)

lemma *NextStateAndChop*:
$\vdash \bigcirc(((init\ w) \wedge f); g) = (\bigcirc\ (init\ w) \wedge \bigcirc(f;\ g))$
proof −
 have *1*: $\vdash ((init\ w) \wedge f); g = ((init\ w) \wedge f;\ g)$ **by** (*rule StateAndChop*)
 hence *2*: $\vdash \bigcirc(((init\ w) \wedge f); g) = \bigcirc((init\ w) \wedge f;\ g)$ **by** (*rule NextEqvNext*)
 have *3*: $\vdash \bigcirc((init\ w) \wedge f;\ g) = (\bigcirc\ (init\ w) \wedge \bigcirc(f;\ g))$ **by** (*rule NextAndEqvNextAndNext*)
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateYieldsEqv*:
$\vdash ((init\ w) \longrightarrow (f\ yields\ \ g)) = ((init\ w) \wedge f)\ yields\ \ g$
proof −
 have *1*: $\vdash ((init\ w) \wedge f); (\neg\ \ g) = ((init\ w) \wedge f;\ (\neg\ \ g))$ **by** (*rule StateAndChop*)
 hence *2*: $\vdash ((init\ w) \longrightarrow \neg\ (f;\ (\neg\ \ g))) = (\neg\ (((init\ w) \wedge f); (\neg\ \ g)\ ))$ **by** *auto*
 from *2* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *StateAndDi*:
$\vdash ((init\ w) \wedge\ \ di\ \ f) =\ \ di\ ((init\ w) \wedge f)$
proof −
 have *1*: $\vdash ((init\ w) \wedge f); \#True = ((init\ w) \wedge f;\ \#True)$ **by** (*rule StateAndChop*)
 from *1* **show** *?thesis* **by** (*metis di-d-def inteq-reflection*)
qed

lemma *DiNext*:
$\vdash di(\ \bigcirc\ f) = \bigcirc\ (di\ \ f)$
proof −
 have *1*: $\vdash (\bigcirc\ f); \#True = \bigcirc(f;\ \#True)$ **by** (*rule NextChop*)
 from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

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

lemma *StateImpBiGen*:
 assumes $\vdash\ (init\ w) \longrightarrow f$
 shows $\vdash (init\ w) \longrightarrow bi\ \ f$
proof −
 have *1*: $\vdash (init\ w) \longrightarrow f$ **using** *assms* **by** *auto*
 hence *2*: $\vdash \neg\ \ f \longrightarrow \neg\ \ (init\ w)$ **by** *auto*
 hence *3*: $\vdash\ di\ (\neg\ \ f) \longrightarrow\ di\ (\neg\ \ (init\ w))$ **by** (*rule DiImpDi*)
 hence *4*: $\vdash\ di\ (\neg\ \ f) \longrightarrow\ di\ \ (init\ (\neg w))$ **by** (*metis Initprop*(*2*) *inteq-reflection*)

have *5*: ⊢ *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 ⊢ *f*⟶ *f1*
shows ⊢ (*f1 yields g*) ⟶ (*f yields g*)
proof −
have *1*: ⊢ *f* ⟶ *f1* **using** *assms* **by** *auto*
hence *2*: ⊢ *f*; (¬ *g*) ⟶ *f1*; (¬ *g*) **by** (*rule LeftChopImpChop*)

hence *3*: ⊢ ¬ (*f1*; (¬ *g*)) ⟶ ¬ (*f*; (¬ *g*)) **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *LeftYieldsEqvYields*:
assumes ⊢ *f* = *f1*
shows ⊢ (*f yields g*) = (*f1 yields g*)
proof −
have *1*: ⊢ *f* = *f1* **using** *assms* **by** *auto*
hence *2*: ⊢ *f*; (¬ *g*) = *f1*; (¬ *g*) **by** (*rule LeftChopEqvChop*)
hence *3*: ⊢ (¬ (*f*; (¬ *g*))) = (¬ (*f1*; (¬ *g*))) **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

7.6 Properties of Fin

lemma *FinEqvTrueChopAndEmpty*:
⊢ *fin f* = #*True*;(*f* ∧ *empty*)
proof −
have *1*: ⊢*fin f* = □(*empty* ⟶ *f*)
by (*simp add*: *fin-d-def*)
have *2*: ⊢ □(*empty* ⟶ *f*) = (¬(◇(¬(*empty* ⟶ *f*))))
by (*simp add*: *always-d-def*)
have *3*: ⊢ (¬(*empty* ⟶ *f*)) = (¬ *f* ∧ *empty*)
by *auto*
hence *4*: ⊢ ◇(¬(*empty* ⟶ *f*)) = ◇(¬ *f* ∧ *empty*)
using *DiamondEqvDiamond* **by** *blast*
hence *5*: ⊢ ¬(◇(¬(*empty* ⟶ *f*))) = (¬(◇(¬ *f* ∧ *empty*)))
by *auto*
have *6*: ⊢ (¬(◇(¬ *f* ∧ *empty*))) = #*True*;(*f* ∧ *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*:
⊢ ◇(*fin w*) = *fin w*
by (*metis DiamondDiamondEqvDiamond FinEqvTrueChopAndEmpty TrueChopEqvDiamond inteq-reflection*)

lemma *ChopFinExportA*:
⊢ *f*;(*g* ∧ *fin w*) ⟶ *fin w*
using *DiamondFin*
by (*metis ChopAndB ChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *FinImpBox*:
⊢ *fin w* ⟶ □(*fin w*)
by (*metis BoxImpBoxBox fin-d-def*)

lemma *FinAndChopImport*:
⊢ (*fin w*) ∧ (*f*;*g*) ⟶ *f*;((*fin w*) ∧ *g*)

proof −
have *1*: ⊢ *fin w* ⟶ □(*fin w*) **by** (*rule FinImpBox*)
hence *2*: ⊢ *fin w* ∧ *f*;*g* ⟶ □(*fin w*) ∧ (*f*;*g*) **by** *auto*
have *3*: ⊢ □(*fin w*) ∧ (*f*;*g*) ⟶ *f*;((*fin w*) ∧ *g*) **using** *BoxAndChopImport* **by** *blast*
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *FinAndChop*:
⊢ (*f*;(*g* ∧ *fin w*)) = (*fin w* ∧ *f*;*g*)
using *FinAndChopImport ChopFinExportA ChopAndA ChopAndCommute* **by** *fastforce*

lemma *ChopAndEmptyEqvEmptyChopEmpty*:
⊢ ((*f*;*g*) ∧ *empty*) = (*f* ∧ *empty*);(*g* ∧ *empty*)
by (*auto simp*: *empty-defs chop-defs*)

lemma *FinAndEmpty*:
⊢ ((*fin w*) ∧ *empty*) = (*w* ∧ *empty*)
proof −
have *1*: ⊢ ((*fin w*) ∧ *empty*) = (#*True*;(*w* ∧ *empty*) ∧ *empty*)
using *FinEqvTrueChopAndEmpty* **by** *fastforce*
have *2*: ⊢ (#*True*;(*w* ∧ *empty*) ∧ *empty*) = ((#*True* ∧ *empty*);(*w* ∧ *empty*))
using *ChopAndEmptyEqvEmptyChopEmpty*[*of LIFT*(#*True*) *LIFT*(*w* ∧ *empty*)]
by (*metis AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty Prop11 Prop12 int-eq*)
have *3*: ⊢ (#*True* ∧ *empty*);(*w* ∧ *empty*) = (*empty*;(*w* ∧ *empty*))
using *LeftChopEqvChop* **by** *fastforce*
have *4*: ⊢ (*empty*;(*w* ∧ *empty*)) = (*w* ∧ *empty*)
using *EmptyChop* **by** *blast*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

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

lemma *AndFinEqvChopStateAndEmpty*:
⊢ (*f* ∧ *fin* (*init w*)) = *f*; ((*init w*) ∧ *empty*)

using *AndFinEqvChopAndEmpty* **by** *blast*

lemma *FinStateEqvStateAndEmptyOrNextFinState*:
$\vdash$ *fin* (*init w*) = (((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*fin* (*init w*)))
proof −
have *1*: $\vdash$ *fin* (*init w*) = $\square$(*empty* $\longrightarrow$ *init w*)
by (*simp add*: *fin-d-def*)
have *2* : $\vdash$ $\square$(*empty* $\longrightarrow$ *init w*) =
((*empty* $\longrightarrow$ *init w*) $\wedge$ *wnext* ($\square$ (*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*:
$\vdash$ $\neg$($\diamond$ (*f* $\wedge$ $\neg$ *f*))
proof −
have *1*: $\vdash$ ($\neg$($\diamond$ (*f* $\wedge$ $\neg$ *f*))) = $\Box$($\neg$(*f* $\wedge$ $\neg$*f*)) **using** *NotDiamondNotEqvBox* **by** *fastforce*
have *2*: $\vdash$ $\neg$(*f* $\wedge$ $\neg$*f*) **by** *simp*
have *3*: $\vdash$ $\Box$($\neg$(*f* $\wedge$ $\neg$*f*)) **using** *2* **by** (*simp add*: *BoxGen*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FinYields*:
$\vdash$ (*fin* (*init w*)) *yields* (*init w*)
proof −
have *1*: $\vdash$ (*fin* (*init w*)); ($\neg$(*init w*)) = $\diamond$((*init w*) $\wedge$ $\neg$(*init w*)) **by** (*rule FinChopEqvDiamond*)
have *2*: $\vdash$ $\neg$($\diamond$((*init w*) $\wedge$ $\neg$ (*init w*))) **by** (*rule NotDiamondAndNot*)
have *3*: $\vdash$ $\neg$ ((*fin* (*init w*)); ($\neg$ (*init w*))) **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ImpAndFinStateOrFinNotState*:
$\vdash$ *f* $\longrightarrow$ (*f* $\wedge$ *fin* (*init w*)) $\vee$ *fin* ($\neg$ (*init w*))
by (*simp add*: *fin-defs Valid-def*)

lemma *AndFinChopEqvStateAndChop*:
$\vdash$ (*f* $\wedge$ *fin* (*init w*)); *g* = *f*; ((*init w*) $\wedge$ *g*)
proof −
have *1*: $\vdash$ (*fin* (*init w*)) *yields* (*init w*)
  by (*rule FinYields*)
have *2*: $\vdash$ *f* $\wedge$ *fin* (*init w*) $\longrightarrow$ *fin* (*init w*)
  by *auto*
hence *3*: $\vdash$ (*fin* (*init w*)) *yields* (*init w*) $\longrightarrow$ (*f* $\wedge$ *fin* (*init w*)) *yields* (*init w*)
  by (*rule LeftYieldsImpYields*)
have *4*: $\vdash$ (*f* $\wedge$ *fin* (*init w*)) *yields* (*init w*)
  using *1 3 MP* **by** *fastforce*
have *5*: $\vdash$ (*f* $\wedge$ *fin* (*init w*)); *g* $\wedge$ (*f* $\wedge$ *fin* (*init w*)) *yields* (*init w*)
    $\longrightarrow$ (*f* $\wedge$ *fin* (*init w*)); (*g* $\wedge$ (*init w*))

by (*rule ChopAndYieldsImp*)
have *6*: $\vdash (f \wedge\ fin\ (init\ w)); g \longrightarrow (f \wedge\ fin\ (init\ w)); (g \wedge (init\ w))$
using *4 5* **by** *fastforce*
have *7*: $\vdash (f \wedge\ fin\ (init\ w)); (g \wedge (init\ w)) \longrightarrow f; (g \wedge (init\ w))$
by (*rule AndChopA*)
have *8*: $\vdash g \wedge (init\ w) \longrightarrow (init\ w) \wedge g$
by *auto*
hence *9*: $\vdash f; (g \wedge (init\ w)) \longrightarrow f; ((init\ w) \wedge g)$
by (*rule RightChopImpChop*)
have *10*: $\vdash (f \wedge\ fin\ (init\ w)); g \longrightarrow f; ((init\ w) \wedge g)$
using *6 7 9* **by** *fastforce*
have *11*: $\vdash f \longrightarrow (f \wedge\ fin\ (init\ w)) \vee\ fin\ (\neg\ (init\ w))$
by (*rule ImpAndFinStateOrFinNotState*)
hence *12*: $\vdash f; ((init\ w) \wedge g) \longrightarrow$
$((f \wedge\ fin\ (init\ w)) \vee\ fin\ (\neg\ (init\ w))); ((init\ w) \wedge g)$
by (*rule LeftChopImpChop*)
have *13*: $\vdash ((f \wedge\ fin\ (init\ w)) \vee\ fin\ (\neg\ (init\ w))); ((init\ w) \wedge g)$
$=$
$((f \wedge\ fin\ (init\ w)); ((init\ w) \wedge g) \vee\ (\ fin\ (\neg\ (init\ w))); ((init\ w) \wedge g))$
by (*rule OrChopEqv*)
have *14*: $\vdash (\ fin\ \ (init\ (\neg\ w))); ((init\ w) \wedge g) \longrightarrow \diamond(\ (init\ (\neg\ w)) \wedge ((init\ w) \wedge g))$
using *FinChopEqvDiamond* **by** *fastforce*
have *141*: $\vdash \neg(\ \diamond(\ (init\ (\neg\ w)) \wedge ((init\ w) \wedge g))) \longrightarrow$
$\neg\ (\ (\ fin\ \ (init\ (\neg\ w))); ((init\ w) \wedge g))$
using *14* **by** *fastforce*
have *15*: $\vdash \neg(\ \diamond(\ (init\ (\neg\ w)) \wedge ((init\ w) \wedge g)))$
using *NotDiamondAndNot Initprop*(*2*) **by** (*auto simp*: *sometimes-defs init-defs*)
have *151*: $\vdash \neg\ (\ (\ fin\ \ (init\ (\neg\ w))); ((init\ w) \wedge g))$
using *15 141* **by** *fastforce*
have *1511*: $\vdash (\ fin\ (\neg\ (init\ w))); ((init\ w) \wedge g) \longrightarrow \#False$
using *151* **by** (*metis Initprop*(*2*) *int-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*: $\vdash (f \wedge fin(init\ w)); \#True = f;((init\ w) \wedge \#True)$ **by** (*rule AndFinChopEqvStateAndChop*)
have *2*: $\vdash ((init\ w) \wedge \#True) = (init\ w)$ **by** *auto*
hence *3*: $\vdash (f; ((init\ w) \wedge \#True)) = (f; (init\ w))$ **by** (*rule RightChopEqvChop*)
have *4*: $\vdash (f \wedge\ fin\ (init\ w)); \#True = f; (init\ w)$ **using** *1 3* **by** *auto*

from 4 show ?thesis by (simp add: di-d-def)
qed

lemma FinNotStateEqvNotFinState:
⊢ 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: ⊢ (init w) ∧ f ⟶ f1 using assms by auto
hence 2: ⊢ (init w) ∧ (f; (¬ g)) ⟶ f1; (¬ g) by (rule StateAndChopImpChopRule)
hence 3: ⊢ (init w) ∧ ¬ (f1; (¬ g)) ⟶ ¬ (f; (¬ 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*: $\vdash$ *skip* ; *#True* $= \bigcirc$*#True* **by** (*rule SkipChopEqvNext*)
hence *2*: $\vdash \bigcirc$*#True* $=$ *skip* ; *#True* **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *more-d-def*)
qed

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

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

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

lemma *WeakNextBoxImpMoreYields*:
$\vdash$ *more* *yields* $f =$ *wnext*($\Box\ f$)
proof $-$
have *1*: $\vdash$ *more* ; $(\neg\ f) = \bigcirc(\Diamond\ (\neg f))$ **by** (*rule MoreChopEqvNextDiamond*)
have *2*: $\vdash \bigcirc(\Diamond\ (\neg f)) = \bigcirc(\neg(\Box f))$ **by** (*auto simp*: *always-d-def*)
have *3*: $\vdash \bigcirc(\neg(\Box f)) = (\neg$ (*wnext*($\Box\ f$))) **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash$ *more* ; $(\neg\ f) = (\neg($*more* *yields* f)) **by** (*simp add*: *yields-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

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

lemma *LeftChopImpMoreRule*:

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

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

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

lemma *ChopImpDi*:
$\vdash$ *f*; *g* $\longrightarrow$ *di* *f*
proof –
have *1*: $\vdash$ *g* $\longrightarrow$ #*True* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ *f*; #*True* **by** (*rule RightChopImpChop*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *TrueEqvTrueChopTrue*:
$\vdash$ #*True* = #*True*; #*True*
proof –
have *1*: $\vdash$ #*True*; #*True* $\longrightarrow$ #*True* **by** *auto*
have *2*: $\vdash$ #*True* $\longrightarrow$ *di* #*True* **by** (*rule DiIntro*)
hence *3*: $\vdash$ #*True* $\longrightarrow$ #*True*; #*True* **by** (*simp add*: *di-d-def*)
from *1 3* **show** *?thesis* **by** *auto*
qed

lemma *DiEqvDiDi*:
$\vdash$ *di* *f* = *di* (*di* *f*)
proof –
have *1*: $\vdash$ #*True* = #*True*; #*True* **by** (*rule TrueEqvTrueChopTrue*)
hence *2*: $\vdash$ *f*; #*True* = *f*; (#*True*; #*True*) **by** (*rule RightChopEqvChop*)

have *3*: ⊢ *f*; (#*True*; #*True*)= (*f*; #*True*); #*True* **by** (*rule ChopAssoc*)
have *4*: ⊢ *f*; #*True* = (*f*; #*True*); #*True* **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis di-d-def*)
qed

lemma *BiEqvBiBi*:
⊢ *bi f* = *bi*(*bi f*)
proof −
have *1*: ⊢ *di* (¬ *f*) = *di*(*di* (¬ *f*)) **by** (*rule DiEqvDiDi*)
have *2*: ⊢ *di* (¬ *f*) = (¬ (*bi f*)) **by** (*rule DiNotEqvNotBi*)
hence *3*: ⊢ *di* (*di* (¬ *f*)) = *di* (¬ (*bi f*)) **by** (*rule DiEqvDi*)
have *4*: ⊢ *di* (¬ *f*) = *di* (¬(*bi f*)) **using** *1 3* **by** *fastforce*
hence *5*: ⊢ (¬ (*di* (¬ *f*))) = (¬ (*di* (¬ (*bi f*)))) **by** *fastforce*
from *5* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *DiOrEqv*:
⊢ *di* (*f* ∨ *g*) = (*di f* ∨ *di g*)
proof −
have *1*: ⊢ (*f*∨ *g*); #*True* = (*f*; #*True* ∨ *g*; #*True*) **by** (*rule OrChopEqv*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

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

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

lemma *DiAndImpAnd*:
⊢ *di* (*f* ∧ *g*) ⟶ *di f* ∧ *di g*
proof −
have *1*: ⊢ *di* (*f* ∧ *g*) ⟶ *di f* **by** (*rule DiAndA*)
have *2*: ⊢ *di* (*f* ∧ *g*) ⟶ *di g* **by** (*rule DiAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DiSkipEqvMore*:
⊢ *di skip* = *more*
proof −
have *1*: ⊢ *skip* ; #*True* = ○#*True* **by** (*rule SkipChopEqvNext*)
have *2*: ⊢ ○#*True* = *more* **by** (*auto simp*: *more-d-def*)

have *3*: ⊢ *skip* ; #*True* = *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiMoreEqvMore*:
⊢ *di* *more* = *more*
proof −
have *1*: ⊢ *di* (○ #*True*) = ○(*di* #*True*) **by** (*rule DiNext*)
have *2*: ⊢ ○(*di* #*True*) ⟶ *more* **by** (*auto simp*: *next-defs di-defs more-defs*)
have *3*: ⊢ *di*(○ #*True*) ⟶ *more* **using** *1 2* **by** *fastforce*
hence *4*: ⊢ *di* *more* ⟶ *more* **by** (*simp add*: *more-d-def*)
have *5*: ⊢ *more* ⟶ *di* *more* **by** (*rule ImpDi*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *DiIfEqvRule*:
assumes ⊢ *f* = *if*$_i$ (*init w*) *then* *g* *else* *h*
shows ⊢ *di* *f* = *if*$_i$ (*init w*) *then* (*di* *g*) *else* (*di* *h*)
proof −
have *1*: ⊢ *f* = *if*$_i$ (*init w*) *then* *g* *else* *h* **using** *assms* **by** *auto*
hence *2*: ⊢ *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*:
⊢ *di* *empty*
proof −
have *1*: ⊢ #*True* **by** *auto*
have *2*: ⊢ *empty* ; #*True* = #*True* **by** (*rule EmptyChop*)
have *3*: ⊢ *empty* ; #*True* **using** *1 2* **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

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

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

lemma *DaEqvNotBaNot*:
⊢ *da* *f* = (¬ (*ba* (¬ *f*)))
proof −
have *1*: ⊢ *ba* (¬ *f*) = (¬ (*da* (¬ ¬ *f*))) **by** (*simp add*: *ba-d-def*)

hence *2*: $\vdash$ *da* $(\neg\neg\ f) = (\neg(\ $*ba*$\ (\neg\ f)))$ **by** *fastforce*
have *3*: $\vdash f = (\neg\neg\ f)$ **by** *simp*
hence *4*: $\vdash$ *da* $f =$ *da* $(\neg\neg\ f)$ **by** (*rule DaEqvDa*)
from *2 4* **show** *?thesis* **by** *simp*
qed

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

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

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

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

have *6*: $\vdash$ *ba* *f* = □(*bi* *f*) **by** (*rule BaEqvBtBi*)
have *7*: $\vdash$ *ba* *g* = □(*bi* *g*) **by** (*rule BaEqvBtBi*)
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndEqv*:
$\vdash$ *ba* (*f* ∧ *g*) = (*ba* *f* ∧ *ba* *g*)
proof −
have *1*: $\vdash$ *ba* (*f* ∧ *g*) = □(*bi* (*f* ∧ *g*))
by (*rule BaEqvBtBi*)
have *2*: $\vdash$ *bi* (*f* ∧ *g*) = (*bi f* ∧ *bi g*)
by (*auto simp*: *bi-defs*)
hence *3*: $\vdash$ □(*bi* (*f* ∧ *g*)) = □(*bi f* ∧ *bi g*)
using *BoxEqvBox* **by** *blast*
have *4*: $\vdash$ □(*bi f* ∧ *bi g*)= (□(*bi f*) ∧ □(*bi g*))
by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
have *5*: $\vdash$*ba* *f* = □(*bi* *f*)
by (*rule BaEqvBtBi*)
have *6*: $\vdash$ *ba* *g* = □(*bi* *g*)
by (*rule BaEqvBtBi*)
from *1 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBaEqvBa*:
$\vdash$ *ba* (*f* = *g*) ⟶ (*ba* *f* = *ba* *g*)
proof −
have *1*: $\vdash$ *ba* (*f* ⟶ *g*) ⟶ *ba* *f* ⟶ *ba* *g* **by** (*rule BaImpDist*)
have *2*: $\vdash$ *ba* (*g* ⟶ *f*) ⟶ *ba* *g* ⟶ *ba* *f* **by** (*rule BaImpDist*)
have *3*: $\vdash$ *ba* (*f* = *g*) = *ba* ((*f* ⟶ *g*) ∧ (*g* ⟶ *f*)) **by** (*auto simp*: *ba-defs*)
have *4*: $\vdash$ *ba* ((*f* ⟶ *g*) ∧ (*g* ⟶ *f*)) = (*ba*((*f* ⟶ *g*)) ∧ *ba*((*g* ⟶ *f*))) **by** (*rule BaAndEqv*)
have *5*: $\vdash$ ((*ba* *f* ⟶ *ba* *g*) ∧ (*ba* *g* ⟶ *ba* *f*)) = (*ba* *f* = *ba* *g*) **by** *auto*
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

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

lemma *DiamondEqvDiamondDiamond*:

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

lemma *DaEqvDaDa*:
⊢ *da f* = *da* (*da f*)
proof −
have *1*: ⊢ *da f* = ◇(*di f*)
  by (*rule DaEqvDtDi*)
have *2*: ⊢ *di f* = (*di* (*di f*))
  by (*rule DiEqvDiDi*)
hence *3*: ⊢ ◇ (*di f*) = ◇ (*di* (*di f*))
  by (*rule DiamondEqvDiamond*)
have *4*: ⊢ ◇ (*di f*) = ◇(◇ (*di* (*di f*)))
  using *DiamondEqvDiamondDiamond DiEqvDiDi* **using** *3* **by** *fastforce*
have *5*: ⊢ ◇ (*di* (*di f*)) = *di* (◇ (*di f*))
  by (*rule DtDiEqvDiDt*)
hence *6*: ⊢ ◇(◇ (*di* (*di f*))) = ◇ (*di* (◇ (*di f*)))
  by (*rule DiamondEqvDiamond*)
have *7*: ⊢ *da f* = ◇ (*di*(◇ (*di f*)))
  using *1 3 4 6* **by** *fastforce*
have *8*: ⊢ *da* (◇ (*di f*)) = ◇(*di* (◇ (*di f*)))
  by (*rule DaEqvDtDi*)
have *9*: ⊢ *da* (*da f*) = *da* (◇ (*di f*))
  using *1* **by** (*rule DaEqvDa*)
from *7 8 9* **show** *?thesis* **by** *fastforce*
qed

lemma *BaEqvBaBa*:
⊢ *ba f* = *ba* (*ba f*)
proof −
have *1*: ⊢ *da* (¬ *f*) = *da* (*da* (¬ *f*)) **by** (*rule DaEqvDaDa*)
have *2*: ⊢ *da* (*da* (¬ *f*)) = (¬ (*ba* (¬ (*da* (¬ *f*))))) **by** (*rule DaEqvNotBaNot*)
have *3*: ⊢ (¬ (*da* (*da* (¬ *f*)))) = *ba* (¬ (*da* (¬ *f*))) **by** (*auto simp*: *ba-d-def*)
have *4*: ⊢ (¬ (*da* (¬ *f*))) = *ba* (¬ (*da* (¬ *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*:
$\vdash$ $(f;\ f1) \land\ ba\ \ g \longrightarrow (f \land g);\ (f1 \land g)$
proof –
have *1*: $\vdash$ $ba\ \ g \land (f;\ f1) \longrightarrow (g \land f);\ (g \land f1)$ **by** (*rule BaAndChopImport*)
have *2*: $\vdash$ $(g \land f);\ (g \land f1) = (f \land g);\ (f1 \land g)$ **by** (*rule AndChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

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

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

lemma *DiNotBaImpNotBa*:
$\vdash$ $di\ (\neg\ (\ ba\ \ f)) \longrightarrow \neg\ (\ ba\ \ f)$
proof –
have *1*: $\vdash$ $ba\ \ f =\ ba(\ \ ba\ \ f)$ **by** (*rule BaEqvBaBa*)
have *2*: $\vdash$ $ba\ (\ ba\ \ f) \longrightarrow bi\ (\ ba\ \ f)$ **by** (*rule BaImpBi*)
have *3*: $\vdash$ $ba\ \ f \longrightarrow bi\ (\ ba\ \ f)$ **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ $ba\ \ f \longrightarrow \neg\ (\ di\ (\neg\ (\ ba\ \ f)))$ **by** (*simp add*: *bi-d-def*)
from *4* **show** *?thesis* **by** *fastforce*

qed

lemma *NotBaChopImpNotBa*:
$\vdash$ $(\neg\ (\ ba\ \ f));\ g \longrightarrow \neg\ (\ ba\ \ f)$
proof –
have *1*: $\vdash (\neg\ (\ ba\ \ f));\ g \longrightarrow\ \ di\ (\neg\ (\ ba\ \ f))$ **by** (*rule ChopImpDi*)
have *2*: $\vdash\ \ di\ (\neg\ (\ ba\ \ f)) \longrightarrow \neg\ (\ ba\ \ f)$ **by** (*rule DiNotBaImpNotBa*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *DiamondFinImpFin*:
$\vdash \Diamond\ (fin\ f) \longrightarrow fin\ f$
proof –
have *1*: $\vdash fin\ f = \#True;(f \wedge empty)$
by (*rule FinEqvTrueChopAndEmpty*)
hence *2*: $\vdash \Diamond\ (fin\ f) = \#True;(\#True;(f \wedge empty))$
by (*metis ChopEqvChop TrueEqvTrueChopTrue inteq-reflection sometimes-d-def*)
have *3*: $\vdash \#True;(\#True;(f \wedge empty)) = (\#True;\#True);(f \wedge empty)$
by (*rule ChopAssoc*)
have *4*: $\vdash (\#True;\#True);(f \wedge empty) = \#True;(f \wedge empty)$
using *TrueEqvTrueChopTrue* **using** *LeftChopEqvChop* **by** (*metis int-eq*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopFinImpFin*:
$\vdash\ \ f;\ \ fin\ \ (init\ w) \longrightarrow\ \ fin\ \ (init\ w)$
proof –
have *1*: $\vdash f;\ \ fin\ \ (init\ w) \longrightarrow \Diamond\ (\ fin\ \ (init\ w))$ **by** (*rule ChopImpDiamond*)
have *2*: $\vdash \Diamond\ \ (fin\ \ (init\ w)) \longrightarrow\ \ fin\ \ (init\ w)$ **by** (*rule DiamondFinImpFin*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

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

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

lemma *ChopAndNotFin*:
 $\vdash$ $(f; g \wedge \neg$ (*fin* $(init\ w))) = f; (g \wedge \neg$ (*fin* $(init\ w)))$
proof −
 have *1*: $\vdash (f; g \wedge$ *fin* $(init\ (\neg\ w))) = f; (g \wedge$ *fin* $(init\ (\neg\ w)))$
  by (*rule ChopAndFin*)
 have *2*: $\vdash$ *fin* $(init\ (\neg\ w)) = (\neg$ (*fin* $(init\ w)\ ))$
  using *FinNotStateEqvNotFinState* **by** *blast*
 hence *3*: $\vdash (g \wedge$ *fin* $(init\ (\neg\ w))) = (g \wedge \neg($ *fin* $(init\ w)))$
  by *auto*
 hence *4*: $\vdash f; (g \wedge$ *fin* $(init\ (\neg\ w))) = f; (g \wedge \neg$ (*fin* $(init\ w)))$
  by (*rule RightChopEqvChop*)
 from *1 2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *FinChopChain*:
 $\vdash$ $((init\ w) \longrightarrow$ *fin* $(init\ w1)); ((init\ w1) \longrightarrow$ *fin* $(init\ w2))$
  $\longrightarrow ((init\ w) \longrightarrow$ *fin* $(init\ w2))$
proof −
 have *1*: $\vdash$ $(init\ w) \wedge ((init\ w) \longrightarrow$ *fin* $(init\ w1)); ((init\ w1) \longrightarrow$ *fin* $(init\ w2))$
   $\longrightarrow$
   $((init\ w) \wedge ((init\ w) \longrightarrow$ *fin* $(init\ w1))); ((init\ w1) \longrightarrow$ *fin* $(init\ w2))$
  by (*rule StateAndChopImport*)
 have *2*: $\vdash (init\ w) \wedge ((init\ w) \longrightarrow$ *fin* $(init\ w1)) \longrightarrow$ *fin* $(init\ w1)$
  by *auto*
 have *3*: $\vdash ((init\ w) \wedge ((init\ w) \longrightarrow$ *fin* $(init\ w1))); ((init\ w1) \longrightarrow$ *fin* $(init\ w2))$
   $\longrightarrow$

(*fin* (*init w1*)); ((*init w1*) $\longrightarrow$ *fin* (*init w2*))
using *2* **by** (*rule LeftChopImpChop*)
have *4*: $\vdash$ (*fin* (*init w1*)); ((*init w1*) $\longrightarrow$ *fin* (*init w2*)) =
$\Diamond$((*init w1*) $\wedge$ ((*init w1*) $\longrightarrow$ *fin* (*init w2*)))
by (*rule FinChopEqvDiamond*)
have *41*: $\vdash$ ((*init w1*) $\wedge$ ((*init w1*) $\longrightarrow$ *fin* (*init w2*))) $\longrightarrow$ *fin* (*init w2*)
by *auto*
have *42*: $\vdash$ $\Diamond$((*init w1*) $\wedge$ ((*init w1*) $\longrightarrow$ *fin* (*init w2*))) $\longrightarrow$ $\Diamond$ (*fin* (*init w2*))
using *41 DiamondImpDiamond* **by** *blast*
have *5*: $\vdash$ $\Diamond$(*fin*(*init w2*)) $\longrightarrow$ *fin* (*init w2*)
using *DiamondFinImpFin* **by** *blast*
have *6*: $\vdash$ (*init w*) $\wedge$ ((*init w*)$\longrightarrow$ *fin* (*init w1*)); ((*init w1*) $\longrightarrow$ *fin* (*init w2*))
$\longrightarrow$ *fin* (*init w2*)
using *1 3 4 5 42* **by** *fastforce*
from *6* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopRule*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *fin* (*init w1*)
$\vdash$ (*init w1*)$\wedge$ *f1* $\longrightarrow$ *fin* (*init w2*)
shows $\vdash$ (*init w*) $\wedge$ (*f*; *f1*) $\longrightarrow$ *fin* (*init w2*)
proof −
have *1*: $\vdash$ (*init w*) $\wedge$ (*f*; *f1*) $\longrightarrow$ ((*init w*) $\wedge$ *f*); *f1* **by** (*rule StateAndChopImport*)
have *2*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *fin* (*init w1*) **using** *assms* **by** *auto*
hence *3*: $\vdash$ ((*init w*) $\wedge$ *f*); *f1* $\longrightarrow$ (*fin* (*init w1*)); *f1* **by** (*rule LeftChopImpChop*)
have *4*: $\vdash$ (*fin* (*init w1*)); *f1* = $\Diamond$((*init w1*) $\wedge$ *f1*) **by** (*rule FinChopEqvDiamond*)
have *5*: $\vdash$ (*init w1*) $\wedge$ *f1* $\longrightarrow$ *fin* (*init w2*) **using** *assms* **by** *auto*
hence *6*: $\vdash$ $\Diamond$((*init w1*) $\wedge$ *f1*) $\longrightarrow$ $\Diamond$ (*fin* (*init w2*)) **by** (*rule DiamondImpDiamond*)
have *7*: $\vdash$ $\Diamond$(*fin* (*init w2*)) $\longrightarrow$ *fin* (*init w2*) **using** *DiamondFinImpFin* **by** *blast*
from *1 3 4 6 7* **show** *?thesis* **by** *fastforce*
qed

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

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

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

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

lemma *ChopLoop*:

assumes $\vdash$ *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

7.7 Properties of Chopstar and Chopplus

lemma *Chopstardef*:
$\vdash$ *chopstar f* = *powerstar* (*f* $\wedge$ *more*)
by (*simp add*: *chopstar-d-def*)

lemma *AndEmptyChopAndEmptyEqvAndEmpty*:
$\vdash$ (*f* $\wedge$ *empty*);(*f* $\wedge$ *empty*) = (*f* $\wedge$ *empty*)
by (*auto simp add*: *Valid-def empty-defs chop-defs sum.case-eq-if*) (*metis interval-st-intlen*)

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

lemma *ChopInductL*:
assumes $\vdash$ *g* $\vee$ *f*;*h* $\longrightarrow$ *h*
shows $\vdash$ (*power f n*);*g* $\longrightarrow$ *h*

proof
(*induct n*)
case *0*
then show *?case* **using** *EmptyChop assms*
by (*metis MP Prop12 int-eq int-iffD1 int-simps*(*33*) *pow-0*)
next
case (*Suc n*)
then show *?case* **using** *assms*
by (*metis ChopAndA ChopAssoc Prop03 Prop10 Prop12 inteq-reflection lift-and-com pow-Suc*)
qed

lemma *ChopInductMoreL*:
assumes $\vdash g \vee ((f \wedge more));h \longrightarrow h$
shows $\vdash (power\ f\ n);g \longrightarrow h$
proof
(*induct n*)
case *0*
then show *?case* **using** *assms* **by** (*metis ChopInductL pow-0*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: $\vdash$ *power f* (*Suc n*);*g* = (*f*;*power f n*);*g*
by *simp*
have *2*: $\vdash$ (*f*;*power f n*);*g* = *f*;((*power f n*);*g*)
by (*meson ChopAssoc Prop11*)
have *3*: $\vdash f;((power\ f\ n);g) \longrightarrow f;h$
by (*simp add*: *RightChopImpChop Suc.hyps*)
have *4*: $\vdash f;h = ((\ (f \wedge more));\ h \vee ((f \wedge empty));h)$
by (*auto simp add*: *Valid-def chop-defs more-defs empty-defs sum.case-eq-if*) *blast*
have *5*: $\vdash (\ (f \wedge more));\ h \longrightarrow h$ **using** *assms* **by** *auto*
have *6*: $\vdash (\ (f \wedge empty));\ h \longrightarrow h$
by (*meson AndChopB EmptyChop Prop11 lift-imp-trans*)
from *5 6 4 3 2 1* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *ChopInductR*:
assumes $\vdash g \vee h;f \longrightarrow h$
shows $\vdash g;(power\ f\ n) \longrightarrow h$
proof
(*induct n*)
case *0*
then show *?case* **using** *ChopEmpty assms*
by (*metis MP Prop12 int-iffD2 int-simps*(*33*) *inteq-reflection pow-0*)
next
case (*Suc n*)
then show *?case* **using** *assms*
by (*metis AndChopA ChopAssoc PowerCommute Prop03 Prop10 Prop12 inteq-reflection lift-and-com pow-Suc*)

qed

lemma *ChopExistPower*:
$\vdash (g;(\exists\, n.\ power\ f\ n)) = (\exists\ \ n.\ g;power\ f\ n)$
using *ChopExist* **by** *fastforce*

lemma *ExistChopPower*:
$\vdash (\exists\, n.\ (power\ f\ n);g) = (\exists\, n.\ power\ f\ n);g$
using *ExistChop* **by** *fastforce*

lemma *PowerStarCommute*:
$\vdash f;(\exists\ \ n.\ power\ f\ n) = (\exists\ \ n.\ power\ f\ n);f$
proof −
 have *1*: $\vdash f\ ;(\exists\ \ n.\ power\ f\ n) =$
 $(\exists\ \ n.\ f\ ;power\ f\ n)$
 using *ChopExistPower* **by** *blast*
 have *2*: $\vdash (\exists\ \ n.\ f\ ;power\ f\ n) =$
 $(\exists\ \ n.\ (power\ f\ n);f\ )$
 using *PowerCommute* **by** *fastforce*
 have *3*: $\vdash (\exists\ \ n.\ (power\ f\ n);f) =$
 $(\exists\ \ n.\ (power\ f\ n));f$
 using *ExistChopPower* **by** *blast*
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *PowerSucAndEmptyEqvAndEmpty*:
$\vdash (power\ (f\ \wedge\ empty)\ (Suc\ n)) = (f\ \wedge\ empty)$
proof
 (*induct n*)
 case *0*
 then show *?case* **using** *ChopEmpty*
 by (*metis pow-0 pow-Suc*)
 next
 case (*Suc n*)
 then show *?case*
 by (*metis AndEmptyChopAndEmptyEqvAndEmpty inteq-reflection pow-Suc*)
qed

lemma *PowerOr*:
$\vdash\ \ (power\ (f\ \vee\ g)\ (Suc\ n)) = (\ (f;power\ (f\ \vee\ g)\ n)\ \vee$
 $(g;power\ (f\ \vee\ g)\ n))$
 by (*simp add*: *OrChopEqvRule*)

lemma *PowerEmptyOrMore*:
$\vdash (power\ (\ (f\ \wedge\ empty) \vee (f\ \wedge\ more))\ (Suc\ n)) =$
 $((f\ \wedge\ empty);(power\ (\ (f\ \wedge\ empty) \vee (f\ \wedge\ more))\ n)\ \vee$
 $(f\ \wedge\ more\ );(power\ (\ (f\ \wedge\ empty) \vee (f\ \wedge\ more))\ n))$
using *PowerOr* **by** *auto*

lemma *PSEqvEmptyOrChopPS*:

$\vdash$ *powerstar f* $=$ (*empty* $\vee$ *f*;*powerstar f*)
using *PowerstarEqvSem Valid-def* **by** *blast*

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

lemma *CSEqvOrChopCS*:
$\vdash$ $f^\star$ $=$ (*empty* $\vee$ (f; $f^\star$))
proof −
have *1*: $\vdash$ $f^\star$ $=$ (*empty* $\vee$ (f $\wedge$ *more*);$f^\star$) **by** (*rule ChopstarEqv*)
have *2*: $\vdash$ (f $\wedge$ *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*: $\vdash$ *empty* $\longrightarrow$ $f^\star$ **by** (*rule EmptyImpCS*)
have *5*: $\vdash$ f $\longrightarrow$ *empty* $\vee$ (f $\wedge$ *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*: $\vdash$ $f^\star$ $\longrightarrow$ *empty* $\vee$ (f $\wedge$ *more*);$f^\star$ **using** *1* **by** *fastforce*
have *8*: $\vdash$ f; $f^\star$ $\longrightarrow$ *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 *PowerChopCommute*:
$\vdash$ ((f $\wedge$ *more*));*power* (f $\wedge$ *more*) n $=$ *power* (f $\wedge$ *more*) n;((f $\wedge$ *more*))
using *PowerCommute* **by** *auto*

lemma *ChopExist*:
$\vdash$ (g;($\exists$ n. *power* (f $\wedge$ *more*) n)) $=$ ($\exists$ n. g;*power* (f $\wedge$ *more*) n)
using *ChopExistPower* **by** *auto*

lemma *ExistChop*:
$\vdash$ ($\exists$ n. (*power* (f $\wedge$ *more*) n);g) $=$ ($\exists$ n. *power* (f $\wedge$ *more*) n);g
using *ExistChopPower* **by** *auto*

lemma *PowerstarInductL*:
assumes $\vdash$ g $\vee$ f;h $\longrightarrow$ h
shows $\vdash$ (*powerstar f*);g $\longrightarrow$ h
proof −
have *1*: $\vdash$ (*powerstar f*);g $=$ ($\exists$ n. *power f n*);g
by (*simp add*: *powerstar-d-def LeftChopEqvChop*)
have *2*: $\vdash$ ($\exists$ n. *power f n*);g $=$
($\exists$ n. (*power f n*);g)
using *ExistChopPower* **by** *fastforce*
have *3*: $\bigwedge$ n. $\vdash$ (*power f n*);g $\longrightarrow$ h

```
 using ChopInductL assms by blast
 have 4: ⊢ (∃ n. ((power f n));g) ⟶ h
 using 3 by (auto simp add: Valid-def)
 from 1 2 4 show ?thesis by (metis inteq-reflection)
qed

lemma PowerstarInductMoreL:
 assumes ⊢ g ∨ ((f ∧ more));h ⟶ h
 shows  ⊢ (powerstar f);g ⟶ h
proof −
 have 1: ⊢ (powerstar f);g = (∃ n. power f n);g
 by (simp add: powerstar-d-def LeftChopEqvChop)
 have 2: ⊢ (∃ n. power f n);g =
          (∃ n. (power f n);g)
 using ExistChopPower by fastforce
 have 3: ⋀ n. ⊢ (power f n);g ⟶ h
 using  ChopInductMoreL assms by blast
 have 4: ⊢ (∃ n. ((power f n));g) ⟶ h
 using 3  by (auto simp add: Valid-def)
 from 1 2 4 show ?thesis by (metis inteq-reflection)
qed

lemma ChopstarInductL:
 assumes ⊢ g ∨ f;h ⟶ h
 shows  ⊢ (chopstar f);g ⟶ h
proof −
 have 1: ⊢ (chopstar f);g = ((∃ n. power (f ∧ more) n));g
 by (simp add: chopstar-d-def powerstar-d-def LeftChopEqvChop)
 have 2: ⊢ (∃ n. power (f ∧ more) n);g =
          (∃ n. (power (f ∧ more) n);g)
 using ExistChopPower by fastforce
 have 21: ⊢ g ∨ (f ∧ more);h ⟶ h
   using AndChopA Prop03 Prop10 assms int-simps(33) inteq-reflection by fastforce
 have 3: ⋀ n. ⊢ (power (f ∧ more) n);g ⟶ h
   using 21 ChopInductL[of g LIFT(f ∧ more) h]  assms by auto
 have 4: ⊢ (∃ n. (power (f ∧ more) n);g) ⟶ h
 using 3 by fastforce
 from 1 2 4 show ?thesis by (metis inteq-reflection)
qed

lemma ChopstarInductMoreL:
 assumes ⊢ g ∨ (f ∧ more);h ⟶ h
 shows  ⊢ (chopstar f);g ⟶ h
proof −
 have 1: ⊢ (chopstar f);g = ((∃ n. power (f ∧ more) n));g
 by (simp add: chopstar-d-def powerstar-d-def LeftChopEqvChop)
 have 2: ⊢ (∃ n. power (f ∧ more) n);g =
          (∃ n. (power (f ∧ more) n);g)
 using ExistChopPower by fastforce
 have 3: ⋀ n. ⊢ (power (f ∧ more) n);g ⟶ h
```

using *ChopInductL assms* **by** (*metis*)
have *4*: $\vdash$ ($\exists$ *n*. (*power* (*f* $\wedge$ *more*) *n*);*g*) $\longrightarrow$ *h*
using *3* **by** *fastforce*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *PowerstarInductR*:
assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;(*powerstar f*) $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ *g*;(*powerstar f*) = *g*;($\exists$ *n*. *power f n*)
by (*simp add*: *powerstar-d-def*)
have *2*: $\vdash$ (*g*;($\exists$ *n*. *power f n*)) = ($\exists$ *n*. *g*;(*power f n*))
using *ChopExistPower* **by** *blast*
have *3*: $\bigwedge$ *n*. $\vdash$ *g*;(*power f n*) $\longrightarrow$ *h*
using *ChopInductR assms* **by** *blast*
have *4*: $\vdash$ ($\exists$ *n*. *g*;(*power f n*)) $\longrightarrow$ *h*
using *3* **by** (*auto simp add*: *Valid-def*)
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *ChopstarInductR*:
assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;(*chopstar f*) $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ *g*;(*chopstar f*) =
g;(($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*))
by (*simp add*: *chopstar-d-def powerstar-d-def*)
have *2*: $\vdash$ (*g*;($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*)) =
(($\exists$ *n*. *g*;*power* (*f* $\wedge$ *more*) *n*))
using *ChopExistPower LeftChopEqvChop* **by** *fastforce*
have *21*: $\vdash$ *g* $\vee$ *h*;(*f* $\wedge$ *more*) $\longrightarrow$ *h*
using *ChopAndA assms* **by** *fastforce*
have *3*: $\bigwedge$ *n*. $\vdash$ *g*;(*power* (*f* $\wedge$ *more*) *n*) $\longrightarrow$ *h*
using *21 ChopInductR*[*of g h LIFT*(*f* $\wedge$ *more*)] *assms* **by** *auto*
have *4*: $\vdash$ ($\exists$ *n*. *g*;((*power* (*f* $\wedge$ *more*) *n*))) $\longrightarrow$ *h*
using *3* **by** (*auto simp add*: *Valid-def*)
from *1 2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopstarInductMoreR*:
assumes $\vdash$ *g* $\vee$ *h*;(*f* $\wedge$ *more*) $\longrightarrow$ *h*
shows $\vdash$ *g*;(*chopstar f*) $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ *g*;(*chopstar f*) = *g*;(($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*))
by (*simp add*: *chopstar-d-def powerstar-d-def*)
have *2*: $\vdash$ (*g*;($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*)) =
(($\exists$ *n*. *g*;*power* (*f* $\wedge$ *more*) *n*))
using *ChopExistPower LeftChopEqvChop* **by** *fastforce*
have *3*: $\bigwedge$ *n*. $\vdash$ *g*;(*power* (*f* $\wedge$ *more*) *n*) $\longrightarrow$ *h*

using *ChopInductR assms* **by** (*metis*)
have *4*: ⊢ (∃ *n*. *g*;((*power* (*f* ∧ *more*) *n*))) ⟶ *h*
using *3* **by** (*auto simp add*: *Valid-def*)
from *1 2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *PSAndMoreImpPS*:
⊢ *powerstar* (*f* ∧ *more*) ⟶ *powerstar f*
proof −
have *2*: ⊢ *empty* ∨ ((*f* ∧ *more*));*powerstar f* ⟶ *powerstar f*
using *AndChopA PSEqvEmptyOrChopPS* **by** *fastforce*
have *3*: ⊢ *powerstar* (*f* ∧ *more*);*empty* ⟶ *powerstar f*
using *2 PowerstarInductL* **by** *blast*
from *2 3* **show** *?thesis* **by** (*metis ChopEmpty int-eq*)
qed

lemma *PSImpAndMorePS*:
⊢ *powerstar f* ⟶ *powerstar* (*f* ∧ *more*)
by (*meson ChopEmpty PSEqvEmptyOrChopPS PowerstarInductMoreL int-iffD2 lift-imp-trans*)

lemma *FPSAndMoreEqvFPS*:
⊢ *powerstar* (*f* ∧ *more*) = *powerstar f*
using *PSAndMoreImpPS PSImpAndMorePS* **by** *fastforce*

lemma *ChopstarImpPowerstar*:
⊢ $f^\star$ ⟶ *powerstar f*
by (*metis ChopEmpty ChopstarInductL PSEqvEmptyOrChopPS int-eq int-iffD2*)

lemma *PowerstarImpChopstar*:
⊢ *powerstar f* ⟶ $f^\star$
by (*metis CSEqvOrChopCS ChopEmpty PowerstarInductL int-iffD2 inteq-reflection*)

lemma *ChopstarEqvPowerstar*:
⊢ $f^\star$ = *powerstar f*
using *ChopstarImpPowerstar PowerstarImpChopstar* **by** *fastforce*

lemma *PowerchopAndMore*:
⊢ ((*power* (*f* ∧ *more*) (*Suc n*)) ∧ *more*) = (*power* (*f* ∧ *more*) (*Suc n*))
proof
(*induct n*)
case *0*
then show *?case*
by (*metis* (*no-types*, *lifting*) *AndChopB DiamondEmpty MoreChopEqvNextDiamond Prop10 int-eq-true inteq-reflection more-d-def pow-0 pow-Suc*)
next
case (*Suc n*)
then show *?case*
by (*metis Prop10 Prop11 Prop12 RightChopImpMoreRule pow-Suc*)
qed

lemma *ExistPowerAndMoreExpand*:
$\vdash (\exists\ n.\ power\ (f \wedge more)\ n) = (\ empty \vee (\exists\ n.\ (power\ (f \wedge more)\ (Suc\ n))))$
using *powersem1*[*of LIFT*(*f* ∧ *more*)] **by** *auto*

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 *ChopplusCommuteImpA*:
$\vdash f^\star;f \longrightarrow f;f^\star$
by (*metis CSEqvOrChopCS ChopAndB ChopEmpty ChopstarInductL EmptyImpCS Prop02 Prop03 Prop10 inteq-reflection*)

lemma *ChopplusCommuteImpB*:
$\vdash f;f^\star \longrightarrow f^\star;f$
by (*metis ChopstarEqvPowerstar PowerStarCommute int-iffD1 inteq-reflection powerstar-d-def*)

lemma *ChopplusCommute*:
$\vdash f;f^\star = f^\star;f$
using *ChopplusCommuteImpA ChopplusCommuteImpB* **by** *fastforce*

lemma *CSEqvOrChopCSB*:
$\vdash f^\star = ($*empty* $\vee (f^\star;f))$
by (*meson CSEqvOrChopCS ChopplusCommute Prop06*)

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 *PowerChopPower*:
$\vdash$ (*power* (*f* $\wedge$ *more*) *n*); (*power* (*f* $\wedge$ *more*) *k*) = (*power* (*f* $\wedge$ *more*) (*n*+*k*))
proof
(*induct n arbitrary*: *k*)
case *0*
then show *?case* **using** *EmptyChopSem* **by** *auto*
next
case (*Suc n*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *ChopAssoc add-Suc inteq-reflection pow-Suc*)
qed

lemma *CSChopCS*:
$\vdash f^\star$; $f^\star = f^\star$
by (*metis CSEqvOrChopCS ChopEmpty ChopstarInductL EmptyImpCS Prop02 Prop03 Prop11 RightChopImp-Chop*
inteq-reflection)

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

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

lemma *CSChopCSImpCS*:
$\vdash$ $f^\star$; $f^\star \longrightarrow f^\star$

proof −
have *1*: $\vdash f^\star = (empty \lor (f \land more); f^\star)$
by (*rule ChopstarEqv*)
hence *2*: $\vdash f^\star; f^\star = (f^\star \lor ((f \land more); f^\star); f^\star)$
by (*rule EmptyOrChopEqvRule*)
have *21*: $\vdash f^\star; f^\star \land \neg (f^\star) \longrightarrow ((f \land more); f^\star); f^\star$
using *2* **by** *auto*
have *22*: $\vdash \neg (f^\star) = (\neg empty \land \neg ((f \land more); f^\star))$
using *1* **by** *fastforce*
have *23*: $\vdash \neg (f^\star) \longrightarrow \neg ((f \land more); f^\star)$
using *2 22* **by** *fastforce*
have *24*: $\vdash f^\star; f^\star \land \neg (f^\star) \longrightarrow \neg (f^\star)$
by *auto*
have *25*: $\vdash f^\star; f^\star \land \neg (f^\star) \longrightarrow \neg ((f \land more); f^\star)$
using *23 24 MP* **by** *auto*
have *3*: $\vdash f^\star; f^\star \land \neg (f^\star) \longrightarrow ((f \land more); f^\star); f^\star \land \neg ((f \land more); f^\star)$
using *21 25* **by** *fastforce*
have *4*: $\vdash (f \land more);(f^\star; f^\star)= ((f \land more); f^\star); f^\star$
by (*rule ChopAssoc*)
have *5*: $\vdash f^\star; f^\star \land \neg (f^\star) \longrightarrow (f \land more); (f^\star; f^\star) \land \neg ((f \land more); f^\star)$
using *3 4* **by** *fastforce*
have *6*: $\vdash f \land more \longrightarrow more$
by *auto*
from *5 6* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

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

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

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

lemma *ChopstarImp*:
 assumes $\vdash f;(chopstar\ g) \lor empty \longrightarrow (chopstar\ g)$
 shows $\vdash (chopstar\ f) \longrightarrow (chopstar\ g)$

using *assms ChopstarInductL ChopEmpty*
by (*metis int-eq int-simps*(*33*) *lift-and-com*)

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 *CSImpCSCS*:
$\vdash f^\star \longrightarrow (f^\star)^\star$
using *ImpCS* **by** *auto*

lemma *CSCSEqvCS*:
$\vdash (f^\star)^\star = f^\star$
by (*simp add*: *CSCSImpCS CSImpCSCS int-iffI*)

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*) *ChopAndEmptyEqvEmptyChopEmpty ChopEmpty ChopImpDiamond DiamondFin Finprop*(*1*) *NotEmptyEqvMore Prop12 always-d-def empty-d-def fin-d-def int-simps*(*14*) *int-simps*(*2*) *int-simps*(*21*) *inteq-reflection sometimes-d-def*)

lemma *ChopCSAndEmptyEqvAndEmpty*:
$\vdash ((f;f^\star) \wedge empty) = (f \wedge empty)$
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*: ⊢ $f \longrightarrow g$ **using** *assms* **by** *blast*
 have *2*: ⊢ $(f;\, g) \longrightarrow g$ **using** *assms* **by** *blast*
 have *3*: ⊢ $(f \wedge$ *more* $);\, g \longrightarrow f;\, g$ **by** (*rule AndChopA*)
 have *4*: ⊢ $(f \wedge$ *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 ⊢ $f = g$
 shows ⊢ $f;f^\star = g;g^\star$
proof −
 have *1*: ⊢ $f = g$ **using** *assms* **by** *auto*
 hence *2*: ⊢ $f \longrightarrow g$ **by** *auto*
 hence *3*: ⊢ $f;f^\star \longrightarrow g;g^\star$ **by** (*rule ChopPlusImpChopPlus*)
 have *4*: ⊢ $g \longrightarrow f$ **using** *1* **by** *auto*
 hence *5*: ⊢ $g;g^\star \longrightarrow f;f^\star$ **by** (*rule ChopPlusImpChopPlus*)
 from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *CSEqvCS*:
 assumes ⊢ $f = g$
 shows ⊢ $f^\star = g^\star$
proof −
 have *1*: ⊢ $f = g$ **using** *assms* **by** *auto*
 hence *2*: ⊢ $f;f^\star = g;g^\star$ **by** (*rule ChopPlusEqvChopPlus*)
 hence *3*: ⊢ (*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*:
⊢ $(f \wedge g)^\star \longrightarrow f^\star$
proof −
 have *1*: ⊢ $f \wedge g \longrightarrow f$ **by** *auto*
 from *1* **show** *?thesis* **using** *CSImpCS* **by** *blast*
qed

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

lemma *CSIntro*:
 assumes ⊢ $f \wedge$ *more* $\longrightarrow (g \wedge$ *more* $);\, f$
 shows ⊢ $f \longrightarrow g^\star$
proof −
 have *1*: ⊢ $f \wedge$ *more* $\longrightarrow (g \wedge$ *more* $);\, f$
  using *assms* **by** *auto*

have *2*: ⊢ *more* = (¬ *empty*)
  by (*auto simp*: *empty-d-def*)
have *3*: ⊢ $f \wedge \neg$ *empty* ⟶ ($g \wedge$ *more*); f
  using *1 2* **by** *fastforce*
have *4*: ⊢ $g^\star$ = (*empty* ∨ ($g \wedge$ *more*); $g^\star$)
  by (*rule ChopstarEqv*)
hence *41*: ⊢ (¬(*empty* ∨ ($g \wedge$ *more*); $g^\star$)) = (¬*empty* ∧ ¬(($g \wedge$ *more*); $g^\star$))
  by *fastforce*
have *411*: ⊢ (¬*empty* ∧ ¬(($g \wedge$ *more*); $g^\star$)) = (*more* ∧ ¬(($g \wedge$ *more*); $g^\star$))
  using *NotEmptyEqvMore* **by** *fastforce*
have *42*: ⊢ ¬($g^\star$) = (*more* ∧ ¬(($g \wedge$ *more*); $g^\star$))
  using *4 41 411* **by** *fastforce*
have *43*: ⊢f ∧ ¬($g^\star$) ⟶ f ∧ *more* ∧ ¬ (($g \wedge$ *more*); $g^\star$)
  using *42* **by** *fastforce*
have *44*: ⊢ f ∧ *more* ∧ ¬ (($g \wedge$ *more*); $g^\star$) ⟶ ($g \wedge$ *more*); f ∧ ¬ (($g \wedge$ *more*); $g^\star$)
  using *3 43 1* **by** *auto*
have *5*: ⊢ f ∧ ¬ ($g^\star$) ⟶
    ($g \wedge$ *more*); f ∧ ¬ (($g \wedge$ *more*); $g^\star$)
  using *43 44 lift-imp-trans* **by** *fastforce*
have *6*: ⊢ $g \wedge$ *more* ⟶ *more*
  by *auto*
from *5 6* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *CSElimWithoutMore*:
assumes ⊢ *empty* ⟶ g
  ⊢ f; g ⟶ g
shows ⊢ $f^\star$ ⟶ g
proof −
have *1*: ⊢ *empty* ⟶ g **using** *assms* **by** *blast*
have *2*: ⊢ f; g ⟶ g **using** *assms* **by** *blast*
have *3*: ⊢ ($f \wedge$ *more*); g ⟶ f; g **by** (*rule AndChopA*)
have *4*: ⊢ ($f \wedge$ *more*); g ⟶ g **using** *2 3 lift-imp-trans* **by** *blast*
from *1 4* **show** *?thesis* **using** *CSElim* **by** *blast*
qed

lemma *ChopAssocB*:
⊢ (f;g);h = f;(g;h)
using *ChopAssoc* **by** *fastforce*

lemma *CSChopEqvChopOrRule*:
assumes ⊢ f= ($g^\star$; h)
shows ⊢ f = ((g; f) ∨ h)
proof −
have *1*: ⊢ f = ($g^\star$; h) **using** *assms* **by** *auto*
have *2*: ⊢ $g^\star$ = (*empty* ∨ (g; $g^\star$)) **by** (*rule CSEqvOrChopCS*)
hence *3*: ⊢ $g^\star$; h = (h ∨ ((g; $g^\star$); h)) **by** (*rule EmptyOrChopEqvRule*)
have *4*: ⊢ (g; $g^\star$); h = g; ($g^\star$; h) **by** (*rule ChopAssocB*)
hence *41*: ⊢ $g^\star$; h = (h ∨ g; ($g^\star$; h)) **using** *3* **by** *fastforce*
have *5*: ⊢ 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*:
$\vdash\ \ ba\ (f \longrightarrow g) \longrightarrow f^{\star} \longrightarrow g^{\star}$
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*:
$\vdash$ *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*:
$\vdash$ *ba* $f \wedge g^\star \longrightarrow (f \wedge g)^\star$
proof −
have *1*: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
hence *2*: $\vdash$ *ba* $f \longrightarrow$ *ba* $(g \longrightarrow f \wedge g)$ **by** (*rule BaImpBa*)
have *3*: $\vdash$ *ba* $(g \longrightarrow f \wedge g) \longrightarrow g^\star \longrightarrow (f \wedge g)^\star$ **by** (*rule BaCSImpCS*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

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

7.8 Properties of While

lemma *WhileEqvIf*:
$\vdash \mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f = \mathit{if}_i\ (\mathit{init}\ w)\ \mathit{then}\ (f; (\mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f))\ \mathit{else}\ \mathit{empty}$
proof −
have *1*: $\vdash \mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f = (((\mathit{init}\ w) \wedge f)^\star \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))$
by (*simp add*: *while-d-def*)
have *2*: $\vdash (\mathit{init}\ w \wedge f)^\star = (\mathit{empty} \vee ((\mathit{init}\ w \wedge f); (\mathit{init}\ w \wedge f)^\star))$
by (*rule CSEqvOrChopCS*)
have *21*: $\vdash (((\mathit{init}\ w) \wedge f)^\star \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))) =$
$((\mathit{empty} \vee ((\mathit{init}\ w \wedge f); (\mathit{init}\ w \wedge f)^\star)) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))$
using *2* **by** *fastforce*
have *22*: $\vdash ((\mathit{empty} \vee ((\mathit{init}\ w \wedge f); (\mathit{init}\ w \wedge f)^\star)) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))) =$
$((\mathit{empty} \wedge \mathit{fin}\ (\neg(\mathit{init}\ w))) \vee (((\mathit{init}\ w \wedge f); (\mathit{init}\ w \wedge f)^\star) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))))$
by *auto*
have *3*: $\vdash (\mathit{empty} \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))) = (\neg\ (\mathit{init}\ w) \wedge \mathit{empty})$
using *AndFinEqvChopAndEmpty EmptyChop* **by** (*metis int-eq*)
have *4*: $\vdash (\mathit{init}\ w \wedge f); (\mathit{init}\ w \wedge f)^\star = (\mathit{init}\ w \wedge (f; (\mathit{init}\ w \wedge f)^\star))$
by (*rule StateAndChop*)
have *41*: $\vdash (((\mathit{init}\ w \wedge f); (\mathit{init}\ w \wedge f)^\star) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))) =$
$(\mathit{init}\ w \wedge (f; (\mathit{init}\ w \wedge f)^\star) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))$
using *4* **by** *auto*
have *42*: $\vdash (\mathit{init}\ w \wedge (f; (\mathit{init}\ w \wedge f)^\star) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))) =$
$(\mathit{init}\ w \wedge (f; (\mathit{init}\ w \wedge f)^\star) \wedge \mathit{fin}\ (\mathit{init}\ (\neg\ w)))$
using *Initprop*(*2*) **by** (*metis StateAndEmptyChop int-eq*)
have *5*: $\vdash ((f; ((\mathit{init}\ w \wedge f)^\star)) \wedge (\mathit{fin}\ (\mathit{init}\ (\neg\ w))))$
$= (f; ((\mathit{init}\ w \wedge f)^\star \wedge (\mathit{fin}\ (\mathit{init}\ (\neg\ w)))))$
by (*rule ChopAndFin*)
have *51*: $\vdash (f; ((\mathit{init}\ w \wedge f)^\star \wedge (\mathit{fin}\ (\mathit{init}\ (\neg\ w))))) =$
$(f; ((\mathit{init}\ w \wedge f)^\star \wedge (\mathit{fin}\ (\neg\ (\mathit{init}\ w)))))$
using *Initprop*(*2*) **by** (*metis FinAndChop int-eq*)
have *52*: $\vdash (\mathit{init}\ w \wedge (f; (\mathit{init}\ w \wedge f)^\star) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))) =$
$(\mathit{init}\ w \wedge (f; ((\mathit{init}\ w \wedge f)^\star \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))))$
using *42 5 51* **by** *fastforce*
have *6*: $\vdash (f; ((\mathit{init}\ w \wedge f)^\star \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))) = f; \mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f$
by (*simp add*: *while-d-def*)
have *61*: $\vdash (\mathit{init}\ w \wedge (f; ((\mathit{init}\ w \wedge f)^\star \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))))) =$
$(\mathit{init}\ w \wedge (f; \mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f))$ **using** *6*
by *auto*
have *62*: $\vdash (\mathit{empty} \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))) \vee (((\mathit{init}\ w \wedge f); (\mathit{init}\ w \wedge f)^\star) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))$
$= (\neg\ (\mathit{init}\ w) \wedge \mathit{empty}) \vee (\mathit{init}\ w \wedge (f; \mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f))$
using *21 22 3 4 52 61* **by** *fastforce*
have *7*: $\vdash \mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f$
$= ((\neg\ (\mathit{init}\ w) \wedge \mathit{empty}) \vee (\mathit{init}\ w \wedge (f; \mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f)))$
using *1 21 22 62*
by (*metis 3 41 42 5 51 inteq-reflection*)
have *71*: $\vdash \mathit{if}_i\ (\mathit{init}\ w)\ \mathit{then}\ (f; (\mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f))\ \mathit{else}\ \mathit{empty} =$
$((\neg\ (\mathit{init}\ w) \wedge \mathit{empty}) \vee (\mathit{init}\ w \wedge (f; \mathit{while}\ (\mathit{init}\ w)\ \mathit{do}\ f)))$
by (*auto simp*: *ifthenelse-d-def*)
from *7 71* **show** *?thesis* **by** *fastforce*
qed

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

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

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

lemma *WhileEqvEmptyOrChopWhile*:

$\vdash$ *while* (*init w*) *do f* = ((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*f* ∧ *more*);*while* (*init w*) *do f*))
proof –
 have *1*: ⊢ (*init w* ∧ *f*)* = (*empty* ∨ ((*init w* ∧ *f*)∧ *more*); (*init w* ∧ *f*)*)
 by (*rule ChopstarEqv*)
 have *2*: ⊢ ((*init w* ∧ *f*) ∧ *more*) = (*init w* ∧ (*f* ∧ *more*))
 by *auto*
 hence *3*: ⊢ ((*init w* ∧ *f*)∧ *more*); (*init w* ∧ *f*)* = (*init w* ∧ *f* ∧ *more*); (*init w* ∧ *f*)*
 by (*rule LeftChopEqvChop*)
 have *4*: ⊢ (*init w* ∧ *f*)* = (*empty* ∨ (*init w* ∧ *f* ∧ *more*); (*init w* ∧ *f*)*)
 using *1 3* **by** *fastforce*
 have *5*: ⊢ ((*init w* ∧ *f*)* ∧ *fin* (¬ (*init w*))) =
 ((*empty* ∧ *fin* (¬ (*init w*))) ∨
 ((*init w* ∧ *f* ∧ *more*); (*init w* ∧ *f*)*∧ *fin* (¬ (*init w*))))
 using *1 4* **by** *fastforce*
 have *6*: ⊢ (*empty* ∧ *fin* (¬ (*init w*))) = (¬ (*init w*) ∧ *empty*)
 using *AndFinEqvChopAndEmpty EmptyChop* **by** (*metis int-eq*)
 have *7*: ⊢ (*init w* ∧ *f* ∧ *more*); (*init w* ∧ *f*)* = (*init w* ∧ (*f* ∧ *more*); (*init w* ∧ *f*)*)
 by (*rule StateAndChop*)
 have *8*: ⊢ (((*f* ∧ *more*); (*init w* ∧ *f*)*) ∧ *fin* (*init* (¬ *w*))) =
 ((*f* ∧ *more*); ((*init w* ∧ *f*)* ∧ *fin* (*init* (¬ *w*))))
 by (*rule ChopAndFin*)
 have *81*: ⊢ *fin* (*init* (¬ *w*)) = *fin* (¬ (*init w*))
 using *FinEqvFin Initprop*(*2*) **by** *fastforce*
 have *82*: ⊢ ((*f* ∧ *more*); (*init w* ∧ *f*)* ∧ *fin* (¬ (*init w*))) =
 ((*f* ∧ *more*); ((*init w* ∧ *f*)* ∧ *fin* (¬ (*init w*))))
 using *8 81*
 by (*metis inteq-reflection*)
 have *9*: ⊢ ((*init w* ∧ *f*)* ∧ *fin* (¬ (*init w*))) =
 ((¬ (*init w*) ∧ *empty*) ∨
 (*init w* ∧ (*f* ∧ *more*); ((*init w* ∧ *f*)* ∧ *fin* (¬ (*init w*)))))
 using *5 6 7 82* **by** *fastforce*
 from *9* **show** *?thesis* **by** (*simp add*: *while-d-def*)
qed

lemma *WhileIntro*:
 assumes ⊢ ¬ (*init w*) ∧ *f* ⟶ *empty*
 ⊢ *init w* ∧ *f* ⟶ (*g* ∧ *more*); *f*
 shows ⊢ *f* ⟶ *while* (*init w*) *do g*
proof –
 have *1*: ⊢ ¬ (*init w*) ∧ *f* ⟶ *empty*
 using *assms* **by** *blast*
 have *2*: ⊢ *init w* ∧ *f* ⟶ (*g* ∧ *more*); *f*
 using *assms* **by** *blast*
 have *3*: ⊢ *while* (*init w*) *do g* =
 ((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*); *while* (*init w*) *do g*))
 by (*rule WhileEqvEmptyOrChopWhile*)
 hence *31*: ⊢ ¬ (*while* (*init w*) *do g*) =
 (¬((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*); *while* (*init w*) *do g*)))
 by *fastforce*
 hence *32*: ⊢ (*f* ∧ ¬ (*while* (*init w*) *do g*)) =

$(f \wedge \neg((\neg(init\ w) \wedge empty) \vee (init\ w \wedge (g \wedge more); while\ (init\ w)\ do\ g)))$
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

7.9 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*: ⊢ *halt* (*init w*); *g* ⟶ ¬ (*halt* (*init w*); (¬ *g*))
by (*rule HaltChopImpNotHaltChopNot*)
hence *41*: ⊢ (*halt* (*init w*); (¬ *g*)) ⟶ ¬(*halt* (*init w*); *g*)
by *auto*
have *42*: ⊢ (*halt* (*init w*) ∧ *f*); *f1* ⟶
¬(*halt* (*init w*); *g*) ∨ ((*halt* (*init w*) ∧ *f*); (*f1* ∧ *g*))
using *31 41* **by** *fastforce*
from *42* **show** *?thesis* **by** *auto*
qed

lemma *HaltImpBoxYields*:
⊢ (*halt* (*init w*)); *f* ⟶ (□(¬ (*init w*))) *yields* ((*halt* (*init w*)); *f*)
proof −
have *1*: ⊢ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) ⟶ *di* (□ (¬ (*init w*)))
by (*rule ChopImpDi*)
have *2*: ⊢ □ (¬ (*init w*)) ⟶ ¬ (*init w*)
by (*rule BoxElim*)
hence *3*: ⊢ *di* (□ (¬ (*init w*))) ⟶ *di* (¬ (*init w*))
by (*rule DiImpDi*)
have *4*: ⊢ *di* (*init* (¬ *w*)) = (*init* (¬*w*))
by (*rule DiState*)
have *41*: ⊢ (*init* (¬ *w*)) = (¬ (*init w*))
using *Initprop*(*2*) **by** *fastforce*
have *42*: ⊢ *di* (¬ (*init w*)) = (¬(*init w*))
using *4 41* **by** (*metis inteq-reflection*)
have *5*: ⊢ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) ⟶ ¬ (*init w*)
using *1 2 42* **using** *3* **by** *fastforce*
hence *51*: ⊢ (*halt* (*init w*); *f*) ∧ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) ⟶
(*halt* (*init w*); *f*) ∧ ¬ (*init w*)
by *fastforce*
have *6*: ⊢ *halt* (*init w*); *f* = *if* $_i$ (*init w*) *then f else* (○(*halt* (*init w*); *f*))
by (*rule HaltChopEqv*)
hence *61*: ⊢ (*halt* (*init w*); *f* ∧ ¬ (*init w*)) =
((*if* $_i$ (*init w*) *then f else* (○(*halt* (*init w*); *f*))) ∧ ¬ (*init w*))
using *6* **by** *auto*
have *62*: ⊢ (*if* $_i$ (*init w*) *then f else* (○(*halt* (*init w*); *f*))) ∧
¬ (*init w*) ⟶ (○(*halt* (*init w*); *f*))
by (*auto simp*: *ifthenelse-d-def*)
have *63*: ⊢ *halt* (*init w*); *f* ∧ ¬ (*init w*) ⟶ (○(*halt* (*init w*); *f*))
using *61 62* **by** *fastforce*
have *7*: ⊢ (*halt* (*init w*); *f*) ∧ (□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) ⟶
○((*halt* (*init w*)); *f*)
using *51 63* **using** *lift-imp-trans* **by** *blast*
have *8*: ⊢ □ (¬ (*init w*)) ⟶ *empty* ∨ ○(□(¬(*init w*)))
using *BoxBoxImpBox BoxEqvAndEmptyOrNextBox* **by** *fastforce*
hence *9*: ⊢ ((□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) ⟶
¬ (*halt* (*init w*); *f*) ∨ ○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
by (*rule EmptyOrNextChopImpRule*)
hence *10*: ⊢ ((*halt* (*init w*)); *f*) ∧ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) ⟶
○((□(¬ (*init w*))); (¬ (*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

7.10 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

end

8 First Order Finite ITL theorems

theory *FOTheorems*
imports
Theorems
begin

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

lemma *EExI-unl*:
w $\models$ *f x* $\Longrightarrow$ *w* $\models$ ($\exists\exists$ *x*. *f x*)
using *EExVal* **by** *auto*

lemma *EExNoDep*:

$\vdash (\exists\exists\ x.\ g) = g$
proof −
have *1*: $\vdash g \longrightarrow (\exists\exists\ x.\ g)$ **by** (*meson EExI*)
have *2*: $\bigwedge x. \vdash g \longrightarrow g$ **by** *simp*
have *3*: $\vdash (\exists\exists\ x.\ g) \longrightarrow g$ **using** *2* **by** (*meson EExE*)
from *1 3* **show** *?thesis* **using** *int-iffI* **by** *blast*
qed

lemma *AAxNoDep*:
$\vdash (\forall\forall\ x.\ g) = g$
using *EExNoDep*[*of LIFT*($\neg$ *g*)] *AAxDef EExE EExI*
by (*simp add*: *exist-state-d-def forall-state-d-def intI*)

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

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

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

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

lemma *EExImpNoDepDist*:
assumes $\vdash f \longrightarrow g\,x$
shows $\vdash f \longrightarrow (\exists\exists\ x.\ g\,x)$
using *assms* **by** (*metis EExI lift-imp-trans*)

lemma *EExOrDist-1*:
$\vdash (\exists\exists\ x.\ h\,x) \longrightarrow (\exists\exists\ x.\ (f\,x) \lor (h\,x))$
proof −
have *1*: $\bigwedge x. \vdash h\,x \longrightarrow f\,x \lor h\,x$ **by** (*simp add*: *Valid-def*)
have *2*: $\bigwedge x. \vdash f\,x \lor h\,x \longrightarrow (\exists\exists\ x.\ (f\,x) \lor (h\,x))$ **by** (*meson EExI*)
have *3*: $\bigwedge x. \vdash h\,x \longrightarrow (\exists\exists\ x.\ (f\,x) \lor (h\,x))$ **using** *1 2* **by** (*meson lift-imp-trans*)
from *3* **show** *?thesis* **using** *EExE* **by** *blast*

qed

lemma *EExOrDist-2*:
$\vdash (\exists\exists\ x.\ f\,x) \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$
proof −
have *1*: $\bigwedge x. \vdash f\,x \longrightarrow f\,x \vee h\,x$ **by** (*simp add*: *Valid-def*)
have *2*: $\bigwedge x. \vdash f\,x \vee h\,x \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$ **by** (*meson EExI*)
have *3*: $\bigwedge x. \vdash f\,x \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$ **using** *1 2* **by** (*meson lift-imp-trans*)
from *3* **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-3*:
$\vdash (\exists\exists\ x.\ f\,x) \vee (\exists\exists\ x.\ h\,x) \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$
using *EExOrDist-2 EExOrDist-1* **by** *fastforce*

lemma *EExOrDist-4*:
$\vdash (\exists\exists\ x.\ (f\,x) \vee (h\,x)) \longrightarrow (\exists\exists\ x.\ f\,x) \vee (\exists\exists\ x.\ h\,x)$
proof −
have *1*: $\bigwedge x. \vdash (f\,x) \vee (h\,x) \longrightarrow (\exists\exists\ x.\ f\,x) \vee (\exists\exists\ x.\ h\,x)$
by (*simp add*: *EExI-unl intI*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

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

lemma *EExOrImport-1*:
$\vdash g \longrightarrow (\exists\exists\ x.\ g \vee (f\,x))$
by (*simp add*: *EExI-unl Valid-def*)

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

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

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

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

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

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

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

lemma *EExAndDist*:
assumes $\vdash f\,x \wedge g\,x$
shows $\vdash (\exists\exists\ x.\ f\,x) \wedge (\exists\exists\ x.\ g\,x)$
proof −
have *1*: $\vdash f\,x$ **using** *assms* **by** *fastforce*
have *2*: $\vdash g\,x$ **using** *assms* **by** *fastforce*
have *3*: $\vdash (\exists\exists\ x.\ f\,x)$ **using** *1* **by** (*meson EExI MP*)
have *4*: $\vdash (\exists\exists\ x.\ g\,x)$ **using** *2* **by** (*meson EExI MP*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

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

lemma *Spec*:
$\vdash (\forall\forall\ x.\ f\,x) \longrightarrow f\,x$
proof −
have *1*: $\vdash \neg(f\,x) \longrightarrow (\exists\exists\ x.\ \neg(f\,x))$ **by** (*meson EExI*)
have *2*: $\vdash \neg(\exists\exists\ x.\ \neg(f\,x)) \longrightarrow f\,x$ **using** *1* **by** *auto*

```
 from 2 show ?thesis using AAxDef by fastforce
qed

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

lemma AAxI:
 assumes ⋀ x. ⊢ f x
 shows  ⊢ (∀∀ x. f x)
using assms by (simp add: Valid-def exist-state-d-def forall-state-d-def)

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

lemma AAxAndDist:
 ⊢ (∀∀ x. (f x) ∧ (g x)) = ((∀∀ x. f x) ∧ (∀∀ x. g x))
proof −
 have 1: ⊢ ((∃∃ x. ¬(f x)) ∨ (∃∃ x. ¬(g x))) = (∃∃ x. ¬(f x) ∨ ¬(g x)) by (simp add:EExOrDist)
 have 2: ⊢ ((∃∃ x. ¬(f x))) = (¬(∀∀ x. f x) )  using AAxDef by fastforce
 have 3: ⊢ ((∃∃ x. ¬(g x))) = (¬(∀∀ x. g x) )  using AAxDef by fastforce
 have 4: ⊢ ((∃∃ x. ¬(f x)) ∨ (∃∃ x. ¬(g x))) = (¬(∀∀ x. f x) ∨ ¬(∀∀ x. g x) )
      using 2 3 by fastforce
 have 5: ⋀ x . ⊢ (¬(f x) ∨ ¬(g x)) = (¬((f x) ∧ (g x))) by auto
 have 6: ⊢ (∃∃ x. ¬(f x) ∨ ¬(g x)) = (∃∃ x. ¬((f x) ∧ (g x)))  using 5 by (simp add: EExEqvRule)
 have 7: ⊢ (∃∃ x. ¬((f x) ∧ (g x))) = (¬(∀∀ x. (f x) ∧ (g x))) using AAxDef by fastforce
 have 8: ⊢ (¬(∀∀ x. f x) ∨ ¬(∀∀ x. g x) ) = (¬( (∀∀ x. f x) ∧ (∀∀ x. g x))) by fastforce
 have 9: ⊢ (¬( (∀∀ x. f x) ∧ (∀∀ x. g x))) = (¬(∀∀ x. (f x) ∧ (g x)))
      using 1 4 6 7 8 by fastforce
 from 9 show ?thesis by fastforce
qed

lemma AAxAndImport:
 ⊢ (g ∧ (∀∀ x. f x)) = (∀∀ x. g ∧ f x)
proof −
 have 1: ⊢ (¬ g ∨ (∃∃ x. ¬(f x))) = (∃∃ x. ¬ g ∨ ¬(f x))  by (simp add: EExOrImport)
 have 2: ⊢ ( (∃∃ x. ¬(f x))) = (¬((∀∀ x. f x))) using AAxDef by fastforce
 have 3: ⊢ ( ¬ g ∨ (∃∃ x. ¬(f x))) = (¬(g ∧ (∀∀ x. f x))) using 2 by fastforce
 have 4: ⋀ x. ⊢ (¬ g ∨ ¬(f x)) = (¬(g ∧ f x)) by auto
 have 5: ⊢ (∃∃ x. ¬ g ∨ ¬(f x)) = (∃∃ x. ¬(g ∧ f x)) using 4 by (simp add: EExEqvRule)
 have 6: ⊢ (∃∃ x. ¬(g ∧ f x)) = (¬(∀∀ x. g ∧ f x)) using AAxDef by fastforce
 have 7: ⊢ (¬(g ∧ (∀∀ x. f x))) = (¬(∀∀ x. g ∧ f x)) using 1 3 5 6 by fastforce
 from 7 show ?thesis by fastforce
qed
```

lemma *AAxOrImport*:
$\vdash (g \lor (\forall\forall\ x.\ f\,x)) = (\forall\forall\ x.\ g \lor f\,x)$
proof –
have *1*: $\vdash (\neg\ g \land (\exists\exists\ x.\ \neg(f\,x))) = (\exists\exists\ x.\ \neg\ g \land \neg(f\,x))$ **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*: $\vdash (\ \neg\ g \land (\exists\exists\ x.\ \neg(f\,x))) = (\neg(g \lor (\forall\forall\ x.\ f\,x)))$ **using** *2* **by** *fastforce*
have *4*: $\bigwedge x.\ \vdash (\neg\ g \land \neg(f\,x)) = (\neg(g \lor f\,x))$ **by** *auto*
have *5*: $\vdash (\exists\exists\ x.\ \neg\ g \land \neg(f\,x)) = (\exists\exists\ x.\ \neg(g \lor f\,x))$ **using** *4* **by** (*simp add*: *EExEqvRule*)
have *6*: $\vdash (\exists\exists\ x.\ \neg(g \lor f\,x)) = (\neg(\forall\forall\ x.\ g \lor f\,x))$ **using** *AAxDef* **by** *fastforce*
have *7*: $\vdash (\neg(g \lor (\forall\forall\ x.\ f\,x))) = (\neg(\forall\forall\ x.\ g \lor f\,x))$ **using** *1 3 5 6* **by** *fastforce*
from *7* **show** *?thesis* **by** *auto*
qed

lemma *EExImpChopRule*:
assumes $\vdash f\,x \longrightarrow g\,x$
shows $\vdash (\exists\exists\ x.\ h;(f\,x) \longrightarrow h;(g\,x))$
using *RightChopImpChop*[*of f x g x h*]
EExImpRule[*of* $\lambda x.\ LIFT(h;(f\,x))\ x\ \lambda x.\ LIFT(h;(g\,x))$] *assms* **by** *auto*

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

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

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

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

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

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

lemma *EExEExChopEqvEExChopEExB*:

$\vdash (\exists\exists\ y.\ (\exists\exists\ v.\ (f\ v);(g\ y)\ )) = (\exists\exists\ y.\ (\exists\exists\ v.\ (f\ v));\ (g\ y))$
by (*simp add*: *exist-state-d-def Valid-def chop-defs*) *blast*

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

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

lemma *CSPowerChop*:
$\vdash (f^\star) = (\exists\, n.\ power\ (f\ \wedge\ more)\ n)$
by (*simp add*: *chopstar-d-def powerstar-d-def Valid-def*)

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

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

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

lemma *TassignEqvExFin*:
$\vdash v \leftarrow e = (\exists\ c.\ \#c{=}e\ \wedge\ fin(\$v\ =\ \#c))$
by (*simp add*: *Valid-def temporal-assign-defs currentval-defs fin-defs*)

lemma *MoreImpNextassignEqvExNext*:
$\vdash more \longrightarrow (v := e) = (\exists\, c.\ \#c{=}e\ \wedge \bigcirc(\$v\ =\ \#c))$
by (*simp add*: *Valid-def next-assign-d-def next-val-d-def next-defs currentval-defs more-defs*)

lemma *MoreImpPrevassignEqvExPrevFin*:
$\vdash more \longrightarrow (v =: e) = (\exists\, c.\ \#c{=}e\ \wedge\ prev(fin(\$v{=}\ \#c))\ \ )$
by (*auto simp add*: *min.absorb1 Valid-def prev-assign-d-def fin-defs penult-val-d-def*
prev-defs currentval-defs more-defs)

end

9 Time Reversal

theory *TimeReversal*
imports

Theorems FOTheorems
begin

Time reversal operator is defined in [6].

9.1 Definition

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

syntax
-reverse-d :: *lift* ⇒ *lift* ((-^{r}) [*85*] *85*)

syntax (*ASCII*)
-reverse-d :: *lift* ⇒ *lift* ((*reverse* -) [*85*] *85*)

translations
-reverse-d ⇌ *CONST reverse-d*

9.2 Time reversal Rules

lemma *EExRev* :
$\vdash (\exists\exists\ x.\ F\ x)^r = (\exists\exists\ x.\ (F\ x)^r)$
by (*simp add*: *Valid-def exist-state-d-def reverse-d-def*)

lemma *rev-const* :
$\vdash (\#c)^r = \#c$
by (*auto simp*: *reverse-d-def*)

lemma *rev-fun1* :
$\vdash (f{<}x{>})^r = f{<}x^r{>}$
by (*auto simp*: *reverse-d-def*)

lemma *rev-fun2*:
$\vdash (f{<}x,y{>})^r = f{<}x^r,y^r{>}$
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*:
$\vdash (f;g)^r = (g^r;f^r)$
proof (*auto simp add*: *Valid-def chop-d-def reverse-d-def*)
show $\bigwedge w\ n.\ n \le intlen\ w \Longrightarrow$
$f\ (prefix\ n\ (intrev\ w)) \Longrightarrow$
$g\ (suffix\ n\ (intrev\ w)) \Longrightarrow$
$\exists\, n{\le}intlen\ w.\ g\ (intrev\ (prefix\ n\ w)) \wedge f\ (intrev\ (suffix\ n\ w))$
by (*metis diff-diff-cancel interval-intrev-prefix interval-intrev-suffix interval-suffix-intlen-bound interval-suffix-length*)
show $\bigwedge w\ n.\ n \le intlen\ w \Longrightarrow$
$g\ (intrev\ (prefix\ n\ w)) \Longrightarrow$
$f\ (intrev\ (suffix\ n\ w)) \Longrightarrow$
$\exists\, n{\le}intlen\ w.\ f\ (prefix\ n\ (intrev\ w)) \wedge g\ (suffix\ n\ (intrev\ w))$
by (*metis interval-intrev-prefix interval-intrev-suffix interval-suffix-intlen-bound interval-suffix-length*)
qed

lemma *RMoreEqvMore*:
$\vdash more^r = more$

by (*simp add*: *Valid-def more-d-def next-d-def chop-d-def skip-d-def reverse-d-def*)

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

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

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

lemma *RevPowerChop*:
$\vdash (power\ (f \wedge more)\ \ n)^r = (power\ ((f \wedge more)^r)\ n)$
proof
(*induct n*)
case *0*
then show *?case* **using** *REmptyEqvEmpty* **by** *auto*
next
case (*Suc n*)
then show *?case*
by (*metis PowerCommute RevChop inteq-reflection pow-Suc*)
qed

lemma *RevChopstar*:
$\vdash (f^\star)^r = (f^r)^\star$
proof −
have *1*: $\vdash (f^\star) = (\exists\, n.\ power\ (f \wedge more)\ n)$
by (*simp add*: *chopstar-d-def powerstar-d-def Valid-def*)
have *2*: $\vdash (f^\star)^r = (\exists\ n.\ power\ (f \wedge more)\ n)^r$
using *REqvRule 1* **by** *blast*
have *3*: $\vdash (\exists\ n.\ power\ (f \wedge more)\ n)^r = (\exists\ n.\ (power\ (f \wedge more)\ n)^r)$
by (*simp add*: *rev-exists*)
have *4*: $\vdash (\exists\ n.\ (power\ (f \wedge more)\ n)^r) = (\exists\ n.\ (power\ ((f \wedge more)^r)\ n))$
by (*simp add*: *RevPowerChop ExEqvRule*)
have *5*: $\vdash (f \wedge more)^r = (f^r \wedge more)$
by (*metis RMoreEqvMore inteq-reflection rev-fun2*)
hence *6*: $\vdash (\exists\ n.\ (power\ ((f \wedge more)^r)\ n)) = (\exists\ n.\ (power\ ((f^r \wedge more))\ n))$
by (*metis 4 inteq-reflection*)

have *7*: $\vdash (\exists\ n.\ (power\ ((f^r \wedge more))\ n)) = (f^r)^\star$
by (*simp add*: *chopstar-d-def powerstar-d-def Valid-def*)
from *2 3 4 6 7* **show** *?thesis* **by** *fastforce*
qed

lemmas *all-rev* = *rev-const rev-fun1 rev-fun2 rev-fun3 rev-forall rev-exists rev-exists1 rev-current rev-next rev-penult rev-fin RevSkip RevChop RevChopstar*

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

9.3 Properties of Time Reversal

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

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

lemma *RTrue*:
$\vdash (\#True)^r = \#True$
using *rev-const* **by** *fastforce*

lemma *ROr*:
$\vdash (f \vee g)^r = (f^r \vee g^r)$
by (*simp add*: *rev-fun2*)

lemma *RROr*:
$\vdash (f^r \vee g^r)^r = (f \vee g)$
proof −
have *1*: $\vdash (f^r \vee g^r)^r = ((f^r)^r \vee (g^r)^r)$ **using** *ROr* **by** *blast*
have *2*: $\vdash ((f^r)^r \vee (g^r)^r) = (f \vee g)$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RAnd*:
$\vdash (f \wedge g)^r = (f^r \wedge g^r)$
by (*simp add*: *rev-fun2*)

lemma *RRAnd*:
$\vdash (f^r \wedge g^r)^r = (f \wedge g)$
proof −
have *1*: $\vdash (f^r \wedge g^r)^r = ((f^r)^r \wedge (g^r)^r)$ **using** *RAnd* **by** *blast*
have *2*: $\vdash ((f^r)^r \wedge (g^r)^r) = (f \wedge g)$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RImpRule*:

assumes $\vdash f \longrightarrow g$
shows $\vdash f^r \longrightarrow g^r$
using *assms* **by** (*simp add*: *Valid-def reverse-d-def*)

lemma *RAndEmptyEqvAndEmpty*:
$\vdash (f \wedge empty)^r = (f \wedge empty)$
by (*simp add*: *Valid-def empty-defs reverse-d-def*,
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 (*simp add*: *all-rev-eq*(*12*) *all-rev-eq*(*13*) *all-rev-eq*(*2*) *next-d-def prev-d-def wnext-d-def*
wprev-d-def)

lemma *RRWNextEqvWPrev*:
$\vdash (wnext\ (f^r))^r = wprev(f)$
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 (*simp add*: *rev-fun2*)
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

lemma *AAxRev*:
$\vdash (\forall\forall\ x.\ F\ x)^r = (\forall\forall\ x.\ (F\ x)^r)$
proof −
have *1*: $\vdash (\forall\forall\ x.\ F\ x) = (\neg(\exists\exists\ x.\ \neg(F\ x)))$ **using** *AAxDef* **by** *blast*
have *2*: $\vdash (\forall\forall\ x.\ F\ x)^r = (\neg(\exists\exists\ x.\ \neg(F\ x)))^r$ **using** *REqvRule 1* **by** *blast*
have *3*: $\vdash (\neg(\exists\exists\ x.\ \neg(F\ x)))^r = (\neg((\exists\exists\ x.(\neg\ (F\ x)))^r))$ **by** (*simp add*: *rev-fun1*)
have *4*: $\vdash ((\exists\exists\ x.(\neg\ (F\ x)))^r) = ((\exists\exists\ x.(\neg\ (F\ x))^r))$ **by** (*simp add*: *EExRev*)
hence *5*: $\vdash (\neg((\exists\exists\ x.(\neg\ (F\ x)))^r)) = (\neg(\exists\exists\ x.(\neg\ (F\ x))^r))$ **by** *auto*
have *51*: $\bigwedge x.\ \vdash (\neg\ (F\ x))^r = (\neg(\ (F\ x)^r))$ **by** (*simp add*: *rev-fun1*)

hence *52*: $\vdash (\exists\exists\ x.(\neg\ (F\ x))^r) = (\exists\exists\ x.\neg(\ (F\ x)^r))$ **using** *EExEqvRule* **by** *fastforce*
hence *6*: $\vdash (\neg(\exists\exists\ x.(\neg\ (F\ x))^r)) = (\neg(\exists\exists\ x.\neg(\ (F\ x)^r)))$ **by** *fastforce*
have *7*: $\vdash (\neg(\exists\exists\ x.\neg(\ (F\ x)^r))) = (\forall\forall\ x.\ (F\ x)^r)$ **using** *AAxDef* **by** *fastforce*
from *1 2 3 5 6 7* **show** *?thesis* **by** *fastforce*
qed

end

10 Projection operator

theory *Projection*
imports *Fuse TimeReversal*
begin

This theory introduces the projection operator [4]. The projection operator is defined and we prove the soundness of the rules and axiom system.

10.1 Definitions

primrec *filt* :: $'a\ interval \Rightarrow index \Rightarrow\ 'a\ interval$
where $filt\ \sigma\ \langle l\rangle = \langle nth\ \sigma\ l\rangle$
$\mid filt\ \sigma\ (x \odot ls) = (nth\ \sigma\ x) \odot filt\ \sigma\ ls$

primrec *lsum* :: $'a\ interval\ interval \Rightarrow nat \Rightarrow index$
where $lsum\ \langle xs\rangle\ a = \langle a+(intlen\ xs)\rangle$
$\mid lsum\ (xs \odot xxs)\ a = (a+(intlen\ xs)) \odot (lsum\ xxs\ (a+(intlen\ xs)))$

definition *addzero* :: $index \Rightarrow index$
where $addzero\ ls = (if\ intlen\ ls = 0\ then$
$(if\ intfirst\ ls = 0\ then\ ls\ else\ 0 \odot ls)\ else\ 0 \odot ls)$

definition *powerinterval* :: $('a{::}world)\ formula \Rightarrow\ 'a\ interval \Rightarrow index \Rightarrow bool$
where $powerinterval\ F\ \sigma\ l = (\forall\ i.\ i<intlen\ l \longrightarrow ((sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models F))$

definition *cpl* :: $('a{::}world)\ formula \Rightarrow\ 'a\ formula \Rightarrow\ 'a\ interval \Rightarrow index$
where $cpl\ f\ g\ \sigma = (\epsilon\ l.\ index\text{-}sequence\ 0\ l \wedge powerinterval\ f\ \sigma\ l \wedge$
$(nth\ l\ (intlen\ l)) = intlen\ \sigma \wedge ((filt\ \sigma\ l) \models g))$

primrec *lcpl* :: $('a\ {::}world)\ formula \Rightarrow\ 'a\ formula \Rightarrow\ 'a\ interval \Rightarrow index \Rightarrow nat\ interval\ interval$
where $lcpl\ \ f\ g\ \sigma\ \langle x\rangle = \langle(map\ (shift\ x)\ (cpl\ f\ g\ (sub\ x\ x\ \sigma)))\rangle$
$\mid\ lcpl\ \ f\ g\ \sigma\ (x \odot xs) =$
$(case\ xs\ of\ \langle y\rangle \Rightarrow \langle(map\ (shift\ x)\ (cpl\ f\ g\ (sub\ x\ y\ \sigma)))\rangle$
$\mid y \odot ys \Rightarrow (map\ (shift\ x)\ (cpl\ f\ g\ (sub\ x\ y\ \sigma))) \odot (lcpl\ f\ g\ \sigma\ xs))$

definition *projection-d* :: $('a{::}\ world)\ formula \Rightarrow\ 'a\ formula \Rightarrow\ 'a\ formula$
where $projection\text{-}d\ F\ G \equiv \lambda s.\ (\exists\ l.\ index\text{-}sequence\ 0\ l\ \wedge (nth\ l\ (intlen\ l)) = (intlen\ s) \wedge$
$powerinterval\ F\ s\ l \wedge ((filt\ s\ l) \models G)$
$)$

syntax
-projection-d :: [*lift*,*lift*] ⇒ *lift* ((- △ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-projection-d :: [*lift*,*lift*] ⇒ *lift* ((- *proj* -) [*84*,*84*] *83*)

translations
-projection-d ⇌ *CONST projection-d*

definition *uprojection-d* :: (*'a*:: *world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *uprojection-d F G* ≡ *LIFT*(¬(*F* △ (¬ *G*)))

syntax
-uprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- ∇ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-uprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *uproj* -) [*84*,*84*] *83*)

translations
-uprojection-d ⇌ *CONST uprojection-d*

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

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

syntax
-dp-d :: *lift* ⇒ *lift* ((*dp* -) [*88*] *87*)
-bp-d :: *lift* ⇒ *lift* ((*bp* -) [*88*] *87*)

syntax (*ASCII*)
-dp-d :: *lift* ⇒ *lift* ((*dp* -) [*88*] *87*)
-bp-d :: *lift* ⇒ *lift* ((*bp* -) [*88*] *87*)

translations
-dp-d ⇌ *CONST dp-d*
-bp-d ⇌ *CONST bp-d*

10.2 Lemmas

10.2.1 filt Lemmas

lemma *filt-intlen*:
intlen(*filt s0 l*) = *intlen l*
by (*induct l*, *simp*, *simp*)

lemma *filt-nth*:
assumes *i* ≤ *intlen* (*filt s0 l*)

shows (*nth* (*filt s0 l*) *i*) = (*nth s0* (*nth l i*))
using *assms*
proof (*induct l arbitrary*: *i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a l*)
then show *?case* **by** *simp* (*metis Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv*)
qed

lemma *filt-expand*:
(*s1* = (*filt s0 l*)) =
(*intlen s1* = *intlen l* ∧
(∀ *i* ≤ *intlen s1*. (*nth s1 i*) = (*nth s0* (*nth l i*))))
by (*metis filt-intlen filt-nth interval-eq-nth-eq*)

lemma *filt-fuse*:
filt xs (*fuse l1 l2*) = (*fuse* (*filt xs l1*) (*filt xs l2*))
by (*induct l1 arbitrary*: *l2 xs*) *simp-all*

lemma *fuse-filt-intlen*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen xs*
shows *intlen* (*filt* (*fuse* (*prefix* (*nth l n*) *xs*) (*suffix* (*nth l n*) *xs*)) *l*) =
intlen (*fuse* (*filt* (*prefix* (*nth l n*) *xs*) (*prefix n l*))
(*filt* (*suffix* (*nth l n*) *xs*) (*map* (*shiftm* (*nth l n*)) (*suffix n l*))))
proof −
have *1*: *intlen* (*filt* (*fuse* (*prefix* (*nth l n*) *xs*) (*suffix* (*nth l n*) *xs*)) *l*) = *intlen l*
by (*simp add*: *filt-intlen*)
have *2*: *intlen* (*fuse* (*filt* (*prefix* (*nth l n*) *xs*) (*prefix n l*))
(*filt* (*suffix* (*nth l n*) *xs*) (*map* (*shiftm* (*nth l n*)) (*suffix n l*)))) =
(*intlen*((*filt* (*prefix* (*nth l n*) *xs*) (*prefix n l*))) +
intlen(*filt* (*suffix* (*nth l n*) *xs*) (*map* (*shiftm* (*nth l n*)) (*suffix n l*))))
using *interval-fuse-intlen-a* **by** *blast*
have *3*: *intlen*((*filt* (*prefix* (*nth l n*) *xs*) (*prefix n l*))) = *intlen*(*prefix n l*)
using *filt-intlen* **by** *blast*
have *4*: *intlen*(*filt* (*suffix* (*nth l n*) *xs*) (*map* (*shiftm* (*nth l n*)) (*suffix n l*))) =
intlen(*suffix n l*)
by (*simp add*: *filt-intlen*)
have *5*: *intlen*(*prefix n l*) + *intlen*(*suffix n l*) = *intlen l*
by (*simp*)
from *1 2 3 4 5* **show** *?thesis* **by** *auto*
qed

lemma *fuse-filt-nth-a*:
assumes *index-sequence 0 l*

```
        (nth l (intlen l)) = intlen xs
        i ≤ intlen l
        n ≤ intlen l
 shows    nth (filt (fuse (prefix (nth l n) xs) (suffix (nth l n) xs)) l) i =
          nth xs (nth l i)
proof −
  have 1: filt (fuse (prefix (nth l n) xs) (suffix (nth l n) xs)) l =
          filt xs l
     using assms by (metis interval-fuse-prefix-suffix interval-idx-less-last-1 le-neq-implies-less
        less-imp-le-nat not-less)
  have 2: nth (filt xs l) i = nth xs (nth l i)
     using assms by (metis filt-nth filt-intlen  )
  from 1 2 show ?thesis by auto
qed
```

```
lemma fuse-filt-nth-b:
 assumes index-sequence 0 l
        (nth l (intlen l)) = intlen xs
        i ≤ intlen l
        n ≤ intlen l
 shows   nth (fuse (filt (prefix (nth l n) xs) (prefix n l))
                (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) ))) i =
         nth xs (nth l i)
proof −
 have 1: i ≤ intlen(fuse (filt (prefix (nth l n) xs) (prefix n l))
                (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) )))
   using assms
   by (metis filt-intlen interval-fuse-intlen-a interval-fuse-prefix-suffix-intlen
      interval-intlen-map)
 have 2:  intlast(filt (prefix (nth l n) xs) (prefix n l)) = nth xs (nth l n)
  using assms filt-nth[of - (prefix (nth l n) xs) (prefix n l) ]
  by (metis filt-intlen interval-idx-less-equal interval-intlast-prefix interval-nth-intlen-intlast
     interval-prefix-length-good order-refl)
 have 3: intfirst(filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) )) =
         nth (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) )) 0
   by simp
 have 4: nth (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) )) 0 =
         nth (suffix (nth l n) xs) (nth (map (shiftm (nth l n)) (suffix n l) ) 0)
        using filt-nth by blast
 have 5: nth (suffix (nth l n) xs) (nth (map (shiftm (nth l n)) (suffix n l) ) 0) =
         nth  (suffix (nth l n) xs) (nth l (n+0) − (nth l n))
   by (simp add: assms interval-nth-map shiftm-def )
 have 6: nth  (suffix (nth l n) xs) (nth l (n+0) − (nth l n)) =
         nth  (suffix (nth l n) xs) 0
   by simp
 have 7: nth  (suffix (nth l n) xs) 0 = nth xs (nth l n)
   using assms by (metis Nat.add-0-right interval-nth-suffix le0  )
 have 8: intlast(filt (prefix (nth l n) xs) (prefix n l)) =
         intfirst(filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) ))
```

```
  using 2 4 5 7 by auto
 have 10: nth (fuse (filt (prefix (nth l n) xs) (prefix n l))
                (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) ))) i =
          (if i ≤ intlen (filt (prefix (nth l n) xs) (prefix n l))
           then nth (filt (prefix (nth l n) xs) (prefix n l)) i
           else nth (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) ))
                 (i − intlen (filt (prefix (nth l n) xs) (prefix n l))))
   using 1 8 interval-fuse-nth by auto
 have 11: intlen (filt (prefix (nth l n) xs) (prefix n l)) = n
   using assms by (metis filt-intlen interval-prefix-length-good)
 have 12: i≤n ⟶ nth (filt (prefix (nth l n) xs) (prefix n l)) i =
                nth (prefix (nth l n) xs) (nth (prefix n l ) i)
   by (simp add: 11 filt-nth)
 have 13: i≤n ⟶ nth (prefix (nth l n) xs) (nth (prefix n l ) i) =
                nth (prefix (nth l n) xs) (nth l i)
   by (simp add: assms)
 have 15:  i≤n ⟶ nth (prefix (nth l n) xs) (nth l i) = nth xs (nth l i)
       using assms(1) assms(2) assms(4) interval-idx-less-equal interval-nth-prefix by blast
 have 16: nth (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) ))
                 (i − intlen (filt (prefix (nth l n) xs) (prefix n l)))  =
           nth (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) ))
                 (i − n)
   by (simp add: 11)
 have 17: nth (filt (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) )) (i − n) =
           nth (suffix (nth l n) xs) (nth (map (shiftm (nth l n)) (suffix n l) ) (i −n))
   using assms filt-nth[of i−n (suffix (nth l n) xs) (map (shiftm (nth l n)) (suffix n l) ) ]
     by (simp add: filt-intlen)
 have 18: i> n ⟶ nth (map (shiftm (nth l n)) (suffix n l) ) (i −n) =
           nth l (n+(i−n)) − (nth l n)
   using assms
   by (simp add: interval-idx-shiftm-suffix-nth)
 have 19: i>n ⟶ nth (suffix (nth l n) xs) (nth (map (shiftm (nth l n)) (suffix n l) ) (i −n))
                = nth (suffix (nth l n) xs) ((nth l i) − (nth l n))
   by (simp add: 18)
 have 20 : i>n ⟶ nth (suffix (nth l n) xs) ((nth l i) − (nth l n)) =
                 nth xs ((nth l n) + ((nth l i) − (nth l n)))
   using assms by (metis diff-le-mono interval-idx-less-last-1 interval-nth-suffix
       le-neq-implies-less less-imp-le-nat nat-le-linear)
 have 22: i>n ⟶ (nth l n) + ((nth l i) − (nth l n)) =  (nth l i)
   using assms using interval-idx-less-equal by fastforce
 have 23: i>n ⟶ nth xs ((nth l n) + ((nth l i) − (nth l n))) = nth xs (nth l i)
   by (simp add: 22)
 from 10 show ?thesis by (simp add: 11 12 13 15 17 19 20 23)
qed

lemma fuse-filt-nth:
assumes index-sequence 0 l
        (nth l (intlen l)) = intlen xs
        i ≤ intlen l
        n ≤ intlen l
```

shows *nth* (*filt* (*fuse* (*prefix* (*nth l n*) *xs*) (*suffix* (*nth l n*) *xs*)) *l*) *i* =
nth (*fuse* (*filt* (*prefix* (*nth l n*) *xs*) (*prefix n l*))
(*filt* (*suffix* (*nth l n*) *xs*) (*map* (*shiftm* (*nth l n*)) (*suffix n l*)))) *i*
using *assms fuse-filt-nth-a*[*of l xs i n*]
fuse-filt-nth-b[*of l xs i n*] **by** *simp*

lemma *fuse-filt*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen xs*
n ≤ *intlen l*
shows *filt* (*fuse* (*prefix* (*nth l n*) *xs*) (*suffix* (*nth l n*) *xs*)) *l* =
fuse (*filt* (*prefix* (*nth l n*) *xs*) (*prefix n l*))
(*filt* (*suffix* (*nth l n*) *xs*) (*map* (*shiftm* (*nth l n*)) (*suffix n l*)))
using *assms fuse-filt-intlen fuse-filt-nth interval-eq-nth-eq* **by** (*metis filt-intlen*)

lemma *fuse-filt-a*:
assumes *index-sequence 0 l1*
index-sequence (*intlast l1*) *l2*
intlast l2 = *intlen xs*
shows *filt* (*fuse* (*prefix* (*intlast l1*) *xs*) (*suffix* (*intfirst l2*) *xs*)) (*fuse l1 l2*) =
fuse (*filt* (*prefix* (*intlast l1*) *xs*) *l1*)
(*filt* (*suffix* (*intfirst l2*) *xs*) (*map* (*shiftm* (*intfirst l2*)) *l2*))
proof −
have *1*: *intlast l1* = *intfirst l2*
using *assms* **by** (*metis index-sequence-def interval-nth-zero-intfirst*)
have *2*: *index-sequence 0* (*fuse l1 l2*)
using *assms* **by** (*metis index-sequence-def interval-idx-fuse interval-nth-zero-intfirst*)
have *3*: *intlast*(*fuse l1 l2*) = *intlen xs*
using *assms* **by** (*metis 1 add.left-neutral interval-fuse-nth-a interval-fuse-intlen-a*
interval-nth-intlen-intlast le-add2)
have *4*: *intlen l1* ≤ *intlen* (*fuse l1 l2*)
by (*simp add*: *interval-fuse-intlen-a*)
have *5*: *filt* (*fuse* (*prefix* (*nth* (*fuse l1 l2*) (*intlen l1*)) *xs*)
(*suffix* (*nth* (*fuse l1 l2*) (*intlen l1*)) *xs*))
(*fuse l1 l2*)
=
fuse (*filt* (*prefix* (*nth* (*fuse l1 l2*) (*intlen l1*)) *xs*) (*prefix* (*intlen l1*) (*fuse l1 l2*)))
(*filt* (*suffix* (*nth* (*fuse l1 l2*) (*intlen l1*)) *xs*)
(*map* (*shiftm* (*nth* (*fuse l1 l2*) (*intlen l1*))) (*suffix* (*intlen l1*) (*fuse l1 l2*))))
using *2 3 4 fuse-filt* **by** *auto*
have *6*: *filt* (*fuse* (*prefix* (*intlast l1*) *xs*) (*suffix* (*intfirst l2*) *xs*)) (*fuse l1 l2*) =
filt (*fuse* (*prefix* (*nth* (*fuse l1 l2*) (*intlen l1*)) *xs*)
(*suffix* (*nth* (*fuse l1 l2*) (*intlen l1*)) *xs*))
(*fuse l1 l2*)
using *1 4 interval-intlast-prefix interval-prefix-fuse* **by** *fastforce*
have *7*: (*filt* (*prefix* (*intlast l1*) *xs*) *l1*) =
(*filt* (*prefix* (*nth* (*fuse l1 l2*) (*intlen l1*)) *xs*) (*prefix* (*intlen l1*) (*fuse l1 l2*)))
using *1 4 interval-intlast-prefix interval-prefix-fuse* **by** *fastforce*
have *8*: (*filt* (*suffix* (*intfirst l2*) *xs*) (*map* (*shiftm* (*intfirst l2*)) *l2*)) =

(filt (suffix (nth (fuse l1 l2) (intlen l1)) xs)
(map (shiftm (nth (fuse l1 l2) (intlen l1))) (suffix (intlen l1) (fuse l1 l2))))
using *1 4 interval-intlast-prefix interval-prefix-fuse interval-suffix-fuse* **by** *metis*
show *?thesis* **using** *5 6 7 8* **by** *auto*
qed

lemma *filt-prefix*:
assumes $n \leq$ *intlen l*
shows *prefix n (filt σ l) = filt σ (prefix n l)*
proof −
have *1*: *intlen (prefix n (filt σ l)) = intlen (filt σ (prefix n l))*
by *(simp add: assms filt-intlen)*
have *2*: ∀ *i≤intlen(prefix n (filt σ l)).*
(nth (prefix n (filt σ l)) i) = (nth (filt σ (prefix n l)) i)
by *(simp add: assms filt-nth filt-intlen le-trans)*
show *?thesis* **using** *1 2 interval-eq-nth-eq* **by** *blast*
qed

lemma *filt-prefix-idx*:
assumes $n \leq$ *intlen σ*
index-sequence 0 l
nth l (intlen l) = n
shows *filt σ l = filt (prefix n σ) l*
proof −
have *1*: *intlen(filt σ l) = intlen (filt (prefix n σ) l)*
by *(simp add: filt-intlen)*
have *2*: ∀ *i≤intlen(filt σ l). (nth (filt σ l) i) = (nth (filt (prefix n σ) l) i)*
by *(metis assms filt-nth filt-intlen interval-idx-less-equal*
interval-nth-prefix interval-prefix-length-good order-refl)
show *?thesis*
by *(simp add: 1 2 interval-eq-nth-eq)*
qed

lemma *filt-suffix*:
assumes $n \leq$ *intlen l*
shows *suffix n (filt σ l) = filt σ (suffix n l)*
proof −
have *1*: *intlen (suffix n (filt σ l)) = intlen (filt σ (suffix n l))*
by *(simp add: assms filt-intlen)*
have *2*: ∀ *i≤intlen (suffix n (filt σ l)).*
(nth (suffix n (filt σ l)) i) = (nth (filt σ (suffix n l)) i)
by *(simp add: assms filt-nth filt-intlen*
ordered-cancel-comm-monoid-diff-class.le-diff-conv2)
show *?thesis* **using** *1 2 interval-eq-nth-eq* **by** *blast*
qed

lemma *filt-suffix-idx-intlen*:

assumes $n \leq$ *intlen* σ
index-sequence 0 l
nth l (*intlen l*) = *intlen* σ − *n*
shows *intlen* (*filt* σ (*map* (*shift n*) *l*)) = *intlen* (*filt* (*suffix n* σ) *l*)
using *assms* **by** (*simp add*: *filt-intlen*)

lemma *filt-suffix-idx-nth*:
assumes $n \leq$ *intlen* σ
index-sequence 0 l
i≤*intlen l*
nth l (*intlen l*) = *intlen* σ − *n*
shows *nth* (*filt* σ (*map* (*shift n*) *l*)) *i* = *nth* (*filt* (*suffix n* σ) *l*) *i*
proof −
have *1*: *nth* (*filt* σ (*map* (*shift n*) *l*)) *i* = *nth* σ (*nth* (*map* (*shift n*) *l*) *i*)
by (*simp add*: *assms filt-nth filt-intlen*)
have *2*: *nth* σ (*nth* (*map* (*shift n*) *l*) *i*) = (*nth* σ ((*nth l i*) + *n*))
by (*simp add*: *Interval.shift-def interval-nth-map*)
have *3*: *nth* (*filt* (*suffix n* σ) *l*) *i* = *nth* (*suffix n* σ) (*nth l i*)
by (*simp add*: *assms filt-nth filt-intlen*)
have *4*: *nth* (*suffix n* σ) (*nth l i*) = (*nth* σ ((*nth l i*) + *n*))
by (*metis add.commute assms eq-iff interval-idx-less-equal interval-nth-suffix*
interval-suffix-length-good)
show *?thesis* **by** (*simp add*: *1 2 3 4*)
qed

lemma *filt-suffix-idx*:
assumes $n \leq$ *intlen* σ
index-sequence 0 l
nth l (*intlen l*) = *intlen* σ − *n*
shows *filt* σ (*map* (*shift n*) *l*) = *filt* (*suffix n* σ) *l*
using *assms filt-suffix-idx-intlen filt-suffix-idx-nth interval-eq-nth-eq*
by (*simp add*: *filt-suffix-idx-nth interval-eq-nth-eq filt-intlen*)

lemma *filt-filt*:
(*nth* (*filt* (*filt xxs l1*) *l2*) *k*) =
(*nth xxs* (*nth l1* (*nth l2 k*)))
by (*metis filt-expand interval-intlen-map interval-nth-map*)

lemma *filt-filt-map*:
(*filt* (*filt xxs l1*) *l2*) = (*filt xxs* (*map* (λ *x*. (*nth l1 x*)) *l2*))
by (*metis filt-expand interval-intlen-map interval-nth-map*)

lemma *filt-map*:
filt xs l = *map* (λ*x*. *nth xs x*) *l*
by (*metis filt-expand interval-intlen-map interval-nth-map*)

lemma *filt-map-filt*:
(*filt* (*filt xxs l1*) *l2*) = *filt xxs* (*filt l1 l2*)
by (*metis filt-filt-map filt-map*)

lemma *filt-sub*:
 assumes $k \leq n$
  $n \leq intlen\ l$
 shows $(sub\ k\ n\ (filt\ \sigma\ l)) = (filt\ \sigma\ (sub\ k\ n\ l))$
using *assms*
by (*simp add*: *sub-def filt-prefix filt-suffix*)

lemma *filt-lfuse-map*:
 $(filt\ \sigma\ (lfuse\ (xxs))) =$
 $(lfuse\ (map\ (\lambda\ xs\ .\ (filt\ \sigma\ xs))\ xxs))$
proof (*induct xxs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xxs*)
then show *?case* **by** (*simp add*: *filt-fuse*)
qed

10.2.2 powerinterval lemmas

lemma *powerinterval-splita0*:
 assumes *index-sequence 0 l*
  $n \leq intlen\ l$
  $nth\ l\ (intlen\ l) = intlen\ \sigma$
  $i < intlen(prefix\ n\ l)$
 shows $(sub\ (nth\ (prefix\ n\ l)\ i)\ (nth\ (prefix\ n\ l)\ (Suc\ i))\ (prefix\ (nth\ l\ n)\ \sigma)) =$
  $(sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma)$

proof −
 have *01*: $(nth\ l\ n) \leq intlen\ \sigma$
  by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *interval-idx-less-last-1 le-less*)
 have *02*: $(nth\ (prefix\ n\ l)\ i) \leq (nth\ (prefix\ n\ l)\ (Suc\ i))$
  using *assms*(*1*) *assms*(*3*) *assms*(*4*) *interval-idx-expand* **by** *fastforce*
 have *03*: $(nth\ (prefix\ n\ l)\ (Suc\ i)) \leq (nth\ l\ n)$
  using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *interval-idx-less-equal* **by** *fastforce*
 have *1*: $intlen\ (sub\ (nth\ (prefix\ n\ l)\ i)\ (nth\ (prefix\ n\ l)\ (Suc\ i))\ (prefix\ (nth\ l\ n)\ \sigma)) =$
  $intlen(sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ (\sigma))$
  using *interval-intlen-sub*
  using *01 02 03 assms*(*4*) **by** *auto*
 have *2*: $(\forall\ j \leq intlen\ (sub\ (nth\ (prefix\ n\ l)\ i)\ (nth\ (prefix\ n\ l)\ (Suc\ i))\ (prefix\ (nth\ l\ n)\ \sigma)).$
  $(nth\ (sub\ (nth\ (prefix\ n\ l)\ i)\ (nth\ (prefix\ n\ l)\ (Suc\ i))\ (prefix\ (nth\ l\ n)\ \sigma))\ j) =$
  $(nth\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ (\sigma))\ j))$

  proof
   fix j
   show $j \leq intlen\ (sub\ (nth\ (prefix\ n\ l)\ i)\ (nth\ (prefix\ n\ l)\ (Suc\ i))\ (prefix\ (nth\ l\ n)\ \sigma)) \longrightarrow$
  $(nth\ (sub\ (nth\ (prefix\ n\ l)\ i)\ (nth\ (prefix\ n\ l)\ (Suc\ i))\ (prefix\ (nth\ l\ n)\ \sigma))\ j) =$
  $(nth\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ (\sigma))\ j)$

proof −
have *21*: *intlen* (*sub* (*nth* (*prefix n l*) *i*) (*nth* (*prefix n l*) (*Suc i*)) (*prefix* (*nth l n*) *σ*)) =
(*nth l* (*Suc i*)) − (*nth l i*)
using *1 assms*(*1*) *assms*(*3*) *assms*(*4*) *interval-idx-expand* **by** *fastforce*
have *22*: (*nth* (*prefix n l*) *i*) = (*nth l i*)
using *assms* **by** *auto*
have *23*: (*nth* (*prefix n l*) (*Suc i*)) = (*nth l* (*Suc i*))
using *assms* **by** *auto*
have *24*: *j* ≤ (*nth l* (*Suc i*)) − (*nth l i*) ⟶
nth (*sub* (*nth l i*) (*nth l* (*Suc i*)) *σ*) *j* = (*nth σ* ((*nth l i*)+*j*))
using *assms interval-idx-expand interval-nth-sub less-le-trans* **by** *fastforce*
have *25*: *j* ≤ (*nth l* (*Suc i*)) − (*nth l i*) ⟶ (*nth l* (*Suc i*)) ≤ *intlen* (*prefix* (*nth l n*) *σ*)
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *interval-idx-expand interval-idx-less-equal*
by *fastforce*
have *26*: *j* ≤ (*nth l* (*Suc i*)) − (*nth l i*) ⟶
(*nth* (*sub* (*nth* (*prefix n l*) *i*) (*nth* (*prefix n l*) (*Suc i*)) (*prefix* (*nth l n*) *σ*)) *j*) =
(*nth* (*prefix* (*nth l n*) *σ*) ((*nth l i*)+*j*))
using *25 assms interval-idx-expand interval-nth-sub less-le-trans* **by** *fastforce*
have *27*: (*nth l* (*Suc i*)) ≤(*nth l n*)
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *interval-idx-less-equal* **by** *fastforce*
have *28*: *j* ≤ (*nth l* (*Suc i*)) − (*nth l i*) ⟶ (*nth l i*)+*j*≤(*nth l n*)
using *27 assms*(*1*) *assms*(*3*) *assms*(*4*) *interval-idx-expand* **by** *fastforce*
have *29*: *j* ≤ (*nth l* (*Suc i*)) − (*nth l i*) ⟶
(*nth* (*prefix* (*nth l n*) *σ*) ((*nth l i*)+*j*)) = (*nth σ* ((*nth l i*)+*j*))
using *assms interval-nth-prefix* **using** *28* **by** *blast*
show *?thesis*
using *21 22 23 24 26 29* **by** *auto*
qed
qed
show *?thesis*
using *1 2 interval-eq-nth-eq* **by** *blast*
qed

lemma *powerinterval-splita*:
assumes *index-sequence 0 l*
n ≤ *intlen l*
nth l (*intlen l*) = *intlen σ*
powerinterval f σ l
shows *powerinterval f* (*prefix* (*nth l n*) *σ*) (*prefix n l*)
proof −
have *0*: (∀ *i*. *i*<*intlen l* ⟶ ((*sub* (*nth l i*) (*nth l* (*Suc i*)) *σ*) ⊨ *f*))
using *assms* **by** (*simp add*: *powerinterval-def*)
have *1*: *powerinterval f* (*prefix* (*nth l n*) *σ*) (*prefix n l*) =
(∀ *i*. *i* < *intlen*(*prefix n l*) ⟶
((*sub* (*nth* (*prefix n l*) *i*) (*nth* (*prefix n l*) (*Suc i*)) (*prefix* (*nth l n*) *σ*)) ⊨ *f*))
using *powerinterval-def* **by** *blast*
have *2*: (∀ *i*. *i* < *intlen*(*prefix n l*) ⟶
((*sub* (*nth* (*prefix n l*) *i*) (*nth* (*prefix n l*) (*Suc i*)) (*prefix* (*nth l n*) *σ*)) ⊨ *f*))
proof
fix *i*

```
    show i < intlen(prefix n l) ⟶
             ((sub (nth (prefix n l) i) (nth (prefix n l) (Suc i)) (prefix (nth l n) σ)) ⊨ f)
    proof −
     have 21:  i<n ⟶ ((sub (nth l i) (nth l (Suc i)) σ) ⊨ f)
       using 0 assms less-le-trans by blast
     have 22: i<intlen(prefix n l) ⟶
             ((sub (nth (prefix n l) i) (nth (prefix n l) (Suc i)) σ) ⊨ f)
       by (simp add: 21 assms)
     have 23: i<intlen(prefix n l) ⟶
             ((sub (nth (prefix n l) i) (nth (prefix n l) (Suc i)) (prefix (nth l n) σ)) ⊨ f)
      using 22 assms powerinterval-splita0 by fastforce
     show ?thesis using 23 by blast
    qed
   qed
  show ?thesis using 1 2 by blast
qed

lemma powerinterval-splitb0:
 assumes  index-sequence 0 l
          n ≤ intlen l
           nth l (intlen l) = intlen σ
           i < (intlen l −n)
 shows   (sub (nth (((map (shiftm (nth l n))  (suffix n l)))) i)
                 (nth (((map (shiftm (nth l n))  (suffix n l)))) (Suc i))
                 (suffix (nth l n) σ)) =
          (sub (nth l (i+n)) (nth l ((Suc i)+n)) σ)
proof −
 have 1: intlen(((map (shiftm (nth l n))  (suffix n l)))) = (intlen l) − n
     by (simp add: assms)
 have 2:  i<(intlen l −n) ⟶
             (nth (((map (shiftm (nth l n))  (suffix n l)))) i) =
              (nth l (n + i)) − (nth l n)
     using assms interval-idx-shiftm-suffix-nth by force
 have 3: i<(intlen l −n) ⟶
            (nth (((map (shiftm (nth l n))  (suffix n l)))) (Suc i)) =
            (nth l (n + (Suc i))) − (nth l n)
      using assms interval-idx-shiftm-suffix-nth by fastforce
 have 4: i<(intlen l −n) ⟶
            (sub (nth (((map (shiftm (nth l n))  (suffix n l)))) i)
                 (nth (((map (shiftm (nth l n))  (suffix n l)))) (Suc i))
                 (suffix (nth l n) σ)) =
              (sub ( (nth l (n + i)) − (nth l n) )
                   ( (nth l (n + (Suc i))) − (nth l n) )
                 (suffix (nth l n) σ))
         by (simp add: 2 3)
 have 5:  i<(intlen l −n) ⟶ (nth l (n + i))  < (nth l (n + (Suc i)))
  using assms index-sequence-def by auto
 have 6: i<(intlen l −n) ⟶ (nth l n) ≤ (nth l (n + i))
    using assms by (metis add.commute interval-idx-less-equal le-add1 less-diff-conv less-imp-le-nat)
 have 7: i<(intlen l −n) ⟶ (nth l n) ≤ (nth l (n + (Suc i)))
```

```
    using 5 6 by linarith
  have 8:  i<(intlen l −n) ⟶ (nth l (n + i)) − (nth l n) < (nth l (n + (Suc i))) − (nth l n)
    using 5 6 diff-less-mono by blast
  have 9: i<(intlen l −n) ⟶ (nth l (n + (Suc i))) − (nth l n) ≤ intlen σ − (nth l n)
   by (metis Suc-leI add.commute assms diff-le-mono interval-idx-less-last-1 le-eq-less-or-eq
          less-diff-conv ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
  have 10: i<(intlen l −n) ⟶
           (sub ( (nth l (n + i)) − (nth l n) )
                ( (nth l (n + (Suc i))) − (nth l n) )
               (suffix (nth l n) σ)) =
            (sub (nth l (i+n)) (nth l ((Suc i)+n)) σ)
    using interval-sub-suffix
    by (metis 6 7 8 9 add.commute le-add-diff-inverse2)
  show ?thesis
  using 2 3 10 assms by auto
qed

lemma powerinterval-splitb:
 assumes  index-sequence 0 l
          n ≤ intlen l
          nth l (intlen l) = intlen σ
         powerinterval f σ l
 shows    powerinterval f (suffix (nth l n) σ) ((map (shiftm (nth l n))  (suffix n l)))
proof −
 have 0: (∀ i. i<intlen l ⟶ ( (sub (nth l i) (nth l (Suc i)) σ) ⊨ f))
   using assms by (simp add: powerinterval-def)
 have 1: powerinterval f (suffix (nth l n) σ) ((map (shiftm (nth l n))  (suffix n l))) =
        (∀ i. i<intlen(((map (shiftm (nth l n))  (suffix n l)))) ⟶
          ( (sub (nth (((map (shiftm (nth l n))  (suffix n l)))) i)
                 (nth (((map (shiftm (nth l n))  (suffix n l)))) (Suc i))
                 (suffix (nth l n) σ)) ⊨ f))
    by (simp add: powerinterval-def)
 have 2: (∀ i. i<intlen(((map (shiftm (nth l n))  (suffix n l)))) ⟶
          ( (sub (nth (((map (shiftm (nth l n))  (suffix n l)))) i)
                 (nth (((map (shiftm (nth l n))  (suffix n l)))) (Suc i))
                 (suffix (nth l n) σ)) ⊨ f))
  proof
   fix i
   show i<intlen(((map (shiftm (nth l n))  (suffix n l)))) ⟶
          ( (sub (nth (((map (shiftm (nth l n))  (suffix n l)))) i)
                 (nth (((map (shiftm (nth l n))  (suffix n l)))) (Suc i))
                 (suffix (nth l n) σ)) ⊨ f)
   proof −
    have 21:  i< (intlen l) −n ⟶
            ((sub (nth l (i+n)) (nth l ((Suc i)+n)) σ) ⊨ f)
       by (simp add: 0)
    show ?thesis
    using 21 assms powerinterval-splitb0 by fastforce
   qed
  qed
```

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

lemma *powerinterval-split*:
assumes *index-sequence 0 l*
n $\leq$ *intlen l*
nth l (*intlen l*) = *intlen* σ
shows *powerinterval f* σ *l* =
(*powerinterval f* (*prefix* (*nth l n*) σ) (*prefix n l*) $\wedge$
powerinterval f (*suffix* (*nth l n*) σ) ((*map* (*shiftm* (*nth l n*)) (*suffix n l*))))
proof –
have *1*: *powerinterval f* σ *l* $\Longrightarrow$
powerinterval f (*prefix* (*nth l n*) σ) (*prefix n l*)
by (*simp add*: *assms powerinterval-splita*)
have *2*: *powerinterval f* σ *l* $\Longrightarrow$
powerinterval f (*suffix* (*nth l n*) σ) ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))
by (*simp add*: *assms powerinterval-splitb*)
have *3*: *powerinterval f* (*prefix* (*nth l n*) σ) (*prefix n l*) =
($\forall$ *i*. *i*<*n* $\longrightarrow$ ((*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) $\models$ *f*))
by (*metis assms interval-prefix-length-good powerinterval-def powerinterval-splita0*)
have *4*: *powerinterval f* (*suffix* (*nth l n*) σ) ((*map* (*shiftm* (*nth l n*)) (*suffix n l*))) =
($\forall$ *i*. *i*<(*intlen l*) –*n* $\longrightarrow$ ((*sub* (*nth l* (*i*+*n*)) (*nth l* ((*Suc i*)+*n*)) σ) $\models$ *f*))
by (*simp add*: *assms powerinterval-def powerinterval-splitb0*)
have *5*: ($\forall$ *i*. *i*<(*intlen l*) –*n* $\longrightarrow$ ((*sub* (*nth l* (*i*+*n*)) (*nth l* ((*Suc i*)+*n*)) σ) $\models$ *f*)) =
($\forall$ *i*. *n* $\leq$ *i* $\wedge$ *i*<(*intlen l*) $\longrightarrow$ ((*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) $\models$ *f*))
by (*metis le-add2 le-add-diff-inverse2 less-diff-conv plus-nat.simps*(*2*))
have *5*: (*powerinterval f* (*prefix* (*nth l n*) σ) (*prefix n l*) $\wedge$
powerinterval f (*suffix* (*nth l n*) σ) ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))) $\Longrightarrow$
powerinterval f σ *l*
using *3 4 5 assms powerinterval-def*
by (*metis not-less-eq not-less-less-Suc-eq order.order-iff-strict*)
show *?thesis*
using *1 2 5* **by** *blast*
qed

lemma *powerinterval-fuse*:
assumes *index-sequence 0 l1*
index-sequence 0 l2
intlast l1 = *cp*
cp $\leq$ *intlen* σ
intlast l2 = *intlen* σ – *cp*
shows *powerinterval f* σ (*fuse l1* (*map* (*shift cp*) *l2*)) =
(*powerinterval f* (*prefix cp* σ) *l1* $\wedge$
powerinterval f (*suffix cp* σ) *l2*)
proof –
have *1*: *index-sequence 0* (*fuse l1* (*map* (*shift cp*) *l2*))
using *assms interval-idx-fuse*[*of l1* (*map* (*shift cp*) *l2*)]
by (*metis index-sequence-def interval-idx-link interval-nth-zero-intfirst*)
have *2*: *cp* = (*nth l1* (*intlen l1*))
using *assms interval-nth-intlen-intlast* **by** *blast*

have *3*: *intlen l1* ≤ *intlen* (*fuse l1* (*map* (*shift cp*) *l2*))
by (*simp add*: *interval-fuse-intlen-a*)
have *4*: *nth* (*fuse l1* (*map* (*shift cp*) *l2*)) (*intlen* (*fuse l1* (*map* (*shift cp*) *l2*))) = *intlen σ*
by (*metis assms interval-idx-fuse-intlen interval-intlen-gr-zero interval-nth-intlen-intlast*)
have *5*: *cp* = *nth* (*fuse l1* (*map* (*shift cp*) *l2*)) (*intlen l1*)
by (*metis 3 assms interval-fuse-nth interval-idx-fuse-intfirst-intlast*
interval-nth-intlen-intlast order-refl)
have *6*: (*map* (*shiftm cp*) (*map* (*shift cp*) *l2*)) = *l2*
using *assms* **by** (*metis interval-idx-link interval-lsk-ls*)
have *7*: *intlast l1* = *intfirst* (*map* (*shift cp*) *l2*)
by (*metis assms interval-idx-fuse-intfirst-intlast interval-nth-intlen-intlast*)
show *?thesis*
using *1 3 4 5 6 7 interval-prefix-fuse interval-suffix-fuse powerinterval-split*
by *fastforce*
qed

lemma *powerinterval-idx*:
(*powerinterval* (*LIFT*(*f* ∧ *more*)) *σ l* ∧ (*nth l 0*) = *0* ∧ (*nth l* (*intlen l*)) = *intlen σ*) =
(*index-sequence 0 l* ∧ (*nth l* (*intlen l*)) = *intlen σ* ∧ *powerinterval f σ l*)
proof (*auto simp add*: *powerinterval-def index-sequence-def more-defs*)
show ⋀*n*. ∀ *i*<*intlen l*.
f (*sub* (*Interval.nth l i*) (*Interval.nth l* (*Suc i*)) *σ*) ∧
0 < *intlen* (*sub* (*Interval.nth l i*) (*Interval.nth l* (*Suc i*)) *σ*) ⟹
Interval.nth l 0 = *0* ⟹
Interval.nth l (*intlen l*) = *intlen σ* ⟹ *n* < *intlen l* ⟹
Interval.nth l n < *Interval.nth l* (*Suc n*)
by (*simp add*: *Interval.sub-def*)
show ⋀*i*. *Interval.nth l 0* = *0* ⟹
∀ *n*<*intlen l*. *Interval.nth l n* < *Interval.nth l* (*Suc n*) ⟹
Interval.nth l (*intlen l*) = *intlen σ* ⟹
∀ *i*<*intlen l*. *f* (*sub* (*Interval.nth l i*) (*Interval.nth l* (*Suc i*)) *σ*) ⟹
i < *intlen l* ⟹ *0* < *intlen* (*sub* (*Interval.nth l i*) (*Interval.nth l* (*Suc i*)) *σ*)
by (*simp add*: *Interval.sub-def*)
(*metis index-sequence-def interval-idx-less-last-1*)
qed

10.2.3 cpl lemmas

lemma *cpl-expand*:
assumes (∃ *l*. *index-sequence 0 l* ∧ *powerinterval f σ l* ∧ (*nth l* (*intlen l*)) = *intlen σ* ∧
((*filt σ l*) ⊨ *g*))
shows *index-sequence 0* (*cpl f g σ*) ∧ *powerinterval f σ* (*cpl f g σ*) ∧
(*nth* (*cpl f g σ*) (*intlen* (*cpl f g σ*))) = *intlen σ* ∧ ((*filt σ* (*cpl f g σ*)) ⊨ *g*)
proof −
have *0*: *cpl f g σ* = (*ϵ l*. *index-sequence 0 l* ∧ *powerinterval f σ l* ∧
(*nth l* (*intlen l*)) = *intlen σ* ∧ ((*filt σ l*) ⊨ *g*))
using *cpl-def* **by** *force*
have *1*: (∃ *l*. *index-sequence 0 l* ∧ *powerinterval f σ l* ∧ (*nth l* (*intlen l*)) = *intlen σ* ∧
((*filt σ l*) ⊨ *g*))

using *assms* **by** *auto*
have *2*: *index-sequence 0* $(cpl\ f\ g\ \sigma) \wedge$ *powerinterval f* $\sigma\ (cpl\ f\ g\ \sigma) \wedge$
$(nth\ (cpl\ f\ g\ \sigma)\ (intlen\ (cpl\ f\ g\ \sigma))) = intlen\ \sigma \wedge ((filt\ \sigma\ (cpl\ f\ g\ \sigma)) \models g)$
using *0 1*
somel-ex[*of* λl. *index-sequence 0 l* $\wedge$ *powerinterval f* σ *l* $\wedge$ $(nth\ l\ (intlen\ l)) = intlen\ \sigma \wedge$
$((filt\ \sigma\ l) \models g)$] **by** *simp*
show *?thesis*
using *2* **by** *blast*
qed

lemma *cpl-projection*:
$(\sigma \models f \bigtriangleup g) =$
(*index-sequence 0* $(cpl\ f\ g\ \sigma) \wedge$ *powerinterval f* $\sigma\ (cpl\ f\ g\ \sigma) \wedge$
$(nth\ (cpl\ f\ g\ \sigma)\ (intlen\ (cpl\ f\ g\ \sigma))) = intlen\ \sigma \wedge g\ (filt\ \sigma\ (cpl\ f\ g\ \sigma))$)
using *cpl-expand* **by** (*simp add*: *projection-d-def*, *blast*)

lemma *cpl-empty*:
assumes $intlen\ \sigma = 0 \wedge (\sigma \models f \bigtriangleup g)$
shows $(cpl\ f\ g\ \sigma) = \langle 0\rangle$
using *assms cpl-projection interval-idx-less-last-1 interval-st-intlen* **by** *fastforce*

lemma *cpl-empty-a*:
assumes $intlen\ \sigma = 0$
$(cpl\ f\ g\ \sigma) = \langle 0\rangle$
$g(filt\ \sigma\ \langle 0\rangle)$
shows $(\sigma \models f \bigtriangleup g)$
proof –
have *1*: *index-sequence 0* $\langle 0\rangle$
by (*simp add*: *index-sequence-def*)
have *2*: *powerinterval f* $\sigma\ \langle 0\rangle$
by (*simp add*: *powerinterval-def*)
have *3*: $(nth\ \langle 0\rangle\ (intlen\ \langle 0\rangle)) = intlen\ \sigma$
by (*simp add*: *assms*)
have *4*: $g(filt\ \sigma\ \langle 0\rangle)$
using *assms* **by** *blast*
from *1 2 3 4* **show** *?thesis*
by (*simp add*: *assms cpl-projection*)
qed

lemma *cpl-more*:
assumes $intlen\ \sigma > 0$
$(\sigma \models f \bigtriangleup g)$
shows $intlen(cpl\ f\ g\ \sigma) > 0$
by (*metis assms cpl-projection gr0I index-sequence-def*)

lemma *cpl-more-than-first*:
assumes $intlen\ \sigma > 0$
$(\sigma \models f \bigtriangleup g)$
shows $(nth\ (cpl\ f\ g\ \sigma)\ 0) = 0$

using *assms cpl-projection index-sequence-def* **by** *auto*

lemma *cpl-more-than-last*:
assumes *intlen* σ *>0*
$(\sigma \models f \triangle g)$
shows $(nth\ (cpl\ f\ g\ \sigma)\ (intlen\ (cpl\ f\ g\ \sigma))) = intlen\ \sigma$
using *assms cpl-projection* **by** *blast*

lemma *cpl-sub-more*:
assumes $n < k$
$k \leq intlen\ \sigma$
$((sub\ n\ k\ \sigma) \models f \triangle g)$
shows $intlen(cpl\ f\ g\ (sub\ n\ k\ \sigma)) > 0$
using *assms*
by (*simp add*: *cpl-more*)

lemma *cpl-bounds*:
assumes $n < k$
$k \leq intlen\ \sigma$
$((sub\ n\ k\ \sigma) \models f \triangle g)$
$i < intlen\ (cpl\ f\ g\ (sub\ n\ k\ \sigma))$
shows $0 \leq (nth\ (cpl\ f\ g\ (sub\ n\ k\ \sigma))\ i) \wedge (nth\ (cpl\ f\ g\ (sub\ n\ k\ \sigma))\ i) \leq k - n$
using *assms*
by (*metis cpl-projection interval-idx-less-last-1 interval-intlen-sub le0 less-imp-le-nat*)

lemma *cpl-map-bounds*:
assumes $n < k$
$k \leq intlen\ \sigma$
$((sub\ n\ k\ \sigma) \models f \triangle g)$
$i < intlen\ (map\ (shift\ n)\ (cpl\ f\ g\ (sub\ n\ k\ \sigma)))$
shows $n \leq (nth\ (map\ (shift\ n)\ (cpl\ f\ g\ (sub\ n\ k\ \sigma)))\ i) \wedge$
$(nth\ (map\ (shift\ n)\ (cpl\ f\ g\ (sub\ n\ k\ \sigma)))\ i) \leq k$
using *assms*
by (*metis Interval.shift-def Nat.le-diff-conv2 cpl-bounds interval-intlen-map*
interval-nth-map le-add2 less-imp-le-nat)

lemma *cpl-intfirst*:
assumes $(sub\ x1a\ (intfirst\ l)\ \sigma) \models f \triangle g$
shows $intfirst((map\ (shift\ x1a)\ (cpl\ f\ g\ (sub\ x1a\ (intfirst\ l)\ \sigma)))) = x1a$
proof −
have *1*: ($index\text{-}sequence\ 0\ (cpl\ f\ g\ (sub\ x1a\ (intfirst\ l)\ \sigma)) \wedge$
$powerinterval\ f\ (sub\ x1a\ (intfirst\ l)\ \sigma)\ (cpl\ f\ g\ (sub\ x1a\ (intfirst\ l)\ \sigma)) \wedge$
$(nth\ (cpl\ f\ g\ (sub\ x1a\ (intfirst\ l)\ \sigma))\ (intlen\ (cpl\ f\ g\ (sub\ x1a\ (intfirst\ l)\ \sigma)))) =$
$intlen\ (sub\ x1a\ (intfirst\ l)\ \sigma) \wedge$
$g\ (filt\ (sub\ x1a\ (intfirst\ l)\ \sigma)\ (cpl\ f\ g\ (sub\ x1a\ (intfirst\ l)\ \sigma)))$)
using *cpl-projection assms* **by** *auto*
have *2*: $index\text{-}sequence\ 0\ (cpl\ f\ g\ (sub\ x1a\ (intfirst\ l)\ \sigma))$
using *1* **by** *auto*
have *3*: $(nth\ (cpl\ f\ g\ (sub\ x1a\ (intfirst\ l)\ \sigma))\ 0) = 0$
using *2 index-sequence-def* **by** *blast*

show *?thesis*
by (*metis 3 Interval.shift-def add.left-neutral interval-nth-map interval-nth-zero-intfirst*)
qed

lemma *cpl-intfirst-same*:
assumes (*sub x1a x1a* σ) $\models$ *f* $\triangle$ *g*
shows *intfirst*((*map* (*shift x1a*) (*cpl f g* (*sub x1a x1a* σ)))) = *x1a*
proof −
have *1*: *intfirst* ($\langle$ *x1a* $\rangle$) = *x1a*
by *auto*
from *1 cpl-intfirst* **show** *?thesis* **by** (*metis assms*)
qed

lemma *cpl-intlast*:
assumes ((*sub x* (*intfirst l*) σ) $\models$ *f* $\triangle$ *g*) $\wedge$ *x* $<$ *intfirst l* $\wedge$ *intfirst l* $\leq$ *intlen* σ
shows *intlast*((*map* (*shift x*) (*cpl f g* (*sub x* (*intfirst l*) σ)))) = *intfirst l*
proof −
have *01*: (*index-sequence 0* (*cpl f g* (*sub x* (*intfirst l*) σ)) $\wedge$
powerinterval f (*sub x* (*intfirst l*) σ) (*cpl f g* (*sub x* (*intfirst l*) σ)) $\wedge$
(*nth* (*cpl f g* (*sub x* (*intfirst l*) σ)) (*intlen* (*cpl f g* (*sub x* (*intfirst l*) σ)))) =
intlen (*sub x* (*intfirst l*) σ) $\wedge$
g (*filt* (*sub x* (*intfirst l*) σ) (*cpl f g* (*sub x* (*intfirst l*) σ))))
using *cpl-projection assms* **by** (*simp add*: *cpl-projection*)
have *02*: (*nth* (*cpl f g* (*sub x* (*intfirst l*) σ)) (*intlen* (*cpl f g* (*sub x* (*intfirst l*) σ)))) =
intlen (*sub x* (*intfirst l*) σ)
using *01* **by** *auto*
have *03*: *intlast*(*cpl f g* (*sub x* (*intfirst l*) σ)) =
(*nth* (*cpl f g* (*sub x* (*intfirst l*) σ)) (*intlen* (*cpl f g* (*sub x* (*intfirst l*) σ))))
by *simp*
have *04*: *x* $<$ *intfirst l*
using *assms* **by** *blast*
have *05*: *intlen* (*sub x* (*intfirst l*) σ) = (*intfirst l*) −*x*
using *assms interval-intlen-sub less-imp-le-nat* **by** *blast*
have *06*: *intlast*((*map* (*shift x*) (*cpl f g* (*sub x* (*intfirst l*) σ)))) =
((*intfirst l*) −*x*)+*x*
by (*metis 02 05 Interval.shift-def interval-intlen-map interval-nth-intlen-intlast interval-nth-map*)
show *?thesis* **using** *04 06* **by** *auto*
qed

lemma *cpl-intlast-i*:
assumes ((*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) $\models$ *f* $\triangle$ *g*)
(*nth l i*) $<$ (*nth l* (*Suc i*))
(*nth l* (*Suc i*)) $\leq$ *intlen* σ
shows *intlast*((*map* (*shift* (*nth l i*)) (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)))) =
(*nth l* (*Suc i*))
proof −
have *01*: (*index-sequence 0* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) $\wedge$
powerinterval f (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)
(*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) $\wedge$

(*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))
(*intlen* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)))) =
intlen (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) $\wedge$
g (*filt* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))))
using *cpl-projection assms* **by** (*simp add*: *cpl-projection*)
have *02*: (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))
(*intlen* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)))) =
intlen (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)
using *01* **by** *auto*
have *03*: *intlast*(*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) =
(*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))
(*intlen* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))))
by *simp*
have *04*: (*nth l i*) < (*nth l* (*Suc i*))
by (*simp add*: *assms*)
have *05*: *intlen* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) = (*nth l* (*Suc i*)) −(*nth l i*)
by (*simp add*: *assms less-imp-le-nat*)
have *06*: *intlast*((*map* (*shift* (*nth l i*)) (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)))) =
((*nth l* (*Suc i*)) −(*nth l i*))+(*nth l i*)
by (*metis 02 05 Interval.shift-def interval-intlen-map interval-nth-intlen-intlast*
interval-nth-map)
show *?thesis* **using** *04 06* **by** *auto*
qed

10.2.4 lcpl lemmas

lemma *lcpl-nth*:
assumes *index-sequence* (*nth l 0*) *l*
i < *intlen l*
shows (*nth* (*lcpl f g* σ *l*) *i*) =
(*map* (*shift* (*nth l i*)) (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)))
using *assms*
proof (*induct l arbitrary*: *i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a ls*)
then show *?case*
proof (*cases i*)
case *0*
then show *?thesis*
proof (*cases ls*)
case (*St x1*)
then show *?thesis* **by** (*simp add*: *0*)
next
case (*Cons x21 x22*)
then show *?thesis* **by** (*simp add*: *0*)
qed
next
case (*Suc nat*)

```
   then show ?thesis
    proof (cases ls)
    case (St x1)
    then show ?thesis
    by (metis Cons.prems(2) One-nat-def Suc intlen.simps(1) intlen.simps(2) leD le-add1
         plus-1-eq-Suc)
    next
    case (Cons x21 x22)
    then show ?thesis
    using Cons.hyps Cons.prems(1) Cons.prems(2) Suc interval-idx-cons by auto
    qed
   qed
qed


lemma lcpl-intlen:
 assumes index-sequence (nth l 0) l
         intlen l > 0
 shows    intlen(lcpl f g σ l) = intlen l −1
using assms
 proof
  (induct l)
  case (St x)
  then show ?case by simp
  next
  case (Cons x1a l)
  then show ?case
   proof (cases l)
   case (St x1)
   then show ?thesis by simp
   next
   case (Cons x21 x22)
   then show ?thesis using Cons.hyps Cons.prems(1) interval-idx-cons by auto
   qed
qed

lemma lcpl-intlen-zero:
 assumes index-sequence (nth l 0) l
        intlen l = 0
 shows   intlen(lcpl f g σ l) = 0
using assms
by (metis interval-suffix-intlen interval-suffix-zero intlen.simps(1) lcpl.simps(1))

lemma lcpl-last:
 assumes index-sequence (nth l 0) l
        (nth l (intlen l)) = intlen σ
         intlen l >0
 shows    intlast (lcpl f g σ l) =
            (map (shift (nth l (intlen l−1)))
                (cpl f g (sub (nth l (intlen l −1)) (nth l (intlen l)) σ)))
```

```
proof -
 have 1: intlast (lcpl f g σ l) = (nth (lcpl f g σ l) (intlen (lcpl f g σ l)))
   by simp
 have 2: (nth (lcpl f g σ l) (intlen (lcpl f g σ l))) =
         (map (shift (nth l (intlen (lcpl f g σ l))))
              (cpl f g (sub (nth l (intlen (lcpl f g σ l)))
                          (nth l (Suc (intlen (lcpl f g σ l))))
                          σ)))
      using assms lcpl-nth
      by (metis One-nat-def Suc-pred diff-le-self lcpl-intlen le-eq-less-or-eq n-not-Suc-n )
 have 3: (map (shift (nth l (intlen (lcpl f g σ l))))
              (cpl f g (sub (nth l (intlen (lcpl f g σ l)))
                          (nth l (Suc (intlen (lcpl f g σ l))))
                          σ))) =
         (map (shift (nth l (intlen l−1)))
              (cpl f g (sub (nth l (intlen l−1)) (nth l (Suc (intlen l−1))) σ)))
        using assms by (metis lcpl-intlen )
 show ?thesis by (simp add: 2 3 assms(3))
qed

lemma lcpl-last-last:
 assumes index-sequence (nth l 0) l
        (nth l (intlen l)) = intlen σ
        ((sub (nth l (intlen l −1) ) (nth l (intlen l)) σ) |= f △ g)
         intlen l > 0
 shows    intlast (intlast (lcpl f g σ l)) = intlen σ
by (metis One-nat-def Suc-pred assms cpl-intlast-i diff-less interval-idx-less-last-1 lcpl-last
le-eq-less-or-eq zero-less-one)


lemma lcpl-zero-zero:
  assumes index-sequence (nth l 0) l
          (nth l (intlen l)) = intlen σ
          intlen l = 0
 shows   (nth (lcpl f g σ l) 0) =
         (map (shift (nth l 0)) (cpl f g (sub (nth l 0) (nth l 0) σ)))
using assms by (metis Interval.nth.simps(1) interval-suffix-intlen interval-suffix-zero lcpl.simps(1))

lemma lcpl-intfirst:
 assumes index-sequence (nth l 0) l
        (nth l (intlen l)) = intlen σ
        (∀ i<intlen l. (sub (nth l i) (nth l (Suc i)) σ) |= f △ g)
         intlen l > 0
 shows    intfirst(intfirst((lcpl f g σ l))) = intfirst l
proof -
 have 01: (intfirst((lcpl f g σ l))) =
          (map (shift (nth l 0)) (cpl f g (sub (nth l 0) (nth l (Suc 0)) σ)))
      using assms by (metis interval-nth-zero-intfirst lcpl-nth  )
 have 02: intlen l > 0 ⟶
          intfirst((map (shift (nth l 0)) (cpl f g (sub (nth l 0) (nth l (Suc 0)) σ)))) =
```

```
          (nth l 0)
          using assms by (metis Suc-leI cpl-intfirst interval-intlast-intfirst interval-intlast-prefix)
 show ?thesis
 using 01 02 interval-nth-zero-intfirst by (simp add: assms(4))
qed


lemma lcpl-lfuse-lastfirst:
 assumes index-sequence (nth l 0) l
        (nth l (intlen l)) = intlen σ
        ((intlen l = 0 ∧ ((sub (nth l 0) (nth l 0) σ) ⊨ f △ g)) ∨
          (intlen l > 0 ∧ (∀ i<intlen l. (sub (nth l i) (nth l (Suc i)) σ) ⊨ f △ g)))
 shows   lastfirst (lcpl f g σ l)
using assms
proof
  (induct l)
  case (St x)
  then show ?case by simp
  next
  case (Cons x1a ls)
  then show ?case
   proof −
    have 1: intlen ls = 0 ⟶ lastfirst (lcpl f g σ (x1a ⊙ ls))
          using Cons.prems by (metis One-nat-def add-diff-cancel-left' interval-st-intlen
              intlen.simps(2) lastfirst.simps(1) lcpl-intlen plus-1-eq-Suc zero-less-one)
    have 2: intlen ls > 0 ⟶ lastfirst ( lcpl f g σ (x1a ⊙ ls) ) =
              lastfirst((map (shift x1a) (cpl f g (sub x1a (intfirst ls) σ))) ⊙ (lcpl f g σ ls))
    by (metis (no-types, lifting) interval.simps(6) interval-intlen-cons-1 interval-nth-zero
        interval-nth-zero-intfirst lcpl.simps(2))
    have 3: intlen ls > 0 ⟶
            lastfirst((map (shift x1a) (cpl f g (sub x1a (intfirst ls) σ))) ⊙ (lcpl f g σ ls)) =
             ( intlast( (map (shift x1a) (cpl f g (sub x1a (intfirst ls) σ))) ) =
              intfirst(intfirst((lcpl f g σ ls))) ∧ lastfirst((lcpl f g σ ls)) )
      using lastfirst.simps(2) by blast
   have 4: x1a < (intfirst ls)
      using Cons.prems by (metis  index-sequence-def interval-nth-Suc interval-nth-zero
                interval-nth-zero-intfirst intlen.simps(2) plus-1-eq-Suc zero-less-Suc)
    have 5:  (intfirst ls) ≤ intlen σ
        using Cons.prems by (metis Suc-lessI eq-iff interval-idx-less-last-1
            interval-nth-Suc interval-nth-zero-intfirst intlen.simps(2) less-imp-le-nat
            plus-1-eq-Suc zero-less-Suc)
    have 6: intlen ls >0 ⟶
             intlast( (map (shift x1a) (cpl f g (sub x1a (intfirst ls) σ))) ) = intfirst ls
      using Cons.prems by (metis 4 5 cpl-intlast interval-nth-Suc interval-nth-zero
        interval-nth-zero-intfirst less-le-not-le)
    have 7: index-sequence (nth ls 0) ls
       using assms Cons.prems index-sequence-def by auto
    have 8: nth ls (intlen ls) = intlen σ
        using Cons.prems by auto
    have 9: ( (∀ i<intlen ( ls). (sub (nth ( ls) i) (nth ( ls) (Suc i)) σ) ⊨ f △ g))
```

using *Cons.prems* **by** *auto*
have *10*: *intlen ls >0* $\longrightarrow$ *intfirst(intfirst((lcpl f g* σ *ls)))* = *intfirst ls*
using *7 8 9*
by (*metis cpl-intfirst interval-intlen-cons-1 interval-nth-Suc interval-nth-zero-intfirst lcpl-nth*)
have *11*: *intlen ls >0* $\longrightarrow$
lastfirst((lcpl f g σ *ls))*
using *7 8 9 Cons.hyps* **by** *blast*
show *?thesis* **using** *1 10 11 2 6* **by** *auto*
qed
qed

lemma *lcpl-lfuse-intlen*:
assumes *index-sequence (nth l 0) l*
(nth l (intlen l)) = *intlen* σ
((intlen l =0 $\wedge$ *((sub (nth l 0) (nth l 0)* σ $\models$ *f* $\triangle$ *g)))* $\vee$
(intlen l >0 $\wedge$ ($\forall$ *i<intlen l. (sub (nth l i) (nth l (Suc i))* σ) $\models$ *f* $\triangle$ *g)))*
shows *(intlen l =0* $\longrightarrow$ *intlen(lfuse (lcpl f g* σ *l))* = *0)* $\wedge$
(intlen l >0 $\longrightarrow$
intlen(lfuse (lcpl f g σ *l))* = ($\sum$ *k=0..intlen l*−*1. intlen (nth (lcpl f g* σ *l) k)))*
proof −
have *1*: *lastfirst (lcpl f g* σ *l)*
using *assms lcpl-lfuse-lastfirst* **by** *blast*
have *2*: *intlen l* = *0* $\longrightarrow$
lfuse (lcpl f g σ *l)* =
((map (shift (nth l 0)) (cpl f g (sub (nth l 0) (nth l 0) σ*))))*
by (*metis interval-suffix-intlen interval-suffix-zero lcpl.simps(1) lfuse-St*)
have *3*: *intlen l* = *0* $\longrightarrow$
intlen ((map (shift (nth l 0)) (cpl f g (sub (nth l 0) (nth l 0) σ*))))* = *0*
using *assms cpl-empty interval-intlen-sub* **by** *fastforce*
have *4*: *intlen l* = *0* $\longrightarrow$ *intlen(lfuse (lcpl f g* σ *l))* = *0*
using *2 3* **by** *auto*
have *5*: *intlen l* > *0* $\longrightarrow$
intlen(lfuse (lcpl f g σ *l))* = ($\sum$ *k=0..intlen l*−*1. intlen (nth (lcpl f g* σ *l) k))*
using *interval-lfuse-intlen*
by (*metis assms lcpl-intlen lcpl-lfuse-lastfirst*)
from *4 5* **show** *?thesis* **by** *simp*
qed

lemma *lcpl-lfuse-idx*:
assumes *index-sequence 0 l*
(nth l (intlen l)) = *intlen* σ
($\forall$ *i<intlen l. (sub (nth l i) (nth l (Suc i))* σ) $\models$ *f* $\triangle$ *g)*
intlen l > *0*
shows *index-sequence (intfirst (lfuse (lcpl f g* σ *l))) (lfuse ((lcpl f g* σ *l)))*
proof −
have *0*: *intlen* σ *>0* $\longrightarrow$ *intlen l >0*
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *2*: *intlen* σ *>0* $\longrightarrow$ *lastfirst (lcpl f g* σ *l)*
using *0 assms index-sequence-def lcpl-lfuse-lastfirst* **by** *blast*

```
 have 3:  (∀ i<intlen l. index-sequence 0 (cpl f g (sub (nth l i) (nth l (Suc i)) σ)))
    using assms cpl-projection by auto
 have 4:  (∀ i<intlen l.
            index-sequence (nth l i)
                          (map (shift (nth l i)) (cpl f g (sub (nth l i) (nth l (Suc i)) σ))))
   using 3 interval-idx-link by blast
 have 5: intlen σ >0 ⟶
         (∀ i<intlen l. index-sequence (nth l i) (nth (lcpl f g σ l) i) )
   using assms by (metis 4 index-sequence-def lcpl-nth)
 have 6:  intlen σ >0 ⟶
        (∀  i<intlen l. intfirst (nth (lcpl f g σ l) i) = (nth l i))
   by (metis 5 index-sequence-def interval-nth-zero-intfirst)
 have 7:  intlen σ >0 ⟶
          (∀ i<intlen l.
            index-sequence (intfirst (nth (lcpl f g σ l) i)) (nth (lcpl f g σ l) i) )
    using 6 5 by simp
 have 8: intlen σ >0 ⟶
         index-sequence (intfirst (lfuse  (lcpl f g σ l)))  (lfuse ( (lcpl f g σ l)))
    using assms
    by (metis (mono-tags, lifting) 2 7 Suc-diff-1 index-sequence-def interval-idx-lfuse
        lcpl-intlen le-imp-less-Suc)
  from 8  show ?thesis
  by (metis assms(1) assms(2) assms(4) index-sequence-def interval-idx-less-last-1)
qed

lemma lcpl-intlen-nth-gr-zero:
 assumes index-sequence 0 l
        (nth l (intlen l)) = intlen σ
        (∀  i<intlen l. (sub (nth l i) (nth l (Suc i)) σ) ⊨ f △ g)
         intlen σ >0
 shows   (∀  j ≤ intlen (lcpl f g σ l). intlen(nth (lcpl f g σ l) j) > 0)
proof
  fix j
  show j ≤ intlen (lcpl f g σ l) ⟶ 0 < intlen (Interval.nth (lcpl f g σ l) j)
  proof −
  have 1: intlen σ>0 ⟶ lastfirst (lcpl f g σ l)
     by (metis assms index-sequence-def lcpl-lfuse-lastfirst neq0-conv)
  have 2: intlen σ>0 ⟶ intlen l >0
     using assms gr-zeroI index-sequence-def by fastforce
  have 3: intlen σ>0 ⟶ j ≤ intlen (lcpl f g σ l) ⟶
         (nth (lcpl f g σ l) j) =
           (map (shift (nth l j)) (cpl f g (sub (nth l j) (nth l (Suc j)) σ)))
    using assms
    by (metis 2 One-nat-def Suc-pred index-sequence-def lcpl-intlen lcpl-nth less-Suc-eq-le)
  have 4: intlen σ>0 ⟶ j ≤ intlen (lcpl f g σ l) ⟶
          intlen (map (shift (nth l j)) (cpl f g (sub (nth l j) (nth l (Suc j)) σ))) > 0
    by (metis (no-types, lifting) 2 One-nat-def Suc-pred add.commute assms cpl-sub-more gr0I
        index-sequence-def interval-idx-expand interval-intlen-map lcpl-intlen
        le-imp-less-Suc plus-1-eq-Suc)
  show ?thesis
```

using *3 4* **by** (*simp add*: *assms*(*4*))
qed
qed

lemma *lcpl-intlast-nth*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen* σ
($\forall$ *i*<*intlen l*. (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) $\models$ *f* $\triangle$ *g*)
intlen σ > *0*
j $\leq$ *intlen* (*lcpl f g* σ *l*)
shows *intlast*(*nth* (*lcpl f g* σ *l*) *j*) = (*nth l* (*Suc j*))
proof −
have *0*: *intlen* σ >*0* $\longrightarrow$ *intlen l* >*0*
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *1*: *intlen* σ>*0* $\longrightarrow$
j $\leq$ *intlen* (*lcpl f g* σ *l*) $\longrightarrow$
nth (*lcpl f g* σ *l*) *j* =
(*map* (*shift* (*nth l j*)) (*cpl f g* (*sub* (*nth l j*) (*nth l* (*Suc j*)) σ)))

using *assms lcpl-nth*[*of l j f g* σ]
proof −
show *?thesis*
by (*metis* (*no-types*) *0 Suc-diff-1*
$\langle$[[*index-sequence* (*nth l 0*) *l*; *j* < *intlen l*]] $\Longrightarrow$
nth (*lcpl f g* σ *l*) *j* =
map (*shift* (*Interval*.*nth l j*)) (*cpl f g* (*sub* (*nth l j*) (*Interval*.*nth l* (*Suc j*)) σ))$\rangle$
$\langle$*index-sequence 0 l*$\rangle$ *index-sequence-def lcpl-intlen le-imp-less-Suc*)
qed
have *2*: *intlen* σ>*0* $\longrightarrow$
j $\leq$ *intlen* (*lcpl f g* σ *l*) $\longrightarrow$ (*nth l* (*Suc j*)) $\leq$ *intlen* σ
using *assms*
by (*metis 0 Suc-diff-1 Suc-eq-plus1 index-sequence-def interval-idx-expand lcpl-intlen leD not-less-eq*)
have *2*: *intlen* σ>*0* $\longrightarrow$
j $\leq$ *intlen* (*lcpl f g* σ *l*) $\longrightarrow$
intlast (*map* (*shift* (*nth l j*)) (*cpl f g* (*sub* (*nth l j*) (*nth l* (*Suc j*)) σ))) =
(*nth l* (*Suc j*))
using *assms cpl-intlast-i*[*of f g l j* σ]
by (*metis* (*no-types*, *hide-lams*) *0 1 2 One-nat-def Suc-le-lessD Suc-pred index-sequence-def lcpl-intlen not-less-eq-eq*)
show *?thesis* **using** *1 2* **by** (*simp add*: *assms*(*4*) *assms*(*5*))
qed

lemma *lcpl-lfuse-filt-power-help*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen* σ
($\forall$ *i*<*intlen l*. (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) $\models$ *f* $\triangle$ *g*)

```
        intlen σ > 0
 shows   (∀ i<intlen l. g (filt σ (nth (lcpl f g σ l) i)) )
proof −
 have 1: (∀ i<intlen l. g (filt (sub (nth l i) (nth l (Suc i)) σ)
                           (cpl f g (sub (nth l i) (nth l (Suc i)) σ)) ))
   using assms cpl-projection by blast
 have 2: intlen σ >0 ⟶
         (∀ i<intlen l.
          (filt σ (nth (lcpl f g σ l) i)) =
          (filt σ (map (shift (nth l i)) (cpl f g (sub (nth l i) (nth l (Suc i)) σ)))))

   using assms by (metis index-sequence-def lcpl-nth)
 have 3: intlen σ >0 ⟶
         (∀ i<intlen l.
            intlen (filt σ (nth (lcpl f g σ l) i)) =
            intlen (filt (sub (nth l i) (nth l (Suc i)) σ)
                           (cpl f g (sub (nth l i) (nth l (Suc i)) σ)) )
         )

       by (simp add: 2 filt-intlen)
 have 4: intlen σ >0 ⟶
         (∀ i<intlen l.
          (∀ j≤ intlen (filt σ (nth (lcpl f g σ l) i)).
             (nth (filt σ (map (shift (nth l i)) (cpl f g (sub (nth l i) (nth l (Suc i)) σ)))) j)
              =
              (nth σ ((nth (cpl f g (sub (nth l i) (nth l (Suc i)) σ)) j) +(nth l i) ))
          )
        )
       using interval-nth-map shift-def filt-map by metis
 have 5: intlen σ >0 ⟶
         (∀ i<intlen l.
          (∀ j≤ intlen (filt σ (nth (lcpl f g σ l) i)).
             (nth (filt (sub (nth l i) (nth l (Suc i)) σ)
                           (cpl f g (sub (nth l i) (nth l (Suc i)) σ)) ) j) =
             (nth (sub (nth l i) (nth l (Suc i)) σ)
                  (nth (cpl f g (sub (nth l i) (nth l (Suc i)) σ)) j))
          ))
      by (simp add: filt-map interval-nth-map)
 have 6: intlen σ >0 ⟶
         (∀ i<intlen l.
             (nth l (Suc i)) ≤ intlen σ
         )
     using assms interval-idx-expand by auto
 have 7: intlen σ >0 ⟶
         (∀ i<intlen l.
              (nth l i) ≤ (nth l (Suc i))
         )
      using assms index-sequence-def less-imp-le by blast
 have 8: intlen σ >0 ⟶
         (∀ i<intlen l.
```

($\forall$ $j\leq$ *intlen* (*filt* σ (*nth* (*lcpl f g* σ *l*) *i*)).
(*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *j*) $\leq$ (*nth l* (*Suc i*)) $-$ (*nth l i*)
))
by (*metis* (*no-types*, *lifting*) *3 6 assms*(*1*) *assms*(*3*) *cpl-bounds cpl-projection*
filt-intlen index-sequence-def interval-intlen-sub le-eq-less-or-eq)
have *9*: *intlen* σ $>$*0* $\longrightarrow$
($\forall$ *i*$<$*intlen l*.
($\forall$ $j\leq$ *intlen* (*filt* σ (*nth* (*lcpl f g* σ *l*) *i*)).
(*nth* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)
(*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *j*)) $=$
(*nth* σ ((*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *j*) $+$ (*nth l i*)))
))
using *6*
by (*simp add*: *7 8 add.commute*)
have *10*: *intlen* σ $>$*0* $\longrightarrow$
($\forall$ *i*$<$*intlen l*.
($\forall$ $j\leq$ *intlen* (*filt* σ (*nth* (*lcpl f g* σ *l*) *i*)).
(*filt* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)
(*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))) $=$
(*filt* σ (*nth* (*lcpl f g* σ *l*) *i*))
))

by (*simp add*: *filt-expand*)
(*metis 2 3 4 9 filt-intlen filt-map interval-nth-map*)
show *?thesis* **using** *1 10 assms*(*4*) **by** *fastforce*
qed

10.2.5 lsum lemmas

lemma *lsum-state*:
lsum $\langle xs\rangle$ *a* $=$ $\langle a+intlen\ xs\rangle$
by *simp*

lemma *lsum-addzero-state*:
addzero (*lsum* $\langle xs\rangle$ *0*) $=$ (*if intlen xs* $=$ *0 then* $\langle 0\rangle$ *else* $\langle 0,\ intlen\ xs\rangle$)
by (*simp add*: *addzero-def*)

lemma *lsum-addzero-cons*:
addzero (*lsum* (*xs*$\odot$*xxs*) *0*) $=$ *0* $\odot$ (*lsum* (*xs*$\odot$*xxs*) *0*)
by (*simp add*: *addzero-def*)

lemma *lsum-intfirst*:
intfirst (*lsum xxs a*) $=$ *a*$+$*intlen*(*intfirst xxs*)
by (*case-tac xxs*) *simp-all*

lemma *lsum-intlen*:
intlen (*lsum xxs a*) $=$ *intlen xxs*
by (*induct xxs arbitrary*: *a*) *simp-all*

lemma *lsum-addzero-intfirst*:

intfirst (*addzero* (*lsum xxs 0*)) = *0*
by (*simp add*: *addzero-def lsum-intfirst*)

lemma *lsum-addzero-intlen*:
(*intlen xxs* = *0* ∧ *intlen*(*intfirst xxs*) = *0* ⟶
intlen (*addzero* (*lsum xxs 0*)) = *0*)
∧
(*intlen xxs* = *0* ∧ *intlen*(*intfirst xxs*) > *0* ⟶
intlen (*addzero* (*lsum xxs 0*)) = *1*)
∧
(*intlen xxs* > *0* ⟶
intlen (*addzero* (*lsum xxs 0*)) = (*intlen xxs*) +*1*)

by (*simp add*: *addzero-def*)
(*metis add.left-neutral interval-nth-zero-intfirst lsum-intfirst lsum-intlen*)

lemma *lsum-nth-help*:
assumes *i*>*0*
i ≤ *intlen xxs* +*1*
shows ($\sum$ *k* = *0*..(*i*−*1*). *intlen* (*Interval.nth* (*xxs*) *k*)) =
($\sum$ *k* = *1*..*i*. *intlen* (*Interval.nth* (*xxs*) (*k*−*1*)))
using *assms*
proof
(*induct i*)
case *0*
then show *?case* **by** *blast*
next
case (*Suc i*)
then show *?case*
proof *simp-all*
assume *a1*: *0* < *i* ⟹ ($\sum$ *k* = *0*..*i* − *Suc 0*. *intlen* (*Interval.nth xxs k*)) =
($\sum$ *k* = *Suc 0*..*i*. *intlen* (*Interval.nth xxs* (*k* − *Suc 0*)))
have *f2*: ∀ *f*. *sum f* {*1*::*nat*..*0*} = (*0*::*nat*)
by *auto*
have *f3*: ∀ *n f*. *sum f* {*n*::*nat*..*n*} = (*f n*::*nat*)
by *simp*
have *f4*: ∀ *f n*. (*sum f* {*0*::*nat*..*n* − *1*} + (*f n*::*nat*) = *sum f* {*0*..*n*} ∨ *0* = *n*) ∨ ¬ *0* ≤ *n*
by (*metis* (*no-types*) *One-nat-def Suc-pred le-eq-less-or-eq sum.atLeast0-atMost-Suc*)
have *f5*: ∀ *n*. (*0*::*nat*) ≤ *n*
by *blast*
have ∀ *n*. (*0*::*nat*) + *n* = *n*
by *linarith*
then show ($\sum$ *n* = *0*..*i*. *intlen* (*Interval.nth xxs n*)) =
($\sum$ *n* = *Suc 0*..*i*. *intlen* (*Interval.nth xxs* (*n* − *Suc 0*))) + *intlen* (*Interval.nth xxs i*)
using *f5 f4 f3 f2 a1* **by** (*metis One-nat-def le-eq-less-or-eq*)
qed
qed

lemma *lsum-nth*:

```
 assumes i≤ intlen xxs
 shows nth (lsum xxs a) i = a+(∑ k::nat= 0..i. intlen(nth xxs k))
using assms
proof
 (induct xxs arbitrary: a i)
 case (St x)
 then show ?case by simp
 next
 case (Cons x1a xxs)
 then show ?case
  proof −
   have 1: nth (lsum (x1a ⊙ xxs) a) i = nth ((a+intlen x1a)⊙ (lsum xxs (a+intlen x1a))) i
    by simp
   have 2: i ≤ intlen xxs +1
    using Cons.prems by auto
   have 3: i =0 ⟶
         nth ((a+intlen x1a)⊙ (lsum xxs (a+intlen x1a))) i = (a+intlen x1a)
     by simp
   have 4: a+intlen x1a = a + (∑ k = 0..0. intlen (Interval.nth (x1a ⊙ xxs) k))
    by simp
   have 5: i>0 ∧ i ≤ intlen xxs +1 ⟶
         nth ((a+intlen x1a)⊙ (lsum xxs (a+intlen x1a))) i =
          nth (lsum xxs (a+intlen x1a)) (i−1)
      by (metis One-nat-def Suc-pred interval-nth-Suc)
   have 6: i>0 ∧ i ≤ intlen xxs +1 ⟶
         nth (lsum xxs (a+intlen x1a)) (i−1) =
          (a+intlen x1a) + (∑ k = 0..(i−1). intlen (Interval.nth (xxs) k))
     using Cons.hyps le-diff-conv by blast
   have 7: i>0 ∧ i ≤ intlen xxs +1 ⟶
         (∑ k = 0..(i−1). intlen (Interval.nth (xxs) k)) =
         (∑ k = 1..i. intlen (Interval.nth (xxs) (k−1)))
      using lsum-nth-help by blast
   have 8: i>0 ∧ i ≤ intlen xxs +1 ⟶
          (∑ k = 1..i. intlen (Interval.nth (xxs) (k−1))) =
          (∑ k = 1..i. intlen (Interval.nth (x1a ⊙ xxs) (k)))

    by (metis (no-types, lifting) atLeastAtMost-iff interval-nth-Suc
        ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc sum.cong)
   have 9: i>0 ∧ i ≤ intlen xxs +1 ⟶
         (a+intlen x1a) + (∑ k = 1..i. intlen (Interval.nth (x1a ⊙ xxs) (k))) =
          a + (∑ k = 0..i. intlen (Interval.nth (x1a ⊙ xxs) (k)))
        by (simp add: sum.atLeast-Suc-atMost)
   show ?thesis
   by (metis 1 2 3 4 5 6 7 8 9 not-gr-zero)
  qed
qed
```

lemma *lsum-addzero-nth*:
 assumes *i≤ intlen (addzero (lsum xxs 0))*

shows (*intlen xxs* = *0* ∧ *intlen*(*intfirst xxs*) = *0* ⟶
nth (*addzero* (*lsum xxs 0*)) *i* = (*nth* (*lsum xxs 0*) *i*))
∧
(*intlen xxs* = *0* ∧ *intlen*(*intfirst xxs*) > *0* ⟶
nth (*addzero* (*lsum xxs 0*)) *i* = (*nth* (*0*⊙(*lsum xxs 0*)) *i*))
∧
(*intlen xxs* > *0* ⟶
nth (*addzero* (*lsum xxs 0*)) *i* = (*nth* (*0*⊙(*lsum xxs 0*)) *i*))

using *assms*
by (*metis add.left-neutral addzero-def less-numeral-extra*(*3*) *lsum-intfirst lsum-intlen*)

lemma *lsum-intlast*:
intlast (*lsum xxs a*) = *a*+(∑ *k*::*nat*= *0*..(*intlen xxs*). *intlen*(*nth xxs k*))
by (*metis interval-nth-intlen-intlast le-refl lsum-intlen lsum-nth*)

lemma *lsum-addzero-intlast*:
intlast (*addzero* (*lsum xxs 0*)) = *intlast*(*lsum xxs 0*)
by (*simp add*: *addzero-def*)

lemma *lsum-nth-leq-Suc*:
assumes *i* < *intlen xxs*
(∀ *j* ≤ *intlen xxs*. *intlen*(*nth xxs j*) > *0*)
shows *nth* (*lsum xxs a*) *i* < *nth* (*lsum xxs a*) (*Suc i*)
proof −
have *1*: *nth* (*lsum xxs a*) *i* = *a*+(∑ *k*::*nat*= *0*..*i*. *intlen*(*nth xxs k*))
using *assms less-imp-le-nat lsum-nth* **by** *blast*
have *2*: *nth* (*lsum xxs a*) (*Suc i*) = *a*+(∑ *k*::*nat*= *0*..(*Suc i*). *intlen*(*nth xxs k*))
using *assms* **by** (*simp add*: *lsum-nth Suc-leI*)
have *3*: (∑ *k*::*nat*= *0*..(*Suc i*). *intlen*(*nth xxs k*)) =
(∑ *k*::*nat*= *0*..*i*. *intlen*(*nth xxs k*)) + *intlen*(*nth xxs* (*Suc i*))
using *sum.atLeast0-atMost-Suc* **by** *blast*
have *4*: *intlen*(*nth xxs* (*Suc i*)) > *0*
using *assms* **by** *auto*
show *?thesis* **using** *1 2 3 4* **by** *linarith*
qed

lemma *lsum-addzero-nth-leq-Suc*:
assumes *i* < *intlen*(*addzero* (*lsum xxs 0*))
(∀ *j* ≤ *intlen xxs*. *intlen*(*nth xxs j*) > *0*)
shows *nth* (*addzero* (*lsum xxs 0*)) *i* < *nth* (*addzero* (*lsum xxs 0*)) (*Suc i*)
proof (*cases i*)
case *0*
then show *?thesis*
by (*metis add.left-neutral addzero-def assms*(*2*) *dual-order.order-iff-strict gr-zeroI*
interval-nth-Suc interval-nth-zero-intfirst lsum-addzero-intfirst lsum-intfirst)
next

```
case (Suc nat)
then show ?thesis
 by (metis Suc-less-eq addzero-def assms(1) assms(2) interval-nth-Suc intlen.simps(2) lsum-intlen
    lsum-nth-leq-Suc plus-1-eq-Suc)
qed

lemma lsum-idx:
assumes (∀ j ≤ intlen xxs. intlen(nth xxs j) > 0)
shows index-sequence (nth (lsum xxs a) 0) (lsum xxs a)
by (simp add: assms index-sequence-def lsum-intlen lsum-nth-leq-Suc)

lemma lsum-addzero-idx:
assumes (∀ j ≤ intlen xxs. intlen(nth xxs j) > 0)
shows index-sequence 0 (addzero (lsum xxs 0))
by (metis interval-idx-expand1 add.left-neutral addzero-def assms interval-nth-zero-intfirst le0
          lsum-idx lsum-intfirst)


lemma filt-lfuse-lsum-a:
assumes lastfirst (xs ⊙xxs)
        (∀ j ≤ intlen xxs. intlen(nth xxs j) > 0)
        intlen xs > 0
shows (filt (fuse xs (lfuse xxs)) (lsum xxs (intlen xs))) =
         (filt (lfuse xxs) ( (lsum xxs 0)))
using assms
proof
  (induct xxs arbitrary: xs)
  case (St x)
  then show ?case
   proof −
    have 1: filt (fuse xs (lfuse ⟨x⟩)) (lsum ⟨x⟩ (intlen xs)) =
             filt (fuse xs x) (lsum ⟨x⟩ (intlen xs))
      by simp
    have 2: filt (fuse xs x) (lsum ⟨x⟩ (intlen xs)) =
             filt (fuse xs x) ⟨intlen xs + intlen x⟩
      by simp
    have 3: filt (fuse xs x) (⟨intlen xs + intlen x⟩) =
             ⟨(nth (fuse xs x) (intlen xs + intlen x))⟩
      by simp
    have 4: ⟨(nth (fuse xs x) (intlen xs + intlen x))⟩ =
             ⟨(nth x (intlen x))⟩
    using St.prems interval-fuse-nth-a by auto
    have 5: filt (lfuse ⟨x⟩) (lsum ⟨x⟩ 0) =
             filt (x) (⟨intlen x⟩)
          by simp
    have 6: filt (x) (⟨intlen x⟩) = ⟨(nth x (intlen x))⟩
            by auto
    show ?thesis
    by (simp add: 4)
   qed
```

next
case (*Cons x1a xxs*)
then show *?case*
 proof −
  have *01*: *filt* (*fuse xs* (*lfuse* (*x1a* ⊙ *xxs*))) (*lsum* (*x1a* ⊙ *xxs*) (*intlen xs*)) =
            filt (*fuse xs* (*fuse x1a* (*lfuse xxs*))) (*lsum* (*x1a* ⊙ *xxs*) (*intlen xs*))
    by *simp*
  have *02*: *filt* (*fuse xs* (*fuse x1a* (*lfuse xxs*)))
                 (*lsum* (*x1a* ⊙ *xxs*) (*intlen xs*)) =
            filt (*fuse xs* (*fuse x1a* (*lfuse xxs*)))
                 ((*intlen xs* + *intlen x1a*) ⊙ *lsum* (*xxs*) (*intlen xs* +*intlen x1a*))
       by *simp*
  have *03*: *filt* (*fuse xs* (*fuse x1a* (*lfuse xxs*)))
                 ((*intlen xs* + *intlen x1a*) ⊙ *lsum* (*xxs*) (*intlen xs* +*intlen x1a*)) =
            (*nth* (*fuse xs* (*fuse x1a* (*lfuse xxs*))) (*intlen xs* + *intlen x1a*)) ⊙
            filt (*fuse xs* (*fuse x1a* (*lfuse xxs*))) (*lsum* (*xxs*) (*intlen xs* +*intlen x1a*))
     by *simp*
  have *04*: (*nth* (*fuse xs* (*fuse x1a* (*lfuse xxs*))) (*intlen xs* + *intlen x1a*)) =
            (*nth* (*fuse xs x1a*) (*intlen xs* +*intlen x1a*))
     proof −
     have *f1*: *Interval.nth xs* (*intlen xs*) = *Interval.nth x1a 0*
     using *Cons.prems*(*1*) **by** *force*
     have *f2*: *lastfirst* (*x1a* ⊙ *xxs*)
     using *Cons.prems*(*1*) *lastfirst.simps*(*2*) **by** *blast*
     then have *Interval.nth x1a* (*intlen x1a*) = *Interval.nth* (*Interval.nth xxs 0*) *0*
     by *simp*
     then show *?thesis*
     using *f2 f1*
       by (*metis Cons.prems*(*1*) *add.right-neutral interval-fuse-intlen-a*
            interval-fuse-nth-a interval-intfirst-lfuse-intfirst interval-nth-intlen-intlast
           interval-nth-zero interval-nth-zero-intfirst le0 le-add1 lfuse-Cons)
     qed
  have *05*: (*nth* (*fuse xs x1a*) (*intlen xs* +*intlen x1a*)) = (*nth x1a* (*intlen x1a*))
    using *Cons.prems*(*1*) *interval-fuse-nth-a* **by** *force*
  have *06*: *filt* (*fuse xs* (*fuse x1a* (*lfuse xxs*))) (*lsum* (*xxs*) (*intlen xs* +*intlen x1a*)) =
             filt (*fuse* (*fuse xs x1a*) (*lfuse xxs*)) (*lsum* (*xxs*) (*intlen*(*fuse xs x1a*)))
    by (*metis Cons.prems*(*1*) *interval-FusionAssoc interval-fuse-intlen-a interval-intfirst-lfuse*
        interval-intfirst-lfuse-intfirst lastfirst.simps(*2*))
  have *07*: *lastfirst* ((*fuse xs x1a*)⊙ *xxs*)
   by (*metis 05 Cons.prems*(*1*) *interval-fuse-intlen-a interval-nth-intlen-intlast lastfirst.simps*(*2*))
  have *08*: *intlen* (*fuse xs x1a*) > *0*
   by (*simp add*: *Cons.prems interval-fuse-intlen-a*)
  have *09*: *filt* (*fuse* (*fuse xs x1a*) (*lfuse xxs*)) (*lsum* (*xxs*) (*intlen*(*fuse xs x1a*))) =
            filt (*lfuse xxs*) (*lsum* (*xxs*) *0*)
     by (*metis 07 08 Cons.prems*(*2*) *Suc-leI interval-nth-Suc intlen.simps*(*2*) *less-Suc-eq-le*
        local.Cons(*1*) *plus-1-eq-Suc*)
  have *10*: *filt* (*fuse xs* (*fuse x1a* (*lfuse xxs*)))
                 ((*intlen xs* + *intlen x1a*) ⊙ *lsum* (*xxs*) (*intlen xs* +*intlen x1a*)) =
            (*nth x1a* (*intlen x1a*)) ⊙ *filt* (*lfuse xxs*) (*lsum* (*xxs*) *0*)
   by (*simp add*: *04 05 06 09*)

```
    have 11: filt (lfuse (x1a ⊙ xxs)) (lsum (x1a ⊙ xxs) 0) =
             filt (fuse x1a (lfuse xxs)) ((intlen x1a) ⊙ (lsum xxs (intlen x1a)))
     by simp
    have 12: filt (fuse x1a (lfuse xxs)) ((intlen x1a) ⊙ (lsum xxs (intlen x1a))) =
             (nth (fuse x1a (lfuse xxs)) (intlen x1a)) ⊙
             (filt (fuse x1a (lfuse xxs)) (lsum xxs (intlen x1a)))
       by simp
    have 13: (nth (fuse x1a (lfuse xxs)) (intlen x1a)) = (nth x1a (intlen x1a))
     by (metis Cons.prems(1) eq-iff interval-fuse-intlen-a interval-fuse-nth
         interval-intfirst-lfuse-intfirst lastfirst.simps(2) le-add1)
    have 14: (filt (fuse x1a (lfuse xxs)) (lsum xxs (intlen x1a))) =
             filt (lfuse xxs) (lsum xxs 0)
      using Cons.prems(1) Cons.prems(2) interval-nth-zero intlen.simps(2) lastfirst.simps(2)
       local.Cons(1) by fastforce
    have 15: filt (fuse x1a (lfuse xxs)) ((intlen x1a) ⊙ (lsum xxs (intlen x1a))) =
             (nth x1a (intlen x1a)) ⊙ (filt (lfuse xxs) (lsum xxs 0))
     by (simp add: 13 14)
     show ?thesis using 10 15 by auto
   qed
qed

lemma filt-lfuse-lsum:
assumes lastfirst (xs ⊙xxs)
        (∀ j ≤ intlen xxs. intlen(nth xxs j) > 0)
        intlen xs > 0
shows  (filt (lfuse (xs⊙xxs)) (addzero (lsum (xs⊙xxs) 0))) =
       (intfirst xs)⊙ (intlast xs)⊙ (filt (lfuse xxs) (lsum xxs 0))
proof −
 have 1:  lfuse (xs ⊙ xxs) = fuse xs (lfuse xxs)
   by simp
 have 2: (filt (lfuse (xs⊙xxs)) (addzero (lsum (xs⊙xxs) 0))) =
         (filt (fuse xs (lfuse xxs)) (addzero (lsum (xs⊙xxs) 0)))
  using 1 by simp
 have 3: addzero (lsum (xs⊙xxs) 0) =
        0⊙(intlen xs) ⊙ (lsum xxs (intlen xs))
   using lsum-addzero-cons by auto
 have 4: (filt (fuse xs (lfuse xxs)) (addzero (lsum (xs⊙xxs) 0))) =
         (filt (fuse xs (lfuse xxs)) (0⊙(intlen xs) ⊙ (lsum xxs (intlen xs))))
   using 3 by auto
 have 5:  (filt (fuse xs (lfuse xxs)) (0⊙(intlen xs) ⊙ (lsum xxs (intlen xs)))) =
           (nth (fuse xs (lfuse xxs)) 0) ⊙
           (nth (fuse xs (lfuse xxs)) (intlen xs)) ⊙
           (filt (fuse xs (lfuse xxs)) (lsum xxs (intlen xs)))
   by simp
 have 6: (nth (fuse xs (lfuse xxs)) 0) = (nth xs 0)
   using assms by (metis interval-fuse-nth interval-intfirst-lfuse-intfirst interval-intlen-gr-zero
      lastfirst.simps(2))
 have 7: (nth (fuse xs (lfuse xxs)) (intlen xs)) = (nth xs (intlen xs))
   using assms by (metis interval-fuse-intlen-a interval-fuse-nth interval-intfirst-lfuse-intfirst
      lastfirst.simps(2) le-add1 order-refl)
```

have *8*: *index-sequence 0* (*addzero* (*lsum* *xxs 0*))
using *assms lsum-addzero-idx* **by** *blast*
have *9*: (*filt* (*fuse xs* (*lfuse xxs*)) (*lsum xxs* (*intlen xs*))) =
(*filt* (*lfuse xxs*) (*lsum xxs 0*))
using *assms filt-lfuse-lsum-a* **by** *blast*
show *?thesis*
using *3 6 7 9 interval-nth-intlen-intlast interval-nth-zero-intfirst* **by** *force*
qed

lemma *filt-lfuse-lsum-1*:
assumes *lastfirst* (*xs* ⊙*xxs*)
(∀ *j* ≤ *intlen xxs*. *intlen*(*nth xxs j*) > *0*)
intlen xs > *0*
shows (*filt* (*lfuse* (*xs*⊙*xxs*)) ((*lsum* (*xs*⊙*xxs*) *0*))) =
(*intlast xs*)⊙ (*filt* (*lfuse xxs*) (*lsum xxs 0*))
using *assms* **by** (*simp*,
metis assms(*1*) *eq-iff filt-lfuse-lsum-a interval-fuse-intlen-a*
interval-fuse-nth interval-intfirst-lfuse-intfirst lastfirst.*simps*(*2*) *le-add1*)

lemma *filt-lfuse-lsum-2*:
assumes *lastfirst* (*xxs*)
(∀ *j* ≤ *intlen xxs*. *intlen*(*nth xxs j*) > *0*)
j ≤ *intlen xxs*
shows (*nth* (*filt* (*lfuse* (*xxs*)) ((*lsum* (*xxs*) *0*))) *j*) = *intlast*(*nth xxs j*)
using *assms*
proof (*induct xxs arbitrary*: *j*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xxs*)
then show *?case*
proof −
have *1*: *j*= *0* ∧ *j* ≤ *intlen* (*x1a* ⊙ *xxs*) ⟶
nth (*filt* (*lfuse* (*x1a* ⊙ *xxs*)) (*lsum* (*x1a* ⊙ *xxs*) *0*)) *j* = *intlast* (*nth* (*x1a* ⊙ *xxs*) *j*)
using *filt-lfuse-lsum-1* **using** *Cons*.*prems* **by** *fastforce*
have *2*: *j*> *0* ∧ *j* ≤ *intlen* (*x1a* ⊙ *xxs*) ⟶
nth (*filt* (*lfuse* (*x1a* ⊙ *xxs*)) (*lsum* (*x1a* ⊙ *xxs*) *0*)) *j* = *intlast* (*nth* (*x1a* ⊙ *xxs*) *j*)
by (*metis Cons*.*hyps Cons*.*prems*(*2*) *Suc-le-mono Suc-pred filt-lfuse-lsum-1*
interval-intlen-gr-zero interval-nth-Suc interval-nth-zero intlen.*simps*(*2*)
lastfirst.*simps*(*2*) *local*.*Cons*(*2*) *plus-1-eq-Suc*)
show *?thesis* **using** *1 2 not-gr-zero* **using** *Cons*.*prems*(*3*) **by** *blast*
qed
qed

lemma *filt-lfuse-lsum-3*:
assumes *lastfirst* (*xxs*)
(∀ *j* ≤ *intlen xxs*. *intlen*(*nth xxs j*) > *0*)
j ≤ *intlen* (*addzero* (*lsum xxs 0*))
shows (*j*=*0* ⟶ (*nth* (*filt* (*lfuse* (*xxs*)) (*addzero*(*lsum* (*xxs*) *0*))) *j*) = *intfirst*(*intfirst xxs*))
∧

$(j>0 \longrightarrow (nth\ (filt\ (lfuse\ (xxs))\ (addzero(lsum\ (xxs)\ 0)))\ j) = intlast(nth\ xxs\ (j-1)))$

using *assms*
proof −
have *1*: $j \leq intlen\ (addzero\ (lsum\ xxs\ 0)) \wedge j=0 \longrightarrow$
$(nth\ (filt\ (lfuse\ (xxs))\ (addzero(lsum\ (xxs)\ 0)))\ j) = intfirst(intfirst\ xxs)$
using *assms* **by** (*metis filt-intlen filt-nth interval-lastfirst-lfuse interval-nth-zero-intfirst lsum-addzero-intfirst*)
have *2*: $j \leq intlen\ (addzero\ (lsum\ xxs\ 0)) \wedge j>0 \longrightarrow$
$(nth\ (filt\ (lfuse\ xxs)\ (addzero(lsum\ (xxs)\ 0)))\ j) =$
$(nth\ (lfuse\ xxs)\ (nth\ (addzero(lsum\ (xxs)\ 0))\ j))$

by (*simp add*: *filt-intlen filt-nth*)
have *3*: $j \leq intlen\ (addzero\ (lsum\ xxs\ 0)) \wedge j>0 \longrightarrow$
$nth\ (addzero(lsum\ (xxs)\ 0))\ j = nth\ (lsum\ xxs\ 0)\ (j-1)$

by (*metis One-nat-def Suc-pred interval-nth-Suc leD lsum-addzero-intlen lsum-addzero-nth neq0-conv*)
have *4*: $j \leq intlen\ (addzero\ (lsum\ xxs\ 0)) \wedge j>0 \longrightarrow$
$(nth\ (lfuse\ xxs)\ (nth\ (lsum\ xxs\ 0)\ (j-1))) = intlast(nth\ xxs\ (j-1))$
by (*metis assms diff-is-0-eq′ filt-intlen filt-lfuse-lsum-2 filt-nth interval-nth-zero-intfirst le0 le-diff-conv lsum-addzero-intlen lsum-intlen neq0-conv*)
show *?thesis*
using *1 2 3 4* **by** (*simp add*: *assms*(*3*))
qed

lemma *filt-lfuse-lsum-4*:
assumes *lastfirst* (*xxs*)
$(\forall\ j \leq intlen\ xxs.\ intlen(nth\ xxs\ j) > 0)$
shows $(nth\ (addzero\ (lsum\ xxs\ 0))\ (intlen\ (addzero\ (lsum\ xxs\ 0)))) \leq intlen\ (lfuse\ xxs)$
using *assms*
by (*metis add-cancel-right-left eq-iff interval-intlast-prefix interval-lfuse-intlen interval-prefix-intlen lsum-addzero-intlast lsum-intlen lsum-nth*)

lemma *filt-lfuse-lsum-5*:
assumes *lastfirst* (*xxs*)
$(\forall\ j \leq intlen\ xxs.\ intlen(nth\ xxs\ j) > 0)$
$i \leq intlen\ (addzero\ (lsum\ xxs\ 0))$
shows $(nth\ (addzero\ (lsum\ xxs\ 0))\ i) \leq intlen\ (lfuse\ xxs)$
using *assms filt-lfuse-lsum-4*[*of xxs*] *lsum-addzero-idx*[*of xxs*]
proof −
have *f1*: $Interval.nth\ (addzero\ (lsum\ xxs\ 0))\ (intlen\ (addzero\ (lsum\ xxs\ 0))) \leq intlen\ (lfuse\ xxs)$
using $\langle\forall j \leq intlen\ xxs.\ 0 < intlen\ (Interval.nth\ xxs\ j)\rangle$
$\langle[\![lastfirst\ xxs; \forall j \leq intlen\ xxs.\ 0 < intlen\ (Interval.nth\ xxs\ j)]\!] \Longrightarrow$
$Interval.nth\ (addzero\ (lsum\ xxs\ 0))\ (intlen\ (addzero\ (lsum\ xxs\ 0))) \leq intlen\ (lfuse\ xxs)\rangle$
$\langle lastfirst\ xxs\rangle$ **by** *blast*
have $\neg\ i < intlen\ (addzero\ (lsum\ xxs\ 0)) \vee$
$nth\ (addzero\ (lsum\ xxs\ 0))\ i < nth\ (addzero\ (lsum\ xxs\ 0))\ (intlen\ (addzero\ (lsum\ xxs\ 0)))$
using $\langle\forall j \leq intlen\ xxs.\ 0 < intlen\ (Interval.nth\ xxs\ j) \Longrightarrow index\text{-}sequence\ 0\ (addzero\ (lsum\ xxs\ 0))\rangle$

```
    ⟨∀j≤intlen xxs. 0 < intlen (Interval.nth xxs j)⟩ interval-idx-less-last-1 by blast
then show ?thesis
using f1 ⟨i ≤ intlen (addzero (lsum xxs 0))⟩ by force
qed

lemma lfuse-intlen-b:
 assumes lastfirst xxs
  shows ( ∀i. 1 ≤ i ∧ i ≤intlen (xxs) ⟶
            (∀j≤intlen (nth (xxs) (i)).
              nth ((lsum xxs 0) ) (i−1) + j ≤ intlen (lfuse xxs)))
proof −
 have 0: (∀i. 1 ≤ i ∧ i ≤intlen (xxs) ⟶
         (i−1) ≤ intlen xxs
        )
    by linarith
 have 1: (∀i. 1 ≤ i ∧ i ≤intlen (xxs) ⟶
            nth ((lsum xxs 0) ) (i−1) =
             (∑ k::nat= 0..(i−1). intlen(nth xxs k))
         )
       by (metis 0 add.left-neutral lsum-nth)
 have 2: intlen (lfuse xxs) = (∑ k::nat= 0..(intlen xxs). intlen(nth xxs k))
    using assms interval-lfuse-intlen by blast
 have 3: (∀i. 1 ≤ i ∧ i ≤intlen (xxs) ⟶
         (∀j≤intlen (nth (xxs) (i)).
          (∑ k::nat= 0..(i−1). intlen(nth xxs k)) + j ≤
           (∑ k::nat= 0..(i). intlen(nth xxs k))
         ))

    by (metis (no-types, lifting) add-le-cancel-left le-add-diff-inverse plus-1-eq-Suc
     sum.atLeast0-atMost-Suc)
 have 4: (∀i. 1 ≤ i ∧ i ≤intlen (xxs) ⟶
         (∑ k::nat= 0..(i). intlen(nth xxs k)) ≤
          (∑ k::nat= 0..(intlen xxs). intlen(nth xxs k))
        )

      by (metis add.commute diff-is-0-eq diff-zero le-add1 le-add-diff-inverse2 not-less-eq-eq
          sum.ub-add-nat)
 show ?thesis
 using 1 2 3 4 by fastforce
qed


lemma lsum-shift:
 assumes lastfirst xxs
        (∀ j ≤ intlen xxs. intlen(nth xxs j) > 0)
        i≤intlen xxs
 shows    nth (lsum xxs a) i = a+ nth (lsum xxs 0) i
using assms by (simp add: lsum-nth)

lemma lsum-lfuse-nth-lsum-nth:
```

```
assumes lastfirst xxs
        (∀ j ≤ intlen xxs. intlen(nth xxs j) > 0)
shows    (∀ i≤intlen xxs.
             (∀ j≤ intlen(nth xxs i).
              ( (nth (lfuse xxs) ((nth (addzero (lsum xxs 0)) i) +j) )) =
              ( (nth (nth xxs i) j))  ))
using assms
proof
 (induct xxs)
 case (St x)
 then show ?case
 by (metis Interval.nth.simps(1) add-cancel-right-left interval-nth-zero-intfirst intlen.simps(1)
     le-zero-eq lfuse-St lsum-addzero-intfirst)
 next
 case (Cons x1a xxs)
 then show ?case
  proof -
   have 1: (∀ i≤intlen (x1a ⊙ xxs).
            (∀ j≤intlen (nth (x1a ⊙ xxs) i).
              nth (lfuse (x1a ⊙ xxs)) (nth (addzero (lsum (x1a ⊙ xxs) 0)) i + j) =
              nth (nth (x1a ⊙ xxs) i) j))
          =
         ( (∀ j≤intlen (nth (x1a ⊙ xxs) 0).
                nth (lfuse (x1a ⊙ xxs)) (nth (addzero (lsum (x1a ⊙ xxs) 0)) 0 + j) =
                nth (nth (x1a ⊙ xxs) 0) j) ∧
           (∀ i. 1 ≤ i ∧ i≤intlen (x1a ⊙ xxs) ⟶
                (∀ j≤intlen (nth (x1a ⊙ xxs) i).
                  nth (lfuse (x1a ⊙ xxs)) (nth (addzero (lsum (x1a ⊙ xxs) 0)) i + j) =
                  nth (nth (x1a ⊙ xxs) i) j)) )

    by (metis One-nat-def Suc-leI interval-intlen-gr-zero not-gr-zero)
   have 2: (∀ j≤intlen (nth (x1a ⊙ xxs) 0).
                nth (lfuse (x1a ⊙ xxs)) (nth (addzero (lsum (x1a ⊙ xxs) 0)) 0 + j) =
                nth (nth (x1a ⊙ xxs) 0) j)
           =
          (∀ j≤intlen (x1a).
                nth (lfuse (x1a ⊙ xxs)) ( j) = nth (x1a) j)

      by (metis add-cancel-right-left interval-nth-zero interval-nth-zero-intfirst
          lsum-addzero-intfirst)
   have 3: intlast x1a = intfirst(intfirst xxs)
     using Cons.prems lastfirst.simps(2) by blast
   have 4: (∀ j≤intlen (x1a).
                nth (lfuse (x1a ⊙ xxs)) ( j) = nth (x1a) j)
    by (metis Cons.prems(1) interval-fuse-intlen-a interval-fuse-nth
        interval-intfirst-lfuse-intfirst lastfirst.simps(2) le-add1 le-trans  lfuse-Cons)
   have 41: (∀ j≤intlen (nth (x1a ⊙ xxs) 0).
                nth (lfuse (x1a ⊙ xxs)) (nth (addzero (lsum (x1a ⊙ xxs) 0)) 0 + j) =
                nth (nth (x1a ⊙ xxs) 0) j)
    using 2 4 by blast
```

have *5*: (∀ *i*. *1* ≤ *i* ∧ *i*≤*intlen* (*x1a* ⊙ *xs*) ⟶
(∀ *j*≤*intlen* (*nth* (*x1a* ⊙ *xs*) *i*).
nth (*lfuse* (*x1a* ⊙ *xs*)) (*nth* (*addzero* (*lsum* (*x1a* ⊙ *xs*) *0*)) *i* + *j*) =
nth (*nth* (*x1a* ⊙ *xs*) *i*) *j*)) =
(∀ *i*. *0* ≤ *i*−*1* ∧ *i*−*1* ≤*intlen* (*xs*) ⟶
(∀ *j*≤*intlen* (*nth* (*x1a* ⊙ *xs*) *i*).
nth (*lfuse* (*x1a* ⊙ *xs*)) (*nth* (*addzero* (*lsum* (*x1a* ⊙ *xs*) *0*)) *i* + *j*) =
nth (*nth* (*x1a* ⊙ *xs*) *i*) *j*))
using *1 2 4 le-diff-conv* **by** *auto*
have *6*: (∀ *i*. *0* ≤ *i*−*1* ∧ *i*−*1* ≤*intlen* (*xs*) ⟶
(∀ *j*≤*intlen* (*nth* (*x1a* ⊙ *xs*) *i*).
nth (*lfuse* (*x1a* ⊙ *xs*)) (*nth* (*addzero* (*lsum* (*x1a* ⊙ *xs*) *0*)) *i* + *j*) =
nth (*nth* (*x1a* ⊙ *xs*) *i*) *j*)) =
(∀ *i* ≤*intlen* (*xs*).
(∀ *j*≤*intlen* (*nth* (*x1a* ⊙ *xs*) (*Suc i*)).
nth (*lfuse* (*x1a* ⊙ *xs*)) (*nth* (*addzero* (*lsum* (*x1a* ⊙ *xs*) *0*)) (*Suc i*) + *j*) =
nth (*nth* (*x1a* ⊙ *xs*) (*Suc i*)) *j*)) (**is** *?L* = *?R*)
proof
show *?L* ⟹ *?R*
using *2 4* **by** *simp*
show *?R* ⟹ *?L*
using *2 4* **by** (*metis One-nat-def Suc-pred gr0I*)
qed
have *7*: (∀ *i* ≤*intlen* (*xs*).
(∀ *j*≤*intlen* (*nth* (*x1a* ⊙ *xs*) (*Suc i*)).
nth (*lfuse* (*x1a* ⊙ *xs*)) (*nth* (*addzero* (*lsum* (*x1a* ⊙ *xs*) *0*)) (*Suc i*) + *j*) =
nth (*nth* (*x1a* ⊙ *xs*) (*Suc i*)) *j*)) =
(∀ *i* ≤*intlen* (*xs*).
(∀ *j*≤*intlen* (*nth* (*xs*) (*i*)).
nth (*lfuse* (*x1a* ⊙ *xs*)) (*nth* (*addzero* (*lsum* (*x1a* ⊙ *xs*) *0*)) (*Suc i*) + *j*) =
nth (*nth* (*xs*) (*i*)) *j*))
by *simp*
have *8*: (∀ *i* ≤*intlen* (*xs*).
(∀ *j*≤*intlen* (*nth* (*xs*) (*i*)).
nth (*lfuse* (*x1a* ⊙ *xs*)) (*nth* (*addzero* (*lsum* (*x1a* ⊙ *xs*) *0*)) (*Suc i*) + *j*) =
nth (*nth* (*xs*) (*i*)) *j*)) =
(∀ *i* ≤*intlen* (*xs*).
(∀ *j*≤*intlen* (*nth* (*xs*) (*i*)).
nth (*fuse x1a* (*lfuse xs*)) (*nth* ((*lsum* (*x1a* ⊙ *xs*) *0*)) (*i*) + *j*) =
nth (*nth* (*xs*) (*i*)) *j*))
by (*metis interval-nth-Suc lfuse-Cons lsum-addzero-cons*)
have *9*: (∀ *i* ≤*intlen* (*xs*).
(∀ *j*≤*intlen* (*nth* (*xs*) (*i*)).
nth (*fuse x1a* (*lfuse xs*)) (*nth* ((*lsum* (*x1a* ⊙ *xs*) *0*)) (*i*) + *j*) =
nth (*nth* (*xs*) (*i*)) *j*))
=
(∀ *i* ≤*intlen* (*xs*).
(∀ *j*≤*intlen* (*nth* (*xs*) (*i*)).
nth (*fuse x1a* (*lfuse xs*)) (*nth* ((*intlen*(*x1a*)⊙(*lsum xs*) (*intlen x1a*))) *i* +*j*) =
nth (*nth* (*xs*) (*i*)) *j*))

```
  by simp
have 11: (∀ i ≤intlen (xxs).
          (∀ j≤intlen (nth (xxs) (i)).
            nth (fuse x1a (lfuse xxs)) (nth ((intlen(x1a)⊙ (lsum xxs) (intlen x1a)) ) (i) + j) =
            nth (nth (xxs) (i)) j))
           =
         ((∀ i. i=0 ⟶
           (∀ j≤intlen (nth (xxs) (i)).
             nth (fuse x1a (lfuse xxs)) (nth ((intlen(x1a)⊙ (lsum xxs) (intlen x1a)) ) (i) + j) =
             nth (nth (xxs) (i)) j))
            ∧
          (∀ i. 1 ≤ i ∧ i ≤intlen (xxs) ⟶
           (∀ j≤intlen (nth (xxs) (i)).
             nth (fuse x1a (lfuse xxs)) (nth ((intlen(x1a)⊙ (lsum xxs) (intlen x1a)) ) (i) + j) =
             nth (nth (xxs) (i)) j)) )

      by (metis Suc-leI add.commute add-less-same-cancel1  interval-nth-zero le0 le-eq-less-or-eq
          less-one not-gr-zero plus-1-eq-Suc)
have 12: (∀ i. i=0 ⟶
          (∀ j≤intlen (nth (xxs) (i)).
            nth (fuse x1a (lfuse xxs)) (nth ((intlen(x1a)⊙ (lsum xxs) (intlen x1a)) ) (i) + j) =
            nth (nth (xxs) (i)) j)) =
         (∀ j≤intlen (nth (xxs) (0)).
            nth (fuse x1a (lfuse xxs)) (nth ((intlen(x1a)⊙ (lsum xxs) (intlen x1a)) ) (0) + j) =
            nth (nth (xxs) (0)) j)

 by blast
have 13: (∀ j≤intlen (nth (xxs) (0)).
           nth (fuse x1a (lfuse xxs)) (nth ((intlen(x1a)⊙ (lsum xxs) (intlen x1a)) ) (0) + j) =
           nth (nth (xxs) (0)) j) =
         (∀ j≤intlen (nth (xxs) (0)).
           nth (fuse x1a (lfuse xxs)) (intlen(x1a) + j) =
           nth (nth (xxs) (0)) j)

  by auto
have 14: (∀ j≤intlen (nth (xxs) (0)).
           nth (fuse x1a (lfuse xxs)) (intlen(x1a) + j) =
           nth (nth (xxs) (0)) j)
         =
         (∀ j≤intlen (nth (xxs) (0)).
           nth ( (lfuse xxs)) (j) =
           nth (nth (xxs) (0)) j)

  using Cons.prems interval-fuse-nth-a interval-intfirst-lfuse-intfirst interval-lfuse-intlen-a
  by (metis lastfirst.simps(2) le0)
have 15: (∀ i. 1 ≤ i ∧ i ≤intlen (xxs) ⟶
          (∀ j≤intlen (nth (xxs) (i)).
            nth (fuse x1a (lfuse xxs)) (nth ((intlen(x1a)⊙ (lsum xxs) (intlen x1a)) ) (i) + j) =
            nth (nth (xxs) (i)) j)) =
```

$(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ (fuse\ x1a\ (lfuse\ xxs))\ (nth\ (\ (lsum\ xxs\ (intlen\ x1a))\ )\ (i-1) + j) =$
$nth\ (nth\ (xxs)\ (i))\ j))$

by (*metis interval-nth-Suc ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
have *16*: $(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$nth\ (\ (lsum\ xxs\ (intlen\ x1a))\ )\ (i-1) =$
$intlen\ x1a + nth\ (lsum\ xxs\ 0)\ \ (i-1))$

by (*simp add*: *le-diff-conv lsum-nth*)
have *17*: $(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ (fuse\ x1a\ (lfuse\ xxs))\ (nth\ (\ (lsum\ xxs\ (intlen\ x1a))\ )\ (i-1) + j) =$
$nth\ (nth\ (xxs)\ (i))\ j))$
$=$
$(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ (fuse\ x1a\ (lfuse\ xxs))\ (intlen\ x1a + nth\ ((lsum\ xxs\ 0)\ )\ (i-1) + j) =$
$nth\ (nth\ (xxs)\ (i))\ j))$
using *16* **by** *auto*
have *18*: $(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ ((lsum\ xxs\ 0)\ )\ (i-1) + j \le intlen\ (lfuse\ xxs)))$
using *Cons.prems lastfirst.simps*(*2*) *lfuse-intlen-b* **by** *blast*
have *19*: $(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ (fuse\ x1a\ (lfuse\ xxs))\ (intlen\ x1a + nth\ ((lsum\ xxs\ 0)\ )\ (i-1) +j) =$
$nth\ (\ (lfuse\ xxs))\ (\ nth\ ((lsum\ xxs\ 0)\ )\ (i-1) + j)))$

by (*metis Cons.prems*(*1*) *ab-semigroup-add-class.add-ac*(*1*) *interval-fuse-nth-a interval-intfirst-lfuse-intfirst lastfirst.simps*(*2*) *lfuse-intlen-b*)
have *20*: $(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ (fuse\ x1a\ (lfuse\ xxs))\ (intlen\ x1a + nth\ ((lsum\ xxs\ 0)\ )\ (i-1) + j) =$
$nth\ (nth\ (xxs)\ (i))\ j)) =$
$(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ (\ (lfuse\ xxs))\ (\ nth\ ((lsum\ xxs\ 0)\ )\ (i-1) + j) =$
$nth\ (nth\ (xxs)\ (i))\ j))$

using *19* **by** *auto*
have *201*: $((\forall\, i.\ i=0 \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ (fuse\ x1a\ (lfuse\ xxs))\ (nth\ ((intlen(x1a) \odot\ (lsum\ xxs)\ (intlen\ x1a))\ )\ (i) + j) =$
$nth\ (nth\ (xxs)\ (i))\ j))$
$\wedge$
$(\forall\, i.\ 1 \le i \wedge i \le intlen\ (xxs) \longrightarrow$
$(\forall\, j \le intlen\ (nth\ (xxs)\ (i)).$
$nth\ (fuse\ x1a\ (lfuse\ xxs))\ (nth\ ((intlen(x1a) \odot\ (lsum\ xxs)\ (intlen\ x1a))\ )\ (i) + j) =$

nth (*nth* (*xxs*) (*i*)) *j*))) =
((∀ *j*≤*intlen* (*nth* (*xxs*) (*0*)).
nth ((*lfuse xxs*)) (*j*) =
nth (*nth* (*xxs*) (*0*)) *j*) ∧
(∀ *i*. *1* ≤ *i* ∧ *i* ≤*intlen* (*xxs*) ⟶
(∀ *j*≤*intlen* (*nth* (*xxs*) (*i*)).
nth ((*lfuse xxs*)) (*nth* ((*lsum xxs 0*)) (*i*−*1*) + *j*) =
nth (*nth* (*xxs*) (*i*)) *j*)))
using *14 15 17 20* **by** *auto*
have *21*: *lastfirst xxs*
using *Cons.prems lastfirst.simps*(*2*) **by** *blast*
have *22*: (∀ *j* ≤ *intlen xxs*. *intlen*(*nth xxs j*) > *0*)
using *Cons.prems* **by** *auto*
have *23*: (∀ *i*≤*intlen xxs*.
(∀ *j*≤*intlen* (*nth xxs i*).
nth (*lfuse xxs*) (*nth* (*addzero* (*lsum xxs 0*)) *i* + *j*) =
nth (*nth xxs i*) *j*))
using *21 22 Cons.hyps* **by** *blast*
have *24*: ((∀ *j*≤*intlen* (*nth xxs 0*).
nth (*lfuse xxs*) (*nth* (*addzero* (*lsum xxs 0*)) *0* + *j*) =
nth (*Interval.nth xxs 0*) *j*) ∧
(∀ *i*. *1* ≤ *i* ∧ *i* ≤*intlen xxs* ⟶
(∀ *j*≤*intlen* (*nth xxs i*).
nth (*lfuse xxs*) (*nth* (*addzero* (*lsum xxs 0*)) *i* + *j*) =
nth (*nth xxs i*) *j*)))
using *23* **by** *blast*
have *25*: (∀ *j*≤*intlen* (*nth* (*xxs*) (*0*)).
nth ((*lfuse xxs*)) (*j*) =
nth (*nth* (*xxs*) (*0*)) *j*)
by (*metis 24 add-cancel-right-left interval-nth-zero-intfirst lsum-addzero-intfirst*)
have *26*: (∀ *i*. *1* ≤ *i* ∧ *i* ≤*intlen* (*xxs*) ⟶
nth (*addzero* (*lsum xxs 0*)) *i* = *nth* ((*lsum xxs 0*)) (*i*−*1*)
)
by (*metis addzero-def interval-nth-Suc less-eq-Suc-le less-le-trans lsum-intlen not-less-eq-eq ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc zero-less-one*)
have *27*: (∀ *i*. *1* ≤ *i* ∧ *i* ≤*intlen* (*xxs*) ⟶
(∀ *j*≤*intlen* (*nth* (*xxs*) (*i*)).
nth ((*lfuse xxs*)) (*nth* ((*lsum xxs 0*)) (*i*−*1*) + *j*) =
nth (*nth* (*xxs*) (*i*)) *j*)) =
(∀ *i*. *1* ≤ *i* ∧ *i* ≤*intlen xxs* ⟶
(∀ *j*≤*intlen* (*nth xxs i*).
nth (*lfuse xxs*) (*nth* (*addzero* (*lsum xxs 0*)) *i* + *j*) =
nth (*nth xxs i*) *j*))
by (*simp add*: *26*)
show *?thesis*
using *11 201 24 25 27 6 8* **by** *auto*
qed

qed

lemma *lcpl-lsum-less-th-equal*:
assumes *index-sequence 0 l*
$(nth\ l\ (intlen\ l)) = intlen\ \sigma$
$(\forall\ i{<}intlen\ l.\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models f \triangle g)$
$intlen\ \sigma > 0$
$i < intlen\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))$
shows $(nth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (Suc\ i)) \leq$
$intlen(\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))$
using *assms*
by (*metis* (*no-types*, *lifting*) *Suc-leI filt-lfuse-lsum-5 index-sequence-def*
lcpl-intlen-nth-gr-zero lcpl-lfuse-lastfirst neq0-conv)

lemma *lcpl-lsum-intlen*:
assumes *index-sequence 0 l*
$(nth\ l\ (intlen\ l)) = intlen\ \sigma$
$(\forall\ i{<}intlen\ l.\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models f \triangle g)$
$intlen\ \sigma > 0$
shows $intlen\ (addzero\ (lsum\ ((lcpl\ f\ g\ \sigma\ l))\ 0)) = intlen\ l$
proof −
have *1*: $intlen\ \sigma > 0 \longrightarrow intlen\ (lcpl\ f\ g\ \sigma\ l) = intlen\ l - 1$
using *assms index-sequence-def lcpl-intlen* **by** *fastforce*
have *2*: $intlen\ \sigma > 0 \longrightarrow intlen\ l > 0$
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *3*: $intlen\ \sigma > 0 \longrightarrow intlen\ (intfirst\ (lcpl\ f\ g\ \sigma\ l)) > 0$
by (*metis assms interval-intlen-gr-zero interval-nth-zero-intfirst lcpl-intlen-nth-gr-zero*)
have *4*: $intlen\ \sigma > 0 \longrightarrow intlen\ (lcpl\ f\ g\ \sigma\ l) = 0 \longrightarrow$
$intlen\ (addzero\ (lsum\ ((lcpl\ f\ g\ \sigma\ l))\ 0)) = intlen\ l$
using *1 2 3 lsum-addzero-intlen* **by** *fastforce*
have *5*: $intlen\ \sigma > 0 \longrightarrow intlen\ (lcpl\ f\ g\ \sigma\ l) > 0 \longrightarrow$
$intlen\ (addzero\ (lsum\ ((lcpl\ f\ g\ \sigma\ l))\ 0)) = intlen\ l$
by (*simp add*: *1 lsum-addzero-intlen*)
show *?thesis* **using** *4 5* **using** *assms*(*4*) **by** *blast*
qed

lemma *lcpl-lsum-nth*:
assumes *index-sequence 0 l*
$(nth\ l\ (intlen\ l)) = intlen\ \sigma$
$(\forall\ i{<}intlen\ l.\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models f \triangle g)$
$intlen\ \sigma{>}0$
$j \leq intlen\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))$
shows $(\ j{=}0 \longrightarrow$
$(nth\ (filt\ (lfuse\ ((lcpl\ f\ g\ \sigma\ l)))\ (addzero(lsum\ ((lcpl\ f\ g\ \sigma\ l))\ 0)))\ j) =$
$intfirst(intfirst\ (lcpl\ f\ g\ \sigma\ l))\ )$

```
         ∧
         (j>0 ⟶ (nth (filt (lfuse ((lcpl f g σ l))) (addzero(lsum ((lcpl f g σ l)) 0))) j) =
           intlast(nth (lcpl f g σ l) (j−1)))

proof −
 have 0: intlen σ >0 ⟶ intlen l >0
   using assms gr-zerol index-sequence-def by fastforce
 have 1: intlen σ >0 ⟶ lastfirst (lcpl f g σ l)
    using 0 assms index-sequence-def lcpl-lfuse-lastfirst by blast
 have 2: intlen σ>0 ⟶
        (∀ j ≤ intlen (lcpl f g σ l). intlen(nth (lcpl f g σ l) j) > 0)
   by (simp add: assms lcpl-intlen-nth-gr-zero)
 have 3:  intlen σ>0 ⟶
          intlen (addzero (lsum (lcpl f g σ l) 0)) =
          intlen (lcpl f g σ l) +1

         by (metis 2 One-nat-def Suc-eq-plus1 interval-intlen-gr-zero interval-nth-zero-intfirst
        le-imp-less-or-eq lsum-addzero-intlen)
 have 4:  intlen σ > 0 ⟶
          (nth (filt (lfuse ((lcpl f g σ l))) (addzero(lsum ((lcpl f g σ l)) 0))) 0) =
            intfirst(intfirst (lcpl f g σ l))
    by (simp add: 1 2 filt-lfuse-lsum-3)
 have 5: intlen σ > 0 ⟶
         j ≤ intlen (addzero (lsum (lcpl f g σ l) 0)) ∧ j>0 ⟶
         (nth (filt (lfuse ((lcpl f g σ l))) (addzero(lsum ((lcpl f g σ l)) 0))) (j))
          = intlast(nth (lcpl f g σ l) (j−1))
   by (simp add: 1 2 filt-lfuse-lsum-3)
 from 4 5 show ?thesis using assms(4) assms(5) by blast
qed

lemma lcpl-lsum-nth-a:
 assumes index-sequence 0 l
         (nth l (intlen l)) = intlen σ
         (∀ i<intlen l. (sub (nth l i) (nth l (Suc i)) σ) ⊨ f △ g)
         intlen σ>0
         j ≤ intlen l
 shows   (nth (filt (lfuse ((lcpl f g σ l))) (addzero(lsum ((lcpl f g σ l)) 0))) j) =
         (nth l j)
proof −
 have 1: intlen σ>0 ⟶ intlen l>0
  using assms gr-zerol index-sequence-def by fastforce
 have 2: intlen σ>0 ⟶ intlen (addzero (lsum (lcpl f g σ l) 0)) = intlen l
    by (simp add: assms lcpl-lsum-intlen)
 have 3: intlen σ>0 ⟶
         j ≤ intlen l ⟶
           ( j=0 ⟶
         (nth (filt (lfuse ((lcpl f g σ l))) (addzero(lsum ((lcpl f g σ l)) 0))) j) =
          intfirst(intfirst (lcpl f g σ l)) )
         ∧
         (j>0 ⟶ (nth (filt (lfuse ((lcpl f g σ l))) (addzero(lsum ((lcpl f g σ l)) 0))) j) =
```

$intlast(nth\ (lcpl\ f\ g\ \sigma\ l)\ (j-1)))$

by (*metis 2 assms lcpl-lsum-nth*)
have *4*: $intlen\ \sigma>0 \longrightarrow$
$j \le intlen\ l \wedge j=0 \longrightarrow intfirst(intfirst\ (lcpl\ f\ g\ \sigma\ l)) = (nth\ l\ j)$
using *assms lcpl-intfirst[of l σ f g]*
by (*metis 1 index-sequence-def interval-nth-zero-intfirst*)
have *5*: $intlen\ \sigma>0 \longrightarrow$
$j \le intlen\ l \wedge j>0 \longrightarrow j-1 \le intlen\ (lcpl\ f\ g\ \sigma\ l)$
using *assms* **by** (*metis 1 diff-le-mono index-sequence-def lcpl-intlen*)
have *6*: $intlen\ \sigma>0 \longrightarrow$
$j \le intlen\ l \wedge j>0 \longrightarrow intlast(nth\ (lcpl\ f\ g\ \sigma\ l)\ (j-1)) = (nth\ l\ j)$
using *lcpl-intlast-nth*
by (*metis 5 One-nat-def Suc-pred assms*)
show *?thesis*
using *3 4 6* **using** *assms(4) assms(5)* **by** *auto*
qed

lemma *lcpl-filt-lfuse-lsum*:
assumes *index-sequence 0 l*
$(nth\ l\ (intlen\ l)) = intlen\ \sigma$
$(\forall\ i<intlen\ l.\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models f \triangle g)$
$intlen\ \sigma>0$
shows $(filt\ (lfuse\ ((lcpl\ f\ g\ \sigma\ l)))\ (addzero(lsum\ ((lcpl\ f\ g\ \sigma\ l))\ 0))) = l$
using *assms*
proof −
have *0*: $intlen\ \sigma>0 \longrightarrow\ \ intlen\ l > 0$
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *1*: $intlen\ \sigma>0 \longrightarrow$
$intlen\ (filt\ (lfuse\ ((lcpl\ f\ g\ \sigma\ l)))\ (addzero(lsum\ ((lcpl\ f\ g\ \sigma\ l))\ 0))) = intlen\ l$
by (*simp add*: *assms filt-intlen lcpl-lsum-intlen*)
have *2*: $intlen\ \sigma>0 \longrightarrow$
$(\forall\ j.\ j\le intlen\ l \longrightarrow$
$(nth\ (filt\ (lfuse\ ((lcpl\ f\ g\ \sigma\ l)))\ (addzero(lsum\ ((lcpl\ f\ g\ \sigma\ l))\ 0)))\ j) =$
$(nth\ l\ j))$
by (*simp add*: *assms lcpl-lsum-nth-a*)
from *1 2* **show** *?thesis* **by** (*simp add*: *assms(4) interval-eq-nth-eq*)
qed

lemma *lcpl-lfuse-filt-power*:
assumes *index-sequence 0 l*
$(nth\ l\ (intlen\ l)) = intlen\ \sigma$
$(\forall\ i<intlen\ l.\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models f \triangle g)$
$intlen\ \sigma >0$
shows $powerinterval\ g\ (filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))$
proof −
have *0*: $intlen\ \sigma > 0 \longrightarrow intlen\ l >0$
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *01*: $(\forall\ i<intlen\ l.$

g (filt (sub (nth l i) (nth l (Suc i)) σ) (cpl f g (sub (nth l i) (nth l (Suc i)) σ))))
using assms cpl-projection **by** auto
have 02: intlen σ > 0 ⟶ (∀ i<intlen l. g (filt σ (nth (lcpl f g σ l) i)))
using 0 assms index-sequence-def lcpl-lfuse-filt-power-help **by** blast
have 03 : powerinterval g (filt σ (lfuse (lcpl f g σ l))) (addzero (lsum (lcpl f g σ l) 0)) =
(∀ i<intlen (addzero (lsum (lcpl f g σ l) 0)).
g (sub (nth (addzero (lsum (lcpl f g σ l) 0)) i)
(nth (addzero (lsum (lcpl f g σ l) 0)) (Suc i))
(filt σ (lfuse (lcpl f g σ l)))))
by (simp add: powerinterval-def)
have 04: intlen σ >0 ⟶ intlen (addzero (lsum (lcpl f g σ l) 0)) = intlen l
by (simp add: assms lcpl-lsum-intlen)
have 05: intlen σ >0 ⟶
(∀ i<intlen l.
(filt σ (nth (lcpl f g σ l) i)) =
(sub (nth (addzero (lsum (lcpl f g σ l) 0)) i)
(nth (addzero (lsum (lcpl f g σ l) 0)) (Suc i))
(filt σ (lfuse (lcpl f g σ l)))))

proof −
have 06: intlen σ >0 ⟶
(∀ i<intlen l. intlen (filt σ (nth (lcpl f g σ l) i)) =
intlen (nth (lcpl f g σ l) i))
using filt-intlen **by** blast
have 07: intlen σ >0 ⟶
(∀ i<intlen l. intlen (nth (lcpl f g σ l) i) =
intlen (map (shift (nth l i)) (cpl f g (sub (nth l i) (nth l (Suc i)) σ))))

using assms **by** (metis index-sequence-def lcpl-nth)
have 08: intlen σ >0 ⟶
(∀ i<intlen l.
intlen (map (shift (nth l i)) (cpl f g (sub (nth l i) (nth l (Suc i)) σ))) =
intlen (cpl f g (sub (nth l i) (nth l (Suc i)) σ)))

using interval-intlen-map **by** blast
have 09: intlen σ >0 ⟶
(∀ i<intlen l.
intlen (sub (nth (addzero (lsum (lcpl f g σ l) 0)) i)
(nth (addzero (lsum (lcpl f g σ l) 0)) (Suc i))
(filt σ (lfuse (lcpl f g σ l)))) =
(nth (addzero (lsum (lcpl f g σ l) 0)) (Suc i)) −
(nth (addzero (lsum (lcpl f g σ l) 0)) i))

by (metis 04 assms filt-intlen interval-intlen-sub lcpl-lsum-less-th-equal
lcpl-intlen-nth-gr-zero le-eq-less-or-eq lsum-addzero-nth-leq-Suc)
have 10: intlen σ >0 ⟶
(∀ i<intlen l. (nth (addzero (lsum (lcpl f g σ l) 0)) (Suc i)) =
(nth (0⊙(lsum (lcpl f g σ l) 0)) (Suc i)))

using 04 addzero-def **by** auto

have *11*: *intlen σ >0* ⟶
  (∀ *i<intlen l*. (*nth* (*0⊙*(*lsum* (*lcpl f g σ l*) *0*)) (*Suc i*)) =
   (∑ *k*::*nat*= *0*..(*i*). *intlen*(*nth* (*lcpl f g σ l*) *k*)))
   using *04*
 proof *simp-all*
  assume *0* < *intlen σ* ⟶ *intlen* (*addzero* (*lsum* (*lcpl f g σ l*) *0*)) = *intlen l*
  show *0* < *intlen σ* ⟶
   (∀ *i<intlen l*. *nth* (*lsum* (*lcpl f g σ l*) *0*) *i* =
    (∑ *k* = *0*..*i*. *intlen* (*nth* (*lcpl f g σ l*) *k*)))
  proof
   assume *0* < *intlen σ*
   show ∀ *i<intlen l*. *nth* (*lsum* (*lcpl f g σ l*) *0*) *i* =
    (∑ *k* = *0*..*i*. *intlen* (*nth* (*lcpl f g σ l*) *k*))
   proof
    fix *i*
    show *i* < *intlen l* ⟶
     nth (*lsum* (*lcpl f g σ l*) *0*) *i* = (∑ *k* = *0*..*i*. *intlen* (*nth* (*lcpl f g σ l*) *k*))
    by (*metis* (*no-types*, *lifting*) *0 One-nat-def Suc-leI Suc-le-mono Suc-pred*
     add.left-neutral assms(*1*) *assms*(*4*) *index-sequence-def lcpl-intlen lsum-nth sum.cong*)
   qed
  qed
 qed
have *12*: *intlen σ >0* ⟶
  (∀ *i<intlen l*. (*nth* (*addzero* (*lsum* (*lcpl f g σ l*) *0*)) *i*) =
   (*nth* (*0⊙*(*lsum* (*lcpl f g σ l*) *0*)) *i*))

 using *04 addzero-def* **by** *auto*
have *13*: *intlen σ >0* ⟶
  (∀ *i<intlen l*. (*nth* (*0⊙*(*lsum* (*lcpl f g σ l*) *0*)) *i*) =
  (*case i of 0* ⇒ *0*
   | *Suc j* ⇒ (∑ *k*::*nat*= *0*..(*j*). *intlen*(*nth* (*lcpl f g σ l*) *k*))))
  using *11* **by** (*simp-all add*: *Nitpick.case-nat-unfold*)
have *14*: *intlen σ >0* ⟶
  (∀ *i<intlen l*.
  (*nth* (*addzero* (*lsum* (*lcpl f g σ l*) *0*)) (*Suc i*)) −
  (*nth* (*addzero* (*lsum* (*lcpl f g σ l*) *0*)) *i*) =
  (∑ *k*::*nat*= *0*..(*i*). *intlen*(*nth* (*lcpl f g σ l*) *k*)) −
  (*case i of 0* ⇒ *0*
   | *Suc j* ⇒ (∑ *k*::*nat*= *0*..(*j*). *intlen*(*nth* (*lcpl f g σ l*) *k*))))

 using *10 11 12 13* **by** *auto*
have *15*: *intlen σ >0* ⟶
  (∀ *i<intlen l*.
   (∑ *k*::*nat*= *0*..(*i*). *intlen*(*nth* (*lcpl f g σ l*) *k*)) −
  (*case i of 0* ⇒ *0*
   | *Suc j* ⇒ (∑ *k*::*nat*= *0*..(*j*). *intlen*(*nth* (*lcpl f g σ l*) *k*))) =
   (*case i of 0* ⇒ *intlen*(*nth* (*lcpl f g σ l*) *0*)
    | *Suc j* ⇒ (∑ *k*::*nat*= *0*..(*Suc j*). *intlen*(*nth* (*lcpl f g σ l*) *k*)) −
      (∑ *k*::*nat*= *0*..(*j*). *intlen*(*nth* (*lcpl f g σ l*) *k*))))

by (*simp add*: *Nitpick.case-nat-unfold*)
have *16*: *intlen* σ *>0* $\longrightarrow$
($\forall$ *i<intlen l*.
(*case i of 0* $\Rightarrow$ *intlen*(*nth* (*lcpl f g* σ *l*) *0*)
| *Suc j* $\Rightarrow$ ($\sum$ *k*::*nat*= *0*..(*Suc j*). *intlen*(*nth* (*lcpl f g* σ *l*) *k*)) −
($\sum$ *k*::*nat*= *0*..(*j*). *intlen*(*nth* (*lcpl f g* σ *l*) *k*))) =
(*case i of 0* $\Rightarrow$ *intlen*(*nth* (*lcpl f g* σ *l*) *0*)
| *Suc j* $\Rightarrow$ *intlen*(*nth* (*lcpl f g* σ *l*) (*Suc j*))))

by (*metis* (*no-types*, *lifting*) *Nitpick.case-nat-unfold add-diff-cancel-left'*
sum.atLeast0-atMost-Suc)
have *17*: *intlen* σ *>0* $\longrightarrow$
($\forall$ *i<intlen l*.
(*case i of 0* $\Rightarrow$ *intlen*(*nth* (*lcpl f g* σ *l*) *0*)
| *Suc j* $\Rightarrow$ *intlen*(*nth* (*lcpl f g* σ *l*) (*Suc j*))) =
intlen(*nth* (*lcpl f g* σ *l*) *i*))

by (*simp add*: *Nitpick.case-nat-unfold*)
have *18*: *intlen* σ *>0* $\longrightarrow$
($\forall$ *i<intlen l*. *intlen* (*filt* σ (*nth* (*lcpl f g* σ *l*) *i*)) =
intlen (*sub* (*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*)
(*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) (*Suc i*))
(*filt* σ (*lfuse* (*lcpl f g* σ *l*)))))

by (*simp add*: *09 14 15 16 17 filt-intlen*)
have *19*: *intlen* σ *>0* $\longrightarrow$
($\forall$ *i<intlen l*.
($\forall$ *j*$\leq$ *intlen*(*nth* (*lcpl f g* σ *l*) *i*).
(*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) (*Suc i*)) $\leq$
intlen (*filt* σ (*lfuse* (*lcpl f g* σ *l*)))))

by (*simp add*: *04 assms filt-intlen lcpl-lsum-less-th-equal*)
have *22*: *intlen* σ *>0* $\longrightarrow$ *lastfirst* (*lcpl f g* σ *l*)
using *0 assms index-sequence-def lcpl-lfuse-lastfirst* **by** *blast*
have *23*: *intlen* σ *>0* $\longrightarrow$ ($\forall$ *j* $\leq$ *intlen* (*lcpl f g* σ *l*). *intlen*(*nth* (*lcpl f g* σ *l*) *j*) > *0*)
by (*simp add*: *assms lcpl-intlen-nth-gr-zero*)
have *190*: *intlen* σ *>0* $\longrightarrow$
($\forall$ *i<intlen l*.
($\forall$ *j*$\leq$ *intlen*(*nth* (*lcpl f g* σ *l*) *i*).
(*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*) $\leq$
(*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) (*Suc i*))
))

by (*simp add*: *04 23 less-imp-le-nat lsum-addzero-nth-leq-Suc*)
have *20*: *intlen* σ *>0* $\longrightarrow$
($\forall$ *i<intlen l*.
($\forall$ *j*$\leq$ *intlen*(*nth* (*lcpl f g* σ *l*) *i*).
(*nth* (*sub* (*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*)
(*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) (*Suc i*))
(*filt* σ (*lfuse* (*lcpl f g* σ *l*)))) *j*) =

(*nth* (*filt* σ (*lfuse* (*lcpl f g* σ *l*)))
((*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*) +*j*))))

by (*simp add*: *14 15 16 17 19 190*)
have *21*: *intlen* σ >*0* ⟶
(∀ *i*<*intlen l*.
(∀ *j*≤ *intlen*(*nth* (*lcpl f g* σ *l*) *i*).
(*nth* (*filt* σ (*lfuse* (*lcpl f g* σ *l*)))
((*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*) +*j*)) =
(*nth* σ (*nth* (*lfuse* (*lcpl f g* σ *l*))
((*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*) +*j*)))))

by (*simp add*: *filt-map interval-nth-map*)
have *24*: *intlen* σ >*0* ⟶
(∀ *i*≤*intlen* (*lcpl f g* σ *l*).
(∀ *j*≤ *intlen*(*nth* (*lcpl f g* σ *l*) *i*).
((*nth* (*lfuse* (*lcpl f g* σ *l*)) ((*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*) +*j*))) =
((*nth* (*nth* (*lcpl f g* σ *l*) *i*) *j*))))

by (*simp add*: *22 23 lsum-lfuse-nth-lsum-nth*)
have *241*: *intlen* σ >*0* ⟶ *intlen* (*lcpl f g* σ *l*) = *intlen l* −*1*
using *0 assms index-sequence-def lcpl-intlen* **by** *blast*
have *25*: *intlen* σ >*0* ⟶
(∀ *i*<*intlen l*.
(∀ *j*≤ *intlen*(*nth* (*lcpl f g* σ *l*) *i*).
(*nth* σ (*nth* (*lfuse* (*lcpl f g* σ *l*))
((*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*) +*j*))) =
(*nth* (*filt* σ (*nth* (*lcpl f g* σ *l*) *i*)) *j*)))

by (*simp add*: *06 24 241 assms filt-nth*)
have *26*: *intlen* σ >*0* ⟶
(∀ *i*<*intlen l*.
(∀ *j*≤ *intlen*(*nth* (*lcpl f g* σ *l*) *i*).
(*nth* (*filt* σ (*nth* (*lcpl f g* σ *l*) *i*)) *j*) =
(*nth* (*sub* (*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) *i*)
(*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) (*Suc i*))
(*filt* σ (*lfuse* (*lcpl f g* σ *l*)))) *j*)))

by (*simp add*: *20 21 25*)
from *18 26* **show** *?thesis*
by (*simp add*: *filt-intlen interval-eq-nth-eq*)
qed
show *?thesis*
using *02 03 04 05* **by** (*simp add*: *assms*(*4*))
qed

lemma *lcpl-lfuse-filt-intlen*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen* σ
(∀ *i*<*intlen l*. (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) ⊨ *f* △ *g*)

intlen σ >*0*
shows (*nth* (*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) (*intlen*(*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)))) =
intlen (*filt* σ (*lfuse* (*lcpl f g* σ *l*)))
proof −
have *0*: *intlen* σ > *0* $\longrightarrow$ *intlen l* > *0*
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *1*: *intlen* σ >*0* $\longrightarrow$
intlen (*filt* σ (*lfuse* (*lcpl f g* σ *l*))) = *intlen* ((*lfuse* (*lcpl f g* σ *l*)))
using *filt-intlen* **by** *blast*
have *2*: *intlen* σ >*0* $\longrightarrow$
intlen ((*lfuse* (*lcpl f g* σ *l*))) =
($\sum$ *k*::*nat*= *0*..(*intlen* (*lcpl f g* σ *l*)). *intlen*(*nth* (*lcpl f g* σ *l*) *k*))
using *0 assms index-sequence-def interval-lfuse-intlen lcpl-lfuse-lastfirst* **by** *blast*
have *3*: *intlen* σ >*0* $\longrightarrow$
intlast(*addzero* (*lsum* (*lcpl f g* σ *l*) *0*)) = *intlast*((*lsum* (*lcpl f g* σ *l*) *0*))
using *lsum-addzero-intlast* **by** *blast*
have *4*: *intlen* σ >*0* $\longrightarrow$
intlast((*lsum* (*lcpl f g* σ *l*) *0*)) =
($\sum$ *k*::*nat*= *0*..(*intlen* (*lcpl f g* σ *l*)). *intlen*(*nth* (*lcpl f g* σ *l*) *k*))
by (*metis add-cancel-right-left lsum-intlast*)
show *?thesis* **using** *1 2 3 4* **by** (*simp add*: *assms*(*4*))
qed

10.3 Soundness of Projection Axioms

10.3.1 PJ1

lemma *PJ1sem*:
($\sigma \models f \triangle (g \vee h) \longrightarrow f \triangle g \vee f \triangle h$)
by (*simp add*: *projection-d-def*) *blast*

lemma *PJ1sema*:
($\sigma \models f \triangle (g \vee h) = (f \triangle g \vee f \triangle h)$)
by (*simp add*: *projection-d-def*) *blast*

10.3.2 PJ2

lemma *PJ2sem*:
($\sigma \models f \triangle$ *empty* = *empty*)
proof *auto*
show ($\sigma \models f \triangle$ *empty*) $\Longrightarrow \sigma \models$ *empty*
unfolding *projection-d-def empty-defs index-sequence-def*
by (*metis filt.simps*(*2*) *interval-intlen-cons-1 neq0-conv*)
show $\sigma \models$ *empty* $\Longrightarrow$ ($\sigma \models f \triangle$ *empty*)
unfolding *projection-d-def empty-defs index-sequence-def powerinterval-def*
by (*metis Interval.nth.simps*(*1*) *filt.simps*(*1*) *index-sequence-def intlen.simps*(*1*)
not-less-zero)
qed

10.3.3 PJ3

lemma *PJ3help*:
sub 0 (*intlen* σ) $\sigma = \sigma$
by (*simp add*: *interval-sub-zero-prefix*)

lemma *PJ3help1*:
assumes *f* $\sigma \wedge 0 <$ *intlen* σ
shows ($\exists$ *l*. *index-sequence 0 l* $\wedge$
nth l (*intlen l*) = *intlen* σ $\wedge$
($\forall$ *i*<*intlen l*. *f* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) $\wedge$
($\exists$ *s1*. *intlen s1* = *intlen l* $\wedge$
($\forall$ *i*$\leq$*intlen s1*. *nth s1 i* = *nth* σ (*nth l i*)) $\wedge$ *intlen s1* = *Suc 0*))
proof −
have *1*: *index-sequence 0* $\langle$*0*,*intlen* $\sigma\rangle$
by (*simp add*: *assms index-sequence-def*)
have *2*: *nth* $\langle$*0*,*intlen* $\sigma\rangle$ (*intlen* $\langle$*0*,*intlen* $\sigma\rangle$) = *intlen* σ
by *auto*
have *3*: ($\forall$ *i*<*intlen* $\langle$*0*,*intlen* $\sigma\rangle$. *f* (*sub* (*nth* $\langle$*0*,*intlen* $\sigma\rangle$ *i*) (*nth* $\langle$*0*,*intlen* $\sigma\rangle$ (*Suc i*)) σ))
by (*simp add*: *PJ3help assms*)
have *4*: *intlen* $\langle$*nth* σ (*0*), *nth* σ (*intlen* σ)$\rangle$ = *intlen* $\langle$*0*,*intlen* $\sigma\rangle$
by *simp*
have *5*: ($\forall$ *i*$\leq$*intlen* $\langle$*nth* σ (*0*), *nth* σ (*intlen* σ)$\rangle$.
nth $\langle$*nth* σ (*0*), *nth* σ (*intlen* σ)$\rangle$ *i* = *nth* σ (*nth* $\langle$*0*,*intlen* $\sigma\rangle$ *i*))
using *antisym-conv2* **by** *fastforce*
have *6*: *intlen* $\langle$*nth* σ (*0*), *nth* σ (*intlen* σ)$\rangle$ = *Suc 0*
by *simp*
show *?thesis*
using *1 2 3 4 5 6* **by** *blast*
qed

lemma *PJ3sem*:
($\sigma \models f \triangle$ *skip* = (*f* $\wedge$ *more*))
proof −
have *1*: ($\sigma \models f \triangle$ *skip*) $\Longrightarrow$ ($\sigma \models f \wedge$ *more*)
by (*metis* (*mono-tags*, *lifting*) *One-nat-def PJ3help cpl-projection filt-expand index-sequence-def more-defs powerinterval-def skip-defs unl-lift2 zero-less-one*)
have *2*: ($\sigma \models f \wedge$ *more*) $\Longrightarrow$ ($\sigma \models f \triangle$ *skip*)
by (*simp add*: *projection-d-def skip-defs more-defs powerinterval-def*,
metis PJ3help1 filt-intlen)
show *?thesis*
using *1 2 unl-lift2* **by** *blast*
qed

10.3.4 PJ4

lemma *PJ4semchaina*:
assumes ($\sigma \models f \triangle$ (*g*;*h*))
shows ($\sigma \models$ (*f* $\triangle$ *g*) ; (*f* $\triangle$ *h*))
proof −

```
have 1: (σ ⊨ f △ (g;h))
 using assms by auto
have 2: (∃ l. index-sequence 0 l ∧
        nth l (intlen l) = intlen σ ∧
        powerinterval f σ l ∧
        (∃ n≤intlen (filt σ l). g (prefix n (filt σ l)) ∧ h (suffix n (filt σ l))))
 by (metis assms chop-defs projection-d-def)
 obtain l where 3: index-sequence 0 l ∧
        nth l (intlen l) = intlen σ ∧
        powerinterval f σ l ∧
        (∃ n≤intlen (filt σ l). g (prefix n (filt σ l)) ∧ h (suffix n (filt σ l)))
   using 2 by auto
have 4: index-sequence 0 l
  using 3 by auto
have 5: powerinterval f σ l
  using 3 by auto
have 6: nth l (intlen l) = intlen σ
  using 3 by auto
have 7: (∃ n≤intlen (filt σ l). g (prefix n (filt σ l)) ∧ h (suffix n (filt σ l)))
  using 3 by auto
 obtain n where 8: n≤intlen (filt σ l) ∧ g (prefix n (filt σ l)) ∧ h (suffix n (filt σ l))
 using 7 by auto
have 9: n≤intlen (filt σ l)
 using 8 by auto
have 10:  g (prefix n (filt σ l))
 using 8 by auto
have 11: h (suffix n (filt σ l))
 using 8 by auto
have 12: index-sequence 0 (prefix n l)
  by (metis 4 8 filt-intlen interval-idx-split)
have 13: index-sequence (nth l n) (suffix n l)
  by (metis 4 9 filt-intlen interval-idx-split)
have 14: index-sequence 0 ((map (shiftm (nth l n))  (suffix n l)))
 using 13 interval-idx-shiftm by blast
have 15: g  (filt σ (prefix n l))
  by (metis 8 filt-intlen filt-prefix)
have 16: h (filt σ (suffix n l))
  by (metis 11 9 filt-intlen filt-suffix)
have 17: g  (filt (prefix (nth l n) σ) (prefix n l))
  by (metis (no-types, lifting) 12 15 4 6 8 filt-intlen filt-prefix-idx
     interval-idx-less-equal interval-intlast-prefix interval-nth-intlen-intlast order-refl)
have 18: h (filt (suffix (nth l n) σ) ((map (shiftm (nth l n))  (suffix n l))))
   proof −
    have 181: intlen((filt σ (suffix n l))) =
              intlen(filt (suffix (nth l n) σ) ((map (shiftm (nth l n))  (suffix n l))))
       by (simp add: filt-intlen)
    have 182: (∀ j≤ intlen((filt σ (suffix n l))).
               (nth (filt σ (suffix n l)) j) =
               (nth σ ((nth l (n+j))))
              )
```

by (*metis 9 filt-intlen filt-map interval-nth-map interval-nth-suffix interval-suffix-length-good*)
have *183*: $(\forall j \leq$ *intlen*((*filt* σ (*suffix n l*))).
(*nth* (*filt* (*suffix* (*nth l n*) σ) ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))) *j*) =
(*nth* (*suffix* (*nth l n*) σ) (*nth* (*map* (*shiftm* (*nth l n*)) (*suffix n l*)) *j*))
)
by (*simp add*: *filt-map interval-nth-map*)
have *184*: (*nth l n*) $\leq$ *intlen* σ
by (*metis 4 6 9 filt-intlen interval-idx-less-equal order-refl*)
have *185*: $(\forall j \leq$ *intlen*((*filt* σ (*suffix n l*))).
(*nth* (*map* (*shiftm* (*nth l n*)) (*suffix n l*)) *j*) =
(*nth* (*suffix n l*) *j*) − (*nth l n*)
)
by (*metis 4 6 9 filt-intlen interval-nth-suffix interval-idx-shiftm-suffix-nth interval-suffix-length-good*)
have *186*: $(\forall j \leq$ *intlen*((*filt* σ (*suffix n l*))).
(*nth* (*suffix n l*) *j*) − (*nth l n*) = (*nth l* (*j*+*n*)) − (*nth l n*)
)
by (*metis 9 add.commute filt-intlen interval-nth-suffix interval-suffix-length-good*)
have *187*: $(\forall j \leq$ *intlen*((*filt* σ (*suffix n l*))).
(*nth l* (*j*+*n*)) − (*nth l n*) $\leq$ *intlen* σ − (*nth l n*)
)
by (*metis 4 6 9 add.commute diff-le-mono eq-imp-le filt-intlen interval-idx-less-equal interval-suffix-length-good nat-add-left-cancel-le ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *188*: $(\forall j \leq$ *intlen*((*filt* σ (*suffix n l*))).
(*nth* (*suffix* (*nth l n*) σ) (*nth* (*map* (*shiftm* (*nth l n*)) (*suffix n l*)) *j*)) =
(*nth* σ ((*nth* (*map* (*shiftm* (*nth l n*)) (*suffix n l*)) *j*) + (*nth l n*)))
)
using *interval-nth-suffix*
by (*simp add*: *184 185 186 187 add.commute*)
have *189*: $(\forall j \leq$ *intlen*((*filt* σ (*suffix n l*))).
(*nth* σ ((*nth* (*map* (*shiftm* (*nth l n*)) (*suffix n l*)) *j*) + (*nth l n*))) =
(*nth* σ ((*nth* (*suffix n l*) *j*)))
)
by (*metis 13 185 filt-intlen interval-idx-greater ordered-cancel-comm-monoid-diff-class.le-imp-diff-is-add*)
have *190*: $(\forall j \leq$ *intlen*((*filt* σ (*suffix n l*))).
(*nth* (*filt* σ (*suffix n l*)) *j*) =
(*nth* (*filt* (*suffix* (*nth l n*) σ) ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))) *j*)
)
using *183 188 189 filt-expand* **by** *fastforce*
show *?thesis*
by (*metis 16 181 190 interval-eq-nth-eq*)
qed
have *19*: *powerinterval f* (*prefix* (*nth l n*) σ) (*prefix n l*)
by (*metis 4 5 6 8 filt-intlen powerinterval-splita*)
have *20*: *powerinterval f* (*suffix* (*nth l n*) σ) ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))
by (*metis 4 5 6 9 filt-intlen powerinterval-split*)
have *21*: (*nth l n*) $\leq$ *intlen* σ

by (*metis 3 9 filt-intlen interval-idx-less-equal order-refl*)
have *22*: *nth* (*prefix n l*) (*intlen* (*prefix n l*)) = (*nth l n*)
by (*metis 8 filt-intlen interval-intlast-prefix interval-nth-intlen-intlast*)
have *23*: *nth* ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))
(*intlen* ((*map* (*shiftm* (*nth l n*)) (*suffix n l*)))) = *intlen* σ −(*nth l n*)
by (*metis 4 6 9 eq-imp-le filt-intlen interval-intlen-map interval-idx-shiftm-suffix-nth interval-suffix-length-good ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *24*: *l* = *fuse* (*prefix n l*)
(*map* (*shift* (*nth l n*)) ((*map* (*shiftm* (*nth l n*)) (*suffix n l*))))
using *interval-fuse-prefix-suffix*
by (*metis 13 14 8 filt-intlen interval-lsk-ls*)
have *25*: ($\exists$ *l1*. *index-sequence 0 l1* $\wedge$
Interval.nth l1 (*intlen l1*) = *intlen* (*prefix* (*nth l n*) σ) $\wedge$
powerinterval f (*prefix* (*nth l n*) σ) *l1* $\wedge$ *g* (*filt* (*prefix* (*nth l n*) σ) *l1*))
by (*metis 12 17 19 21 22 interval-prefix-length-good*)
have *26*: ($\exists$ *l2*. *index-sequence 0 l2* $\wedge$
Interval.nth l2 (*intlen l2*) = *intlen* (*suffix* (*nth l n*) σ) $\wedge$
powerinterval f (*suffix* (*nth l n*) σ) *l2* $\wedge$ *h* (*filt* (*suffix* (*nth l n*) σ) *l2*))
using *18 14 20 23 21* **by** *auto*
have *27*: (*prefix* (*nth l n*) σ) $\models$ *f*$\triangle$*g*
by (*metis 25 projection-d-def*)
have *28*: (*suffix* (*nth l n*) σ) $\models$ *f*$\triangle$*h*
by (*metis 26 projection-d-def*)
show *?thesis*
using *21 27 28 chop-defs* **by** *auto*
qed

lemma *PJ4semchainb*:
assumes (σ $\models$ (*f* $\triangle$ *g*) ; (*f* $\triangle$ *h*))
shows (σ $\models$ *f* $\triangle$ (*g*;*h*))
proof −
have *1*: ($\exists$ *n*≤*intlen* σ.
($\exists$ *l*. *index-sequence 0 l* $\wedge$
Interval.nth l (*intlen l*) = *intlen* (*prefix n* σ) $\wedge$
powerinterval f (*prefix n* σ) *l* $\wedge$ *g* (*filt* (*prefix n* σ) *l*)) $\wedge$
($\exists$ *l*. *index-sequence 0 l* $\wedge$
Interval.nth l (*intlen l*) = *intlen* (*suffix n* σ) $\wedge$
powerinterval f (*suffix n* σ) *l* $\wedge$ *h* (*filt* (*suffix n* σ) *l*)))
using *assms* **by** (*metis chop-defs cpl-projection*)
obtain *cp* **where** *2*: *cp*≤*intlen* σ $\wedge$
($\exists$ *l*. *index-sequence 0 l* $\wedge$
Interval.nth l (*intlen l*) = *intlen* (*prefix cp* σ) $\wedge$
powerinterval f (*prefix cp* σ) *l* $\wedge$ *g* (*filt* (*prefix cp* σ) *l*)) $\wedge$
($\exists$ *l*. *index-sequence 0 l* $\wedge$
Interval.nth l (*intlen l*) = *intlen* (*suffix cp* σ) $\wedge$
powerinterval f (*suffix cp* σ) *l* $\wedge$ *h* (*filt* (*suffix cp* σ) *l*))
using *1* **by** *auto*
have *3*: *cp*≤*intlen* σ
using *2* **by** *auto*
have *4*: ($\exists$ *l*. *index-sequence 0 l* $\wedge$

Interval.nth l (intlen l) = intlen (prefix cp σ) ∧
powerinterval f (prefix cp σ) l ∧ g (filt (prefix cp σ) l))
using *2* **by** *auto*
obtain *l1* **where** *5*: *index-sequence 0 l1 ∧*
Interval.nth l1 (intlen l1) = intlen (prefix cp σ) ∧
powerinterval f (prefix cp σ) l1 ∧ g (filt (prefix cp σ) l1)
using *4* **by** *auto*
have *6*: *index-sequence 0 l1*
using *5* **by** *auto*
have *7*: *Interval.nth l1 (intlen l1) = intlen (prefix cp σ)*
using *5* **by** *auto*
have *8*: *powerinterval f (prefix cp σ) l1*
using *5* **by** *auto*
have *9*: *g (filt (prefix cp σ) l1)*
using *5* **by** *auto*
have *10*: *(∃ l. index-sequence 0 l ∧*
Interval.nth l (intlen l) = intlen (suffix cp σ) ∧
powerinterval f (suffix cp σ) l ∧ h (filt (suffix cp σ) l))
using *2* **by** *auto*
obtain *l2* **where** *11*: *index-sequence 0 l2 ∧*
Interval.nth l2 (intlen l2) = intlen (suffix cp σ) ∧
powerinterval f (suffix cp σ) l2 ∧ h (filt (suffix cp σ) l2)
using *10* **by** *auto*
have *12*: *index-sequence 0 l2*
using *11* **by** *auto*
have *13*: *Interval.nth l2 (intlen l2) = intlen (suffix cp σ)*
using *11* **by** *auto*
have *14*: *powerinterval f (suffix cp σ) l2*
using *11* **by** *auto*
have *15*: *h (filt (suffix cp σ) l2)*
using *11* **by** *auto*
have *16*: *index-sequence 0 (fuse l1 (map (shift cp) l2))*
by (*metis 11 12 2 5 6 eq-imp-le interval-idx-fuse-idx interval-prefix-length-good*
interval-suffix-length-good)
have *17*: *intlast l1 = intfirst (map (shift cp) l2)*
by (*metis 12 13 3 6 7 interval-idx-fuse-intfirst-intlast interval-prefix-length-good*
interval-suffix-length-good)
have *18*: *nth (fuse l1 (map (shift cp) l2)) (intlen l1) = cp*
by (*metis 17 3 7 eq-imp-le interval-fuse-intlen-a interval-fuse-nth*
interval-prefix-length-good le-add1)
have *19*: *intlast (fuse l1 (map (shift cp) l2)) = intlen σ*
proof −
have *191*: *intlast (fuse l1 (map (shift cp) l2)) = intlast (map (shift cp) l2)*
by (*metis 17 eq-imp-le interval-fuse-nth-a interval-fuse-intlen-a*
interval-nth-intlen-intlast)
have *192*: *intlast (map (shift cp) l2) = intlast l2 + cp*
by (*metis Interval.shift-def interval-intlen-map interval-nth-intlen-intlast*
interval-nth-map)
have *193*: *intlast l2 + cp = intlen σ*
by (*metis 13 3 Nat.le-imp-diff-is-add interval-nth-intlen-intlast*

interval-suffix-length-good)
show *?thesis* **using** *191 192 193* **by** *auto*
qed
have *20*: *powerinterval f σ* (*fuse l1* (*map* (*shift cp*) *l2*))
using *powerinterval-fuse*[*of l1* (*l2*) *cp σ f*]
using *12 13 14 3 5* **by** *auto*
have *21*: σ = *fuse* (*prefix cp σ*) (*suffix cp σ*)
by (*simp add*: *3 interval-fuse-prefix-suffix*)
have *22*: *nth* ((*fuse l1* (*map* (*shift cp*) *l2*))) (*intlen l1*) = *cp*
using *18* **by** *blast*
have *23*: (*prefix* (*intlen l1*) (*filt σ* (*fuse l1* (*map* (*shift cp*) *l2*)))) =
(*filt* (*prefix cp σ*) *l1*)
by (*metis 17 2 5 filt-prefix filt-prefix-idx interval-fuse-intlen-a*
interval-prefix-fuse interval-prefix-length-good le-add1)
have *24*: *g* (*prefix* (*intlen l1*) (*filt σ* (*fuse l1* (*map* (*shift cp*) *l2*))))
by (*simp add*: *23 9*)
have *25*: (*suffix* (*intlen l1*) (*filt σ* (*fuse l1* (*map* (*shift cp*) *l2*)))) =
(*filt* (*suffix cp σ*) *l2*)
by (*metis 12 13 17 2 23 filt-intlen filt-suffix filt-suffix-idx*
interval-pref-intlen-bound interval-suffix-fuse interval-suffix-length-good)
have *26*: *intlen l1* $\leq$ *intlen* (*filt σ* (*fuse l1* (*map* (*shift cp*) *l2*)))
by (*metis 23 filt-intlen interval-pref-intlen-bound*)
have *27*: (∃ *l*. *index-sequence 0 l* ∧
Interval.nth l (*intlen l*) = *intlen σ* ∧
powerinterval f σ l ∧
(∃ *n*≤*intlen* (*filt σ l*). *g* (*prefix n* (*filt σ l*)) ∧ *h* (*suffix n* (*filt σ l*))))
by (*metis 11 16 19 20 24 25 26 interval-nth-intlen-intlast*)
show *?thesis*
by (*metis 27 interval-chop-fuse interval-fuse-prefix-suffix interval-intlast-intfirst*
projection-d-def)
qed

lemma *PJ4sem*:
($\sigma \models f \bigtriangleup (g;h) = (f \bigtriangleup g) ; (f \bigtriangleup h)$)
using *PJ4semchaina PJ4semchainb unl-lift2* **by** *blast*

10.3.5 PJ5

lemma *PJ5sem*:
($\sigma \models f \bigtriangleup$ *init*(g) $\longrightarrow$ *init*(g))
by (*simp add*: *projection-d-def init-defs*)
(*metis filt-nth filt-intlen index-sequence-def interval-intlen-gr-zero*)

10.3.6 PJ6

lemma *PJ6help1*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = (*intlen σ*)
shows (∀ *i*. *0*≤ *i* ∧ *i*<*intlen l* ⟶ *intlen* (*sub* (*nth l i*) (*nth l* (*Suc i*)) *σ*)
= (*nth l* (*Suc i*)) − (*nth l i*))
proof

fix *i*
show $0 \leq i \wedge i < intlen\ l \longrightarrow$
$intlen\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) = nth\ l\ (Suc\ i) - nth\ l\ i$
using *assms*
by (*simp add*: *index-sequence-def sub-def*)
(*metis Suc-lessI assms*(*1*) *interval-idx-less-last-1 le-diff-iff le-eq-less-or-eq min.orderE*)
qed

lemma *PJ6help2*:
assumes *index-sequence 0 l*
$nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
$(\forall i{<}intlen\ l.\ nth\ l\ (Suc\ i) - nth\ l\ i = Suc\ 0)$
shows $(\forall i{\leq}intlen\ l.\ nth\ l\ i = i)$
proof
fix *i*
show $i{\leq}intlen\ l \longrightarrow nth\ l\ i = i$
proof
(*induct i*)
case *0*
then show *?case* **using** *assms index-sequence-def* **by** *blast*
next
case (*Suc i*)
then show *?case*
by (*metis One-nat-def Suc-eq-plus1 Suc-leD Suc-le-lessD assms interval-idx-expand le-add-diff-inverse2 plus-1-eq-Suc*)
qed
qed

lemma *PJ6help3*:
assumes *index-sequence 0 l*
$nth\ l\ (intlen\ l) = intlen\ \sigma$
$(\forall i{\leq}intlen\ l.\ nth\ l\ i = i)$
shows $(\forall i{<}intlen\ l.\ nth\ l\ (Suc\ i) - nth\ l\ i = Suc\ 0)$

proof
fix *i*
show $i{<}intlen\ l \longrightarrow nth\ l\ (Suc\ i) - nth\ l\ i = Suc\ 0$
by (*simp add*: *assms*)
qed

lemma *PJ6help4*:
$(\exists l.\ index\text{-}sequence\ 0\ l \wedge l = [0..{\leq}intlen\ \sigma] \wedge$
$Interval.nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
$(\forall i{\leq}intlen\ l.\ nth\ l\ i = i) \wedge$
$(\exists s1.\ intlen\ s1 = intlen\ l \wedge (\forall i{\leq}intlen\ s1.\ nth\ s1\ i = nth\ \sigma\ (i)) \wedge s1 = \sigma))$

by (*simp add*: *index-sequence-def upt-length upt-nth*)

lemma *PJ6help5*:
 $(\exists l.\ \textit{index-sequence}\ 0\ l \wedge$
 $\quad \textit{Interval.nth}\ l\ (\textit{intlen}\ l) = \textit{intlen}\ \sigma \wedge$
 $\quad (\forall i{<}\textit{intlen}\ l.\ \textit{intlen}\ (\textit{sub}\ (\textit{nth}\ l\ i)\ (\textit{nth}\ l\ (\textit{Suc}\ i))\ \sigma) = \textit{Suc}\ 0) \wedge$
 $\quad (\exists s1.\ \textit{intlen}\ s1 = \textit{intlen}\ l \wedge (\forall i{\le}\textit{intlen}\ s1.\ \textit{nth}\ s1\ i = \textit{nth}\ \sigma\ (\textit{Interval.nth}\ l\ i)) \wedge g\ s1))$
 $= g\ \sigma$
proof −
 have $(\exists l.\ \textit{index-sequence}\ 0\ l \wedge$
 $\quad \textit{Interval.nth}\ l\ (\textit{intlen}\ l) = \textit{intlen}\ \sigma \wedge$
 $\quad (\forall i{<}\textit{intlen}\ l.\ \textit{intlen}\ (\textit{sub}\ (\textit{nth}\ l\ i)\ (\textit{nth}\ l\ (\textit{Suc}\ i))\ \sigma) = \textit{Suc}\ 0) \wedge$
 $\quad (\exists s1.\ \textit{intlen}\ s1 = \textit{intlen}\ l \wedge (\forall i{\le}\textit{intlen}\ s1.\ \textit{nth}\ s1\ i = \textit{nth}\ \sigma\ (\textit{nth}\ l\ i)) \wedge g\ s1))$
 $=$
 $(\exists l.\ \textit{index-sequence}\ 0\ l \wedge$
 $\quad \textit{Interval.nth}\ l\ (\textit{intlen}\ l) = \textit{intlen}\ \sigma \wedge$
 $\quad (\forall i{<}\textit{intlen}\ l.\ \textit{nth}\ l\ (\textit{Suc}\ i) - \textit{nth}\ l\ i = \textit{Suc}\ 0) \wedge$
 $\quad (\exists s1.\ \textit{intlen}\ s1 = \textit{intlen}\ l \wedge (\forall i{\le}\textit{intlen}\ s1.\ \textit{nth}\ s1\ i = \textit{nth}\ \sigma\ (\textit{Interval.nth}\ l\ i)) \wedge g\ s1))$
 using *PJ6help1* **by** (*metis zero-order*(*1*))
 also have ... =
 $(\exists l.\ \textit{index-sequence}\ 0\ l \wedge$
 $\quad \textit{Interval.nth}\ l\ (\textit{intlen}\ l) = \textit{intlen}\ \sigma \wedge$
 $\quad (\forall i{\le}\textit{intlen}\ l.\ \textit{nth}\ l\ i = i) \wedge$
 $\quad (\exists s1.\ \textit{intlen}\ s1 = \textit{intlen}\ l \wedge (\forall i{\le}\textit{intlen}\ s1.\ \textit{nth}\ s1\ i = \textit{nth}\ \sigma\ (\textit{nth}\ l\ i)) \wedge g\ s1))$

 using *PJ6help2 PJ6help3* **by** *blast*
 also have ... =
 $(\exists l.\ \textit{index-sequence}\ 0\ l \wedge$
 $\quad \textit{Interval.nth}\ l\ (\textit{intlen}\ l) = \textit{intlen}\ \sigma \wedge$
 $\quad (\forall i{\le}\textit{intlen}\ l.\ \textit{nth}\ l\ i = i) \wedge$
 $\quad (\exists s1.\ \textit{intlen}\ s1 = \textit{intlen}\ l \wedge (\forall i{\le}\textit{intlen}\ s1.\ \textit{nth}\ s1\ i = \textit{nth}\ \sigma\ (i)) \wedge g\ s1))$

 by *metis*
 also have ... =
 $(\exists l.\ \textit{index-sequence}\ 0\ l \wedge$
 $\quad \textit{Interval.nth}\ l\ (\textit{intlen}\ l) = \textit{intlen}\ \sigma \wedge$
 $\quad (\forall i{\le}\textit{intlen}\ l.\ \textit{nth}\ l\ i = i) \wedge$
 $\quad (\exists s1.\ \textit{intlen}\ s1 = \textit{intlen}\ l \wedge (\forall i{\le}\textit{intlen}\ s1.\ \textit{nth}\ s1\ i = \textit{nth}\ \sigma\ (i)) \wedge s1 = \sigma \wedge g\ \sigma))$

 by (*metis interval-eq-nth-eq le-eq-less-or-eq*)
 also have ... =
 $g\ \sigma$

 using *PJ6help4* **by** *blast*
 finally show $(\exists l.\ \textit{index-sequence}\ 0\ l \wedge$
 $\quad \textit{Interval.nth}\ l\ (\textit{intlen}\ l) = \textit{intlen}\ \sigma \wedge$
 $\quad (\forall i{<}\textit{intlen}\ l.\ \textit{intlen}\ (\textit{sub}\ (\textit{nth}\ l\ i)\ (\textit{nth}\ l\ (\textit{Suc}\ i))\ \sigma) = \textit{Suc}\ 0) \wedge$
 $\quad (\exists s1.\ \textit{intlen}\ s1 = \textit{intlen}\ l \wedge (\forall i{\le}\textit{intlen}\ s1.\ \textit{nth}\ s1\ i = \textit{nth}\ \sigma\ (\textit{Interval.nth}\ l\ i)) \wedge g\ s1))$
 $= g\ \sigma$

.
qed

lemma *PJ6sem*:
$(\sigma \models skip \triangle g = g)$
proof –
 have *1*: $(\sigma \models skip \triangle g = g) =$
 $((\exists l.\ index\text{-}sequence\ 0\ l \wedge nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
 $(\forall i<intlen\ l.\ \ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models skip) \wedge g\ (filt\ \sigma\ l)) =$
 $g\ \sigma)$
 by (*simp add*: *projection-d-def powerinterval-def*)
 have *2*: $(\exists l.\ index\text{-}sequence\ 0\ l \wedge nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
 $(\forall i<intlen\ l.\ \ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models skip) \wedge g\ (filt\ \sigma\ l)) =$
 $(\exists l\ s1\ .\ index\text{-}sequence\ 0\ l \wedge nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
 $(\forall i<intlen\ l.\ \ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models skip) \wedge$
 $intlen\ s1 = intlen\ l \wedge$
 $(\forall\ \ i\leq intlen\ s1.\ (nth\ s1\ i) = (nth\ \sigma\ (nth\ l\ i))) \wedge$
 $g\ s1)$
 using *filt-expand* **by** *metis*
 have *3*: $(\exists l\ s1\ .\ index\text{-}sequence\ 0\ l \wedge nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
 $(\forall i<intlen\ l.\ \ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models skip) \wedge$
 $intlen\ s1 = intlen\ l \wedge$
 $(\forall\ \ i\leq intlen\ s1.\ (nth\ s1\ i) = (nth\ \sigma\ (nth\ l\ i))) \wedge$
 $g\ s1) =$
 $(\exists l.\ index\text{-}sequence\ 0\ l \wedge nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
 $(\forall i<intlen\ l.\ intlen\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) = Suc\ 0) \wedge$
 $(\exists s1.\ intlen\ s1 = intlen\ l \wedge (\forall i\leq intlen\ s1.\ nth\ s1\ i = nth\ \sigma\ (nth\ l\ i)) \wedge g\ s1))$
 by (*simp add*: *skip-defs*)
 have *4*: $(\exists l.\ index\text{-}sequence\ 0\ l \wedge nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
 $(\forall i<intlen\ l.\ intlen\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) = Suc\ 0) \wedge$
 $(\exists s1.\ intlen\ s1 = intlen\ l \wedge (\forall i\leq intlen\ s1.\ nth\ s1\ i = nth\ \sigma\ (nth\ l\ i)) \wedge g\ s1)) =$
 $(g\ \sigma)$
 by (*simp add*: *PJ6help5*)
 from *1 2 3 4* **show** *?thesis*
 by *simp*
qed

10.3.7 PJ7

lemma *PJemptyImp*:
assumes $intlen\ \sigma = 0$
 shows $(\sigma \models (f\ \triangle\ g) = g)$
using *assms*
by (*simp add*: *projection-d-def index-sequence-def powerinterval-def*,
 auto,
 metis filt.simps(1) interval-st-intlen lessI not-less-zero old.nat.exhaust,
 metis Interval.nth.simps(1) filt.simps(1) interval-suffix-intlast interval-suffix-zero
 intlen.simps(1) not-less-zero)

lemma *PJ7empty*:

```
 assumes  intlen σ = 0
 shows   (σ ⊨ f △ (g △ h) = (f △ g) △ h)
proof −
 have 1: (σ ⊨ f △ (g △ h) = (g △ h))
   using PJemptyImp assms by blast
 have 2: (σ ⊨ (g △ h) = h)
   using PJemptyImp assms by blast
 have 3: (σ ⊨ (f △ g) △ h = h)
   using PJemptyImp assms by blast
 from 1 2 3 show ?thesis by simp
qed

lemma PJ7helpchain1a-help-1:
 assumes index-sequence 0 l
        (nth l (intlen l)) = intlen σ
         (∀ i<intlen l. (sub (nth l i) (nth l (Suc i)) σ) ⊨ f △ g)
         intlen σ > 0
 shows    index-sequence 0  (lfuse ( (lcpl f g σ l)))
proof −
 have 0:  intlen σ > 0 ⟶ intlen l >0
    using assms gr-zeroI index-sequence-def by fastforce
 have 01: intfirst l = 0
   using assms index-sequence-def by auto
 have 02: lastfirst (lcpl f g σ l)
    using assms lcpl-lfuse-lastfirst by (simp add: projection-d-def)
     (metis 0 01 assms(3) interval-nth-zero-intfirst lcpl-lfuse-lastfirst)
 have 1: intlen σ > 0 ⟶ intfirst (lfuse  (lcpl f g σ l)) = 0
   using assms 0  01  02
     lcpl-intfirst[of l σ  f g ]
    by (metis (mono-tags, lifting)  interval-lastfirst-lfuse interval-nth-zero-intfirst  )
 from 1 0 show ?thesis using assms lcpl-lfuse-idx[of l σ  f g] by(simp add: projection-d-def)
qed

lemma PJ7helpchain1a-help-2:
 assumes index-sequence 0 l
         (nth l (intlen l)) = intlen σ
         (∀ i<intlen l. (sub (nth l i) (nth l (Suc i)) σ) ⊨ f △ g)
         intlen σ > 0
 shows    powerinterval f  σ (lfuse ( (lcpl f g σ l)))
proof −
 have 1: (∀ i<intlen l.
            powerinterval f  (sub (nth l i) (nth l (Suc i)) σ)
                           (cpl f g (sub (nth l i) (nth l (Suc i)) σ)))
   using assms cpl-projection by blast
 have 2: ∀ i<intlen l.
          ∀ ia<intlen (cpl f g (sub (nth l i) (nth l (Suc i)) σ)).
           f  (sub (nth (cpl f g (sub (nth l i) (nth l (Suc i)) σ)) ia)
                 (nth (cpl f g (sub (nth l i) (nth l (Suc i)) σ)) (Suc ia))
                 (sub (nth l i) (nth l (Suc i)) σ))
   using 1 by (simp add: powerinterval-def)
```

have *3*: ∀ *i*<*intlen l*.
 ∀ *ia*<*intlen* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)).
 (*sub* (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *ia*)
 (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) (*Suc ia*))
 (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) =
 (*sub* (*nth* (*map* (*shift* (*nth l i*)) (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))) *ia*)
 (*nth* (*map* (*shift* (*nth l i*)) (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))) (*Suc ia*))
 σ)
proof −
have *30*: ∀ *i*<*intlen l*.
 ∀ *ia*<*intlen* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)).
 ∀ *j*≤*intlen* (*sub* (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *ia*)
 (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) (*Suc ia*))
 (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)).
 (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) (*Suc ia*)) ≤
 (*nth l* (*Suc i*)) − (*nth l* (*i*))
using *assms* **by** (*metis add.commute cpl-projection interval-idx-expand interval-intlen-sub plus-1-eq-Suc*)
have *31*: ∀ *i*<*intlen l*.
 ∀ *ia*<*intlen* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)).
 intlen (*sub* (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *ia*)
 (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) (*Suc ia*))
 (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) =
 intlen (*sub* (*nth* (*map* (*shift* (*nth l i*))
 (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))) *ia*)
 (*nth* (*map* (*shift* (*nth l i*))
 (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))) (*Suc ia*))
 σ)
using *assms* **by** (*auto simp add*: *shift-def interval-nth-map*)
 (*metis* (*no-types*, *lifting*) *PJ6help1 add.commute cpl-projection interval-idx-expand interval-sub-sub-1 le-add1 le-add-same-cancel1 plus-1-eq-Suc*)
have *32*: ∀ *i*<*intlen l*.
 ∀ *ia*<*intlen* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)).
 ∀ *j*≤*intlen* (*sub* (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *ia*)
 (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) (*Suc ia*))
 (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)).
 nth (*sub* (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *ia*)
 (*nth* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) (*Suc ia*))
 (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)) *j* =
 nth (*sub* (*nth* (*map* (*shift* (*nth l i*))
 (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))) *ia*)
 (*nth* (*map* (*shift* (*nth l i*))
 (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ))) (*Suc ia*))
 σ) *j*
using *assms* **by** (*simp add*: *shift-def interval-nth-map cpl-projection*)
 (*metis 30 add.commute interval-idx-expand interval-sub-sub-1 plus-1-eq-Suc*)
show *?thesis* **using** *31 32 interval-eq-nth-eq* **by** (*simp add*: *interval-eq-nth-eq*)
qed
have *4*: ∀ *i*<*intlen l*.
 ∀ *ia*<*intlen* (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ)).

f (*sub* (*nth* (*map* (*shift* (*nth l i*)) (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) *σ*))) *ia*)
(*nth* (*map* (*shift* (*nth l i*)) (*cpl f g* (*sub* (*nth l i*) (*nth l* (*Suc i*)) *σ*))) (*Suc ia*))
σ)
using *2 3* **by** *auto*
have *5*: *intlen*(*lcpl f g σ l*) = *intlen l −1*
using *assms index-sequence-def lcpl-intlen lcpl-intlen-zero* **by** *fastforce*
have *6*: *intlen σ >0 ⟶ intlen l >0*
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *7*: ∀ *i*< *intlen l*.
∀ *ia*<*intlen* ((*nth* (*lcpl f g σ l*) *i*)).
f (*sub* (*nth* (*nth* (*lcpl f g σ l*) *i*) *ia*)
(*nth* (*nth* (*lcpl f g σ l*) *i*) (*Suc ia*))
σ)
using *assms* **by** (*metis* (*no-types*, *lifting*) *4 index-sequence-def interval-intlen-map lcpl-nth*)
have *8*: *intlen σ >0 ⟶*
(∀ *i*≤ *intlen*(*lcpl f g σ l*).
∀ *ia*<*intlen* ((*nth* (*lcpl f g σ l*) *i*)).
f (*sub* (*nth* (*nth* (*lcpl f g σ l*) *i*) *ia*)
(*nth* (*nth* (*lcpl f g σ l*) *i*) (*Suc ia*))
σ))
by (*metis 5 6 7 One-nat-def Suc-pred le-imp-less-Suc*)
have *9*: *intlen σ >0 ⟶*
powerinterval f σ (*lfuse* ((*lcpl f g σ l*))) =
(∀ *i*<*intlen* (*lfuse* (*lcpl f g σ l*)).
f (*sub* (*nth* (*lfuse* (*lcpl f g σ l*)) *i*) (*nth* (*lfuse* (*lcpl f g σ l*)) (*Suc i*)) *σ*))
by (*simp add*: *powerinterval-def*)
have *10*: *intlen σ >0 ⟶ lastfirst* (*lcpl f g σ l*)
using *6 assms index-sequence-def lcpl-lfuse-lastfirst* **by** *blast*
have *11*: *intlen σ >0 ⟶*
(∀ *j* ≤ *intlen* (*lcpl f g σ l*). *intlen*(*nth* (*lcpl f g σ l*) *j*) > *0*)

by (*simp add*: *assms lcpl-intlen-nth-gr-zero*)
have *12*: *intlen σ >0 ⟶*
(∀ *i*≤ *intlen*(*lcpl f g σ l*).
(∀ *ia*<*intlen* ((*nth* (*lcpl f g σ l*) *i*)).
f (*sub* (*nth* (*nth* (*lcpl f g σ l*) *i*) *ia*)
(*nth* (*nth* (*lcpl f g σ l*) *i*) (*Suc ia*))
σ))) =
(∀ *j*<*intlen* (*lfuse* (*lcpl f g σ l*)).
f (*sub* (*nth* (*lfuse* (*lcpl f g σ l*)) *j*)
(*nth* (*lfuse* (*lcpl f g σ l*)) (*Suc j*))
σ))

using *interval-lfuse-split*[*of* (*lcpl f g σ l*) *f σ*] *10 11* **by** *auto*
show *?thesis* **using** *12 8 9* **using** *assms*(*4*) **by** *blast*
qed

lemma *PJ7helpchain1a-help-3*:
assumes *index-sequence 0 l*
(*nth l* (*intlen l*)) = *intlen σ*

$(\forall\ i<intlen\ l.\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models f \triangle g)$
$h\ (filt\ \sigma\ l)$
$intlen\ \sigma > 0$
shows $intlast\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)) = intlen\ \sigma$
proof −
have *0*: $intlen\ \sigma > 0 \longrightarrow intlen\ l > 0$
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *1*: $intlen\ \sigma > 0 \longrightarrow$
$intlast\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)) = intlast(intlast\ (\ (lcpl\ f\ g\ \sigma\ l)))$
using *assms*
using *0 index-sequence-def interval-lastfirst-lfuse-intlast lcpl-lfuse-lastfirst* **by** *blast*
have *2*: $intlen\ \sigma > 0 \longrightarrow$
$intlast\ (\ (lcpl\ f\ g\ \sigma\ l))$
$= (nth\ (lcpl\ f\ g\ \sigma\ l)\ (intlen\ l\ - 1))$
using *assms* **by** (*simp add*: *0 index-sequence-def lcpl-intlen*)
have *3*: $intlen\ \sigma > 0 \longrightarrow$
$(nth\ (lcpl\ f\ g\ \sigma\ l)\ (intlen\ l\ - 1)) =$
$(map\ (shift\ (nth\ l\ (intlen\ l\ - 1)))$
$(cpl\ f\ g\ (sub\ (nth\ l\ (intlen\ l\ - 1))\ (nth\ l\ (Suc\ (intlen\ l\ - 1)))\ \sigma)))$

using *assms* **by** (*simp add*: *0 index-sequence-def lcpl-nth*)
have *4*: $intlen\ \sigma > 0 \longrightarrow$
$intlast(\ (\ (cpl\ f\ g\ (sub\ (nth\ l\ (intlen\ l\ - 1))\ (nth\ l\ (Suc\ (intlen\ l\ - 1)))\ \sigma)))) =$
$intlen\ ((sub\ (nth\ l\ (intlen\ l\ - 1))\ (nth\ l\ (Suc\ (intlen\ l\ - 1)))\ \sigma))$
using *0 assms*
by (*metis cpl-projection diff-less interval-nth-intlen-intlast zero-less-one*)
have *5*: $intlen\ \sigma > 0 \longrightarrow$
$intlen\ ((sub\ (nth\ l\ (intlen\ l\ - 1))\ (nth\ l\ (Suc\ (intlen\ l\ - 1)))\ \sigma)) =$
$(nth\ l\ (Suc\ (intlen\ l\ - 1))) - (nth\ l\ (intlen\ l\ - 1))$

using *0 PJ6help1 assms diff-less zero-less-one* **by** *blast*
have *6*: $intlen\ \sigma > 0 \longrightarrow$
$intlast\ (\ (map\ (shift\ (nth\ l\ (intlen\ l\ - 1)))$
$(cpl\ f\ g\ (sub\ (nth\ l\ (intlen\ l\ - 1))\ (nth\ l\ (Suc\ (intlen\ l\ - 1)))\ \sigma)))) =$
$(shift\ (nth\ l\ (intlen\ l\ - 1)))$
$(intlast\ (cpl\ f\ g\ (sub\ (nth\ l\ (intlen\ l\ - 1))\ (nth\ l\ (Suc\ (intlen\ l\ - 1)))\ \sigma)))$

by (*metis interval-intlen-map interval-nth-intlen-intlast interval-nth-map*)
have *7*: $intlen\ \sigma > 0 \longrightarrow$
$(shift\ (nth\ l\ (intlen\ l\ - 1)))$
$(intlast\ (cpl\ f\ g\ (sub\ (nth\ l\ (intlen\ l\ - 1))\ (nth\ l\ (Suc\ (intlen\ l\ - 1)))\ \sigma))) =$
$(shift\ (nth\ l\ (intlen\ l\ - 1)))\ ((nth\ l\ (Suc\ (intlen\ l\ - 1))) - (nth\ l\ (intlen\ l\ - 1)))$

using *4 5* **by** *auto*
have *8*: $intlen\ \sigma > 0 \longrightarrow$
$(shift\ (nth\ l\ (intlen\ l\ - 1)))\ ((nth\ l\ (Suc\ (intlen\ l\ - 1))) - (nth\ l\ (intlen\ l\ - 1))) =$
$((nth\ l\ (Suc\ (intlen\ l\ - 1))) - (nth\ l\ (intlen\ l\ - 1))) + (nth\ l\ (intlen\ l\ - 1))$

by (*simp add*: *Interval.shift-def*)
have *9*: $intlen\ \sigma > 0 \longrightarrow$

$((nth\ l\ (Suc\ (intlen\ l\ -1))) - (nth\ l\ (intlen\ l\ -1))) + (nth\ l\ (intlen\ l\ -1))$
$= (nth\ l\ (Suc\ (intlen\ l\ -1)))$

using *assms*
by (*metis* (*no-types*, *lifting*) *0 2 3 6 7 8 Suc-pred′ index-sequence-def lcpl-last-last lessI*)
have *10*: $intlen\ \sigma > 0 \longrightarrow (nth\ l\ (Suc\ (intlen\ l\ -1))) = intlen\ \sigma$
by (*simp add*: *0 assms*)
show *?thesis*
using *1 10 2 3 6 7 8 9 assms*(*5*) **by** *presburger*
qed

lemma *PJ7helpchain1a-help-4*:
assumes *index-sequence 0 l*
$(nth\ l\ (intlen\ l)) = intlen\ \sigma$
$(\forall\ i{<}intlen\ l.\ (sub\ (nth\ l\ i)\ (nth\ l\ (Suc\ i))\ \sigma) \models f \bigtriangleup g)$
$h\ (filt\ \sigma\ l)$
$intlen\ \sigma > 0$
shows $((filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l))) \models g \bigtriangleup h)$
proof −
have *0*: $intlen\ \sigma > 0 \longrightarrow intlen\ l > 0$
using *assms gr-zeroI index-sequence-def* **by** *fastforce*
have *1*: $intlen\ \sigma > 0 \longrightarrow index\text{-}sequence\ 0\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))$
by (*simp add*: *assms lcpl-intlen-nth-gr-zero lsum-addzero-idx*)
have *2*: $intlen\ \sigma > 0 \longrightarrow$
$(nth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (intlen(addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0)))) =$
$intlen\ (filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))$
by (*simp add*: *assms lcpl-lfuse-filt-intlen*)
have *3*: $intlen\ \sigma > 0 \longrightarrow$
$powerinterval\ g\ (filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))$

by (*simp add*: *assms lcpl-lfuse-filt-power*)
have *4*: $intlen\ \sigma > 0 \longrightarrow$
$h\ (filt\ (filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0)))$
by (*simp add*: *assms filt-map-filt lcpl-filt-lfuse-lsum*)
show *?thesis* **by** (*metis 1 2 3 4 assms*(*5*) *projection-d-def*)
qed

lemma *PJ7helpchain1a*:
assumes $intlen\ \sigma > 0$
$(\sigma \models (f \bigtriangleup g) \bigtriangleup h)$
shows $(\sigma \models f \bigtriangleup (g \bigtriangleup h))$
proof −
have *1*: $intlen\ \sigma > 0$
using *assms* **by** *auto*
have *2*: $(\exists\ l.\ index\text{-}sequence\ 0\ l \wedge nth\ l\ (intlen\ l) = intlen\ \sigma \wedge$
$powerinterval\ (LIFT(f \bigtriangleup g))\ \sigma\ l \wedge$
$h\ (filt\ \sigma\ l))$
using *assms* **using** *cpl-projection* **by** *blast*

obtain *l* **where** *3*: *index-sequence 0 l* $\wedge$ *nth l* (*intlen l*) = *intlen* σ $\wedge$
powerinterval (*LIFT*(*f* $\triangle$ *g*)) σ *l* $\wedge$
h (*filt* σ *l*)
using *2* **by** *blast*
have *4*: *index-sequence 0 l* $\wedge$ *nth l* (*intlen l*) = *intlen* σ $\wedge$
($\forall$ *i* <*intlen l*. (*sub* (*nth l i*) (*nth l* (*Suc i*)) σ) $\models$ *f* $\triangle$ *g*) $\wedge$
h (*filt* σ *l*)
using *3* **by** (*simp add*: *powerinterval-def*)
have *6*: *intlen l*>*0*
using *1 4 gr0I index-sequence-def* **by** *force*
have *7*: *index-sequence 0* (*lfuse* (*lcpl f g* σ *l*))
by (*metis* (*no-types*, *lifting*) *1 4 PJ7helpchain1a-help-1*)
have *8*: *powerinterval f* σ (*lfuse* (*lcpl f g* σ *l*))
using *4 6 PJ7helpchain1a-help-2 assms index-sequence-def* **by** *auto*
have *9*: (*nth* (*lfuse* (*lcpl f g* σ *l*)) (*intlen* (*lfuse* (*lcpl f g* σ *l*)))) = *intlen* σ
by (*metis 1 4 interval-nth-intlen-intlast PJ7helpchain1a-help-3*)
have *10*: ((*filt* σ (*lfuse* (*lcpl f g* σ *l*))) $\models$ *g* $\triangle$ *h*)
by (*simp add*: *1 4 6 PJ7helpchain1a-help-4*)
show *?thesis*
using *10 7 8 9* **by** (*metis projection-d-def*)
qed

lemma *PJ7helpchain1b*:
assumes *intlen* σ > *0*
(σ $\models$ *f* $\triangle$ (*g* $\triangle$ *h*))
shows (σ $\models$ (*f* $\triangle$ *g*) $\triangle$ *h*)
proof −
have *1*: *intlen* σ > *0*
using *assms* **by** *auto*
have *2*: ($\exists$ *l*. *index-sequence 0 l* $\wedge$ *nth l* (*intlen l*) = *intlen* σ $\wedge$
powerinterval f σ *l* $\wedge$
((*filt* σ *l*) $\models$ *g* $\triangle$ *h*))
using *assms* **by** (*simp add*: *projection-d-def*)
obtain *l* **where** *3*: *index-sequence 0 l* $\wedge$ *nth l* (*intlen l*) = *intlen* σ $\wedge$
powerinterval f σ *l* $\wedge$
((*filt* σ *l*) $\models$ *g* $\triangle$ *h*)
using *2* **by** *blast*
have *4*: ($\exists$ *la*. *index-sequence 0 la* $\wedge$ *nth la* (*intlen la*) = *intlen*(*filt* σ *l*) $\wedge$
powerinterval g (*filt* σ *l*) *la* $\wedge$
((*filt* (*filt* σ *l*) *la*) $\models$ *h*))
using *3* **using** *cpl-projection* **by** *blast*
obtain *la* **where** *5*: *index-sequence 0 la* $\wedge$ *nth la* (*intlen la*) = *intlen*(*filt* σ *l*) $\wedge$
powerinterval g (*filt* σ *l*) *la* $\wedge$
((*filt* (*filt* σ *l*) *la*) $\models$ *h*)
using *4* **by** *blast*
have *6*: *intlen l* >*0*
using *1 3 gr0I index-sequence-def* **by** *force*
have *7*: *intlen*(*filt* σ *l*) = *intlen l*
by (*simp add*: *filt-intlen*)

have *8*: (*filt* (*filt* σ *l*) *la*) = (*filt* σ (*filt l la*))
using *filt-map-filt* **by** *blast*
have *9*: (*nth* (*filt l la*) (*intlen* (*filt l la*))) = *intlen* σ
by (*metis 3 5 filt-expand order-refl*)
have *10*: *intlen la* >*0*
using *5 6 7 gr0I index-sequence-def* **by** *force*
have *11*: *intlen* (*filt l la*) > *0*
by (*simp add*: *10 filt-intlen*)
have *12*: *index-sequence 0* (*filt l la*)
proof −
have *111*: *nth* (*filt l la*) *0* = *0*
by (*metis 3 5 filt-nth index-sequence-def interval-intlen-gr-zero*)
have *112*: *intlen*(*filt l la*) = *intlen la*
using *filt-expand* **by** *blast*
have *113*: ($\forall$ *i*< *intlen la*. (*nth* (*filt l la*) *i*) = (*nth l* (*nth la i*)))
by (*simp add*: *filt-map interval-nth-map*)
have *114*: ($\forall$ *i*< *intlen la*. (*nth* (*filt l la*) (*Suc i*)) = (*nth l* (*nth la* (*Suc i*))))
by (*simp add*: *filt-map interval-nth-map*)
have *115*: ($\forall$ *i*< *intlen la*. (*nth l* (*nth la i*)) < (*nth l* (*nth la* (*Suc i*))))
by (*metis 3 5 7 Suc-lessI interval-idx-less-than interval-idx-less-last-1 lessI less-imp-le-nat*)
show *?thesis* **by** (*simp add*: *111 113 114 115 filt-intlen index-sequence-def*)
qed
have *20*: *powerinterval* (*LIFT*($f\triangle g$)) σ (*filt l la*)
proof −
have *201*: *powerinterval* (*LIFT*($f\triangle g$)) σ (*filt l la*) =
($\forall$ *i*< *intlen la*. (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ) $\models f \triangle g$)
by (*simp add*: *filt-map interval-nth-map powerinterval-def*)
have *202*: ($\forall$ *i*< *intlen la*. (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ) $\models f \triangle g$) =
($\forall$ *i*< *intlen la*.
($\exists$ *ll*. *index-sequence 0 ll*
$\wedge$ *intlast ll* = *intlen* (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ) $\wedge$
powerinterval f (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ) *ll* $\wedge$
((*filt* (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ) *ll*) $\models g$)
))

by (*simp add*: *projection-d-def*)
have *203*: ($\forall$ *i*< *intlen la*. *intlen* (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ) =
(*nth l* (*nth la* (*Suc i*))) − (*nth l* (*nth la i*)))
by (*metis 3 5 7 Suc-leI Suc-lessI interval-idx-less-equal interval-idx-less-last-1 interval-intlen-sub lessI less-imp-le-nat*)
have *2041*: ($\forall$ *i*< *intlen la*.
(*nth la* (*Suc i*)) $\leq$ *intlen l*
)

using *5 7 Suc-lessI interval-idx-less-last-1* **by** *fastforce*
have *204*: ($\forall$ *i*< *intlen la*.
intlen (*map* (*shiftm* (*nth l* (*nth la i*))) (*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*)) =
(*nth la* (*Suc i*)) − (*nth la i*))

by (*simp add*: *5 PJ6help1 filt-intlen*)
have *205*: ($\forall$ *i*< *intlen la*.
($\forall$ *j* $\leq$ (*nth la* (*Suc i*)) $-$ (*nth la i*).
(*nth* (*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*) *j*) =
(*nth l* ((*nth la i*) +*j*))
)
)

using *2041 5 index-sequence-def interval-nth-sub order.strict-implies-order* **by** *blast*
have *2060*: ($\forall$ *i*< *intlen la*.
(*nth la i*) $\leq$ (*nth la* (*Suc i*)))

using *5 interval-idx-expand* **by** *fastforce*
have *206*: ($\forall$ *i*< *intlen la*.
($\forall$ *j* $\leq$ (*nth la* (*Suc i*)) $-$ (*nth la i*).
(*nth l* (*nth la i*)) $\leq$ (*nth l* ((*nth la i*) +*j*))
)
)

using *2041 3 2060 interval-idx-less-equal* **by** *fastforce*
have *207*: ($\forall$ *i*< *intlen la*.
($\forall$ *j* $\leq$ (*nth la* (*Suc i*)) $-$ (*nth la i*).
(*nth* (*map* (*shiftm* (*nth l* (*nth la i*))) (*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*)) *j*) =
(*nth l* ((*nth la i*) +*j*)) $-$ (*nth l* (*nth la i*))
))

by (*simp add*: *205 interval-nth-map shiftm-def*)
have *208*: ($\forall$ *i*< *intlen la*.
(*nth* (*map* (*shiftm* (*nth l* (*nth la i*))) (*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*)) *0*) =*0*
)
by (*simp add*: *207*)
have *209*: ($\forall$ *i*< *intlen la*.
intlast (*map* (*shiftm* (*nth l* (*nth la i*))) (*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*)) =
(*nth l* (*nth la* (*Suc i*))) $-$ (*nth l* (*nth la i*))
)

using *204 207 5 2060* **by** *simp-all*
have *210*: ($\forall$ *i*< *intlen la*.
index-sequence 0 (*map* (*shiftm* (*nth l* (*nth la i*))) (*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*))
)

by (*metis 3 5 7 add.commute interval-idx-expand interval-idx-shiftm*
interval-idx-sub plus-1-eq-Suc)
have *211*: ($\forall$ *i*< *intlen la*.
powerinterval f (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ)
(*map* (*shiftm* (*nth l* (*nth la i*))) (*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*))
)

proof $-$
have *2111*: ($\forall$ *i*< *intlen la*.

```
           powerinterval f (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
                           (map (shiftm (nth l (nth la i)))
                           (sub (nth la i) (nth la (Suc i)) l))
          ) =
          (∀ i<intlen la.
           (∀ ia<intlen (sub (nth la i) (nth la (Suc i)) l).
             f (sub (nth (map (shiftm (nth l (nth la i)))
                             (sub (nth la i) (nth la (Suc i)) l)) ia)
                   (nth (map (shiftm (nth l (Interval.nth la i)))
                             (sub (nth la i) (nth la (Suc i)) l)) (Suc ia))
                   (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ) )
           )
           )
 by (simp add: powerinterval-def )
have 2112: ... =
          (∀ i<intlen la.
           (∀ ia<  (nth la (Suc i)) − (nth la i).
             f (sub (nth (map (shiftm (nth l (nth la i)))
                             (sub (nth la i) (nth la (Suc i)) l)) ia)
                   (nth (map (shiftm (nth l (Interval.nth la i)))
                             (sub (nth la i) (nth la (Suc i)) l)) (Suc ia))
                   (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ))
           )
          )
      using 204 by auto
have 2113: ... =
          (∀ i<intlen la.
           (∀ ia<  (nth la (Suc i)) − (nth la i).
              f (sub  ((nth l ((nth la i) +ia)) − (nth l (nth la i)))
                     ((nth l ((nth la i) +(Suc ia))) − (nth l (nth la i)))
                     (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ))
           )
          )
      using 207 by auto
have 2114:
          (∀ i<intlen la.
            (let m = (nth la (Suc i)); n= (nth la i) in
             (∀ ia<  m − n.
               (nth l (n +ia))  ≤ ((nth l (n+ (Suc ia))) )
             )
            )
           )
       using 2041 3 2060 interval-idx-less-equal by simp-all fastforce
  have 2115:
          (∀ i<intlen la.
            (let m = (nth la (Suc i)); n= (nth la i) in
             (∀ ia<  m − n.
               (nth l (n +ia)) − (nth l n) ≤ ((nth l ((Suc (n+ia)))) − (nth l n))
             )
            )
```

)
by (*metis 2114 add-Suc-right diff-le-mono*)
have *2116*:
(∀ *i*<*intlen la*.
(*let m* = (*nth la* (*Suc i*)); *n*= (*nth la i*) *in*
(∀ *ia*< *m* − *n*.
((*nth l* ((*Suc* (*n*+*ia*)))) − (*nth l n*)) ≤ (*nth l m*) − (*nth l n*)
)
)
)
by (*metis 3 5 7 Interval.interval-idx-expand Suc-leI add.commute diff-le-mono interval-idx-less-equal less-diff-conv plus-1-eq-Suc*)
have *2117*: (∀ *i*<*intlen la*.
(*let m* = (*nth la* (*Suc i*)); *n*= (*nth la i*) *in*
(*nth l m*) ≤ *intlen σ*))
by (*metis 12 9 Suc-lessI filt-nth filt-intlen interval-idx-less-last-1 less-or-eq-imp-le*)
have *2118*: (∀ *i*<*intlen la*.
(*let m* = (*nth la* (*Suc i*)); *n*= (*nth la i*) *in*
(∀ *ia*< *m* − *n*.
(*n*+(*Suc ia*)) ≤ *intlen l*)
)
)
using *5 7 Interval.interval-idx-expand less-diff-conv* **by** *fastforce*
have *21190*: (∀ *i*<*intlen la*.
(*let m* = (*nth la* (*Suc i*)); *n*= (*nth la i*) *in*
(∀ *ia*< *m* − *n*.
((*nth l* (*n* +*ia*)) − (*nth l n*)) ≤ ((*nth l* (*n*+(*Suc ia*))) − (*nth l n*))
)))

by (*meson 2114 diff-le-mono*)
have *21191*: (∀ *i*<*intlen la*.
(*let m* = (*nth la* (*Suc i*)); *n*= (*nth la i*) *in*
(∀ *ia*< *m* − *n*.
((*nth l* (*n*+(*Suc ia*))) − (*nth l n*)) ≤
intlen (*sub* (*nth l n*) (*nth l m*) *σ*)
)))

by (*simp add*: *203 2116*)
have *2119*: (∀ *i*<*intlen la*.
(*let m* = (*nth la* (*Suc i*)); *n*= (*nth la i*) *in*
(∀ *ia*< *m* − *n*.
intlen (*sub* ((*nth l* (*n* +*ia*)) − (*nth l n*))
((*nth l* (*n*+(*Suc ia*))) − (*nth l n*))
(*sub* (*nth l n*) (*nth l m*) *σ*)) =
((*nth l* (*n*+(*Suc ia*)))) − ((*nth l* (*n* +*ia*)))
)
)
)

```
     by (metis 203 206 2115 2116  Nat.diff-diff-eq Suc-leI add-Suc-right
        interval-intlen-sub less-imp-le-nat)
have 2120: (∀ i<intlen la.
          (let m = (nth la (Suc i)); n= (nth la i) in
           (∀ ia<  m − n.
            (∀ j ≤ ((nth l (n+(Suc ia)))) − ((nth l (n +ia))).
               nth  (sub  ((nth l (n +ia)) − (nth l n))
                          ((nth l (n+(Suc ia))) − (nth l n))
                          (sub (nth l n) (nth l m) σ)) j =
               nth (sub (nth l n) (nth l m) σ) (((nth l (n +ia)) − (nth l n))+j)
      )   )))
      by (metis 2119 21190 21191 interval-intlen-sub interval-nth-sub)
have 212111: (∀ i<intlen la.
           (let m = (nth la (Suc i)); n= (nth la i) in
            (∀ ia<  m − n.
                 ((nth l (n +ia))) ≤ ((nth l (n+(Suc ia))))
            )
           )
          )

      using 2114 by blast
have 212112: (∀ i<intlen la.
           (let m = (nth la (Suc i)); n= (nth la i) in
            (∀ ia<  m − n.
                 ((nth l (n))) ≤ ((nth l (n+(ia))))
            )
           )
          )
   by (simp add: 206)
have 212113: (∀ i<intlen la.
           (let m = (nth la (Suc i)); n= (nth la i) in
            (∀ ia<  m − n.
             (∀  j ≤ ((nth l (n+(Suc ia)))) − ((nth l (n +ia))).
               ((nth l (n +ia)) )+j ≤ (nth l m)
             )
            )
           )
          )
    using 212111  212112  206 2060 2116
    unfolding Let-def by fastforce
have 21211: (∀ i<intlen la.
           (let m = (nth la (Suc i)); n= (nth la i) in
            (∀ ia<  m − n.
             (∀  j ≤ ((nth l (n+(Suc ia)))) − ((nth l (n +ia))).
               ((nth l (n +ia)) − (nth l n))+j ≤ (nth l m) − (nth l n)
             )
            )
           )
          )
```

```
 by (metis 212112 212113 Nat.add-diff-assoc2 diff-le-mono)
have 2121: (∀ i<intlen la.
           (let m = (nth la (Suc i)); n= (nth la i) in
            (∀ ia<  m − n.
             (∀  j ≤ ((nth l (n+(Suc ia)))) − ((nth l (n +ia))).
               (nth (sub (nth l n) (nth l m) σ) (((nth l (n +ia)) − (nth l n))+j) ) =
               (nth σ ((nth l n) + (((nth l (n +ia)) − (nth l n))+j))   )
             )
            )
           )
          )

      by (metis 206 2117 21211 add-diff-inverse-nat interval-nth-sub nat-diff-split
          not-less-zero order-refl)
have 2122: (∀ i<intlen la.
           (let m = (nth la (Suc i)); n= (nth la i) in
            (∀ ia<  m − n.
             (∀  j ≤ ((nth l (n+(Suc ia)))) − ((nth l (n +ia))).
              (nth σ ((nth l n) + (((nth l (n +ia)) − (nth l n))+j))   ) =
              (nth σ (  (((nth l (n +ia)))+j))   )
             )
            )
           )
          )
     by (metis 206 add.assoc le-add-diff-inverse less-imp-le-nat)
have 2123:  (∀ i<intlen la.
            (let m = (nth la (Suc i)); n= (nth la i) in
             (∀ ia<  m − n.
              (∀  j ≤ ((nth l (n+(Suc ia)))) − ((nth l (n +ia))).
                nth (sub (nth l (n +ia)) ((nth l (n+(Suc ia)))) σ) j =
                (nth σ (  (((nth l (n +ia)))+j)) )
              )
             )
            )
           )

          by (metis 2114 2118 3 eq-imp-le interval-idx-less-equal interval-nth-sub)
have 2124: (∀ i<intlen la.
           (let m = (nth la (Suc i)); n= (nth la i) in
            (∀ ia<  m − n.
              intlen (sub (nth l (n +ia)) ((nth l (n+(Suc ia)))) σ) =
              ((nth l (n+(Suc ia)))) − ((nth l (n +ia)))
            )
           )
          )

      by (metis 2118 3 Suc-eq-plus1 Suc-le-lessD add-Suc-right interval-idx-expand
         interval-intlen-sub)
have 2125: (∀ i<intlen la.
          (let m = (nth la (Suc i)); n= (nth la i) in
```

```
          (∀ ia<  m − n.
            (∀ j ≤ ((nth l (n+(Suc ia)))) − ((nth l (n +ia))).
               nth  (sub  ((nth l (n +ia)) − (nth l n))
                          ((nth l (n+(Suc ia))) − (nth l n))
                          (sub (nth l n) (nth l m) σ)) j =
             (nth (sub (nth l (n +ia)) ((nth l (n+(Suc ia)))) σ) j)
            )
           )
          )
         )

 by (metis 2120 2121 2122 2123)
have 2126: (∀ i<intlen la.
         (let m = (nth la (Suc i)); n= (nth la i) in
           (∀ ia<  m − n.
             intlen (sub  ((nth l (n +ia)) − (nth l n))
                          ((nth l (n+(Suc ia))) − (nth l n))
                          (sub (nth l n) (nth l m) σ)) =
             intlen (sub (nth l (n +ia)) ((nth l (n+(Suc ia)))) σ)
           )
          )
         )
   by (metis 2119 2124)
have 2127: (∀ i<intlen la.
         (let m = (nth la (Suc i)); n= (nth la i) in
           (∀ ia<  m − n.
            (sub  ((nth l (n +ia)) − (nth l n))
                  ((nth l (n+(Suc ia))) − (nth l n))
                  (sub (nth l n) (nth l m) σ)) =
            (sub (nth l (n +ia)) ((nth l (n+(Suc ia)))) σ)
           )
          )
         )

    using interval-eq-nth-eq 2125 2126
    by (metis 2119)
have 2128: (∀ i<intlen la.
         (let m = (nth la (Suc i)); n= (nth la i) in
           (∀ ia<  m − n.
            f (sub (nth l (n +ia)) ((nth l (n+(Suc ia)))) σ)
           )
          )
         )

      by (metis 2041 3 add.commute add-Suc-right less-diff-conv less-le-trans
           powerinterval-def)
 show ?thesis
 by (metis 2111 2112 2113 2127 2128)
qed
have 220: (∀  i< intlen la.
```

```
      ((filt (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
            (map (shiftm (nth l (nth la i))) (sub (nth la i) (nth la (Suc i)) l)) )
        ⊨ g)
      )

proof −
 have 2201: (∀ i< intlen la.
             intlen (filt (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
                         (map (shiftm (nth l (nth la i)))
                              (sub (nth la i) (nth la (Suc i)) l)) ) =
           intlen (map (shiftm (nth l (nth la i))) (sub (nth la i) (nth la (Suc i)) l))
         )

     using filt-intlen by blast
 have 2202: (∀ i< intlen la.
              intlen (map (shiftm (nth l (nth la i)))
                         (sub (nth la i) (nth la (Suc i)) l)) =
                 (nth la (Suc i)) − (nth la i)
           )

   using 204 by blast
 have 2203: (∀ i< intlen la.
            (∀ j≤ (nth la (Suc i)) − (nth la i) .
              nth ((filt (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
                        (map (shiftm (nth l (nth la i)))
                             (sub (nth la i) (nth la (Suc i)) l)) ) ) j =
                nth (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
                    (nth (map (shiftm (nth l (nth la i)))
                             (sub (nth la i) (nth la (Suc i)) l)) j)
             ))

      by (simp add: filt-map interval-nth-map)
 have 2204: (∀ i< intlen la.
            (∀ j≤ (nth la (Suc i)) − (nth la i) .
               nth (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
                   (nth (map (shiftm (nth l (nth la i)))
                             (sub (nth la i) (nth la (Suc i)) l)) j) =
               nth (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
                  ((nth (sub (nth la i) (nth la (Suc i)) l) j) −(nth l (nth la i)))
            ))

    by (simp add: interval-nth-map shiftm-def)
 have 2205: (∀ i< intlen la.
            (
               (nth l (nth la i)) ≤ (nth l (nth la (Suc i))) ∧
               (nth l (nth la (Suc i))) ≤ intlen σ
            ))

         by (metis 12 9 add.commute filt-intlen filt-map interval-idx-expand
               interval-nth-map  plus-1-eq-Suc)
```

```
have 2206 : (∀ i< intlen la.
             (∀ j≤ (nth la (Suc i)) − (nth la i) .
               (nth (sub (nth la i) (nth la (Suc i)) l) j) =
                (nth l ((nth la i)+j))
             ))

  using 205 by blast
have 2207: (∀ i< intlen la.
             (∀ j≤ (nth la (Suc i)) − (nth la i) .
              ((nth l (nth la i)) + ((nth (sub (nth la i) (nth la (Suc i)) l) j)
                                    −(nth l (nth la i)))) =
                 (nth l ((nth la i)+j))
             ))

    by (simp add: 206 2206)
have 2208: (∀ i< intlen la.
             (∀ j≤ (nth la (Suc i)) − (nth la i) .
               nth (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
                   ((nth (sub (nth la i) (nth la (Suc i)) l) j) −(nth l (nth la i))) =
               nth σ  (nth l ((nth la i)+j))
             ))

     by (metis (no-types, lifting) 2041 206 2060 2205 3 Nat.le-diff-conv2
          add.commute interval-idx-less-equal interval-nth-sub le-add-diff-inverse2)
have 2209: (∀ i< intlen la.
             (∀ j≤ (nth la (Suc i)) − (nth la i) .
                (nth (filt σ l) ((nth la i)+j)) = (nth σ (nth l ((nth la i)+j)))
             ))

  by (simp add: filt-map interval-nth-map)
have 2210: (∀ i< intlen la.
              intlen  (sub (nth la i) (nth la (Suc i)) (filt σ l)) =
                (nth la (Suc i)) − (nth la i)
           )

          using 5 PJ6help1 by blast
have 2211: (∀ i< intlen la.
            (∀ j≤ (nth la (Suc i)) − (nth la i) .
             (nth (sub (nth la i) (nth la (Suc i)) (filt σ l)) j) =
              (nth σ (nth l ((nth la i)+j)))
            ))

          using 2209 5 interval-idx-expand interval-nth-sub by fastforce
have 2212: (∀ i< intlen la.
            (∀ j≤ (nth la (Suc i)) − (nth la i) .
                   nth ((filt (sub (nth l (nth la i)) (nth l (nth la (Suc i))) σ)
                          (map (shiftm (nth l (nth la i)))
                               (sub (nth la i) (nth la (Suc i)) l)) ) ) j =
           (nth (sub (nth la i) (nth la (Suc i)) (filt σ l)) j)
       ))
```

by (*simp add*: *2203 2204 2208 2211*)
have *2213*: (∀ *i*< *intlen la*.
intlen ((*filt* (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ)
(*map* (*shiftm* (*nth l* (*nth la i*)))
(*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*)))) =
intlen (*sub* (*nth la i*) (*nth la* (*Suc i*)) (*filt* σ *l*))
)

by (*metis 2202 2210 filt-intlen*)
have *2214*: (∀ *i*< *intlen la*.
(*filt* (*sub* (*nth l* (*nth la i*)) (*nth l* (*nth la* (*Suc i*))) σ)
(*map* (*shiftm* (*nth l* (*nth la i*))) (*sub* (*nth la i*) (*nth la* (*Suc i*)) *l*))) =
(*sub* (*nth la i*) (*nth la* (*Suc i*)) (*filt* σ *l*))
)

using *interval-eq-nth-eq 2212 2213* **using** *2201 2202* **by** *fastforce*
have *2215*: (∀ *i*< *intlen la*.
(*sub* (*nth la i*) (*nth la* (*Suc i*)) (*filt* σ *l*)) ⊨ *g*
)
using *5 powerinterval-def* **by** *blast*
show *?thesis* **by** (*simp add*: *2214 2215*)
qed
show *?thesis*
using *201 202 203 209 210 211 220* **by** *metis*
qed
show *?thesis*
by (*metis 12 20 5 8 9 projection-d-def*)
qed

lemma *PJ7sem*:
(σ ⊨ *f* △ (*g* △ *h*) = (*f* △ *g*) △ *h*)
proof −
have *1*: *intlen* σ >*0* ⟶ (σ ⊨ *f* △ (*g* △ *h*) = (*f* △ *g*) △ *h*)
using *PJ7helpchain1a PJ7helpchain1b unl-lift2* **by** *blast*
have *2*: *intlen* σ = *0* ⟶ (σ ⊨ *f* △ (*g* △ *h*) = (*f* △ *g*) △ *h*)
using *PJ7empty* **by** *blast*
from *1 2* **show** *?thesis* **by** *auto*
qed

10.3.8 PJ8

lemma *PJ8semhelp*:
assumes *index-sequence 0 l*
Interval.nth l (*intlen l*) = *intlen* σ
(∀ *n na*. *na* + *n* ≤ *intlen* σ ⟶ *f* (*sub n* (*n*+ *na*) σ) ⟶ *g* (*sub n* (*n* + *na*) σ))
shows
(∀ *i*<*intlen l*. *f* (*sub* (*Interval.nth l i*) (*Interval.nth l* (*Suc i*)) σ)

$\longrightarrow g\ (sub\ (Interval.nth\ l\ i)\ (Interval.nth\ l\ (Suc\ i))\ \sigma)$
)
by (*metis add.commute assms interval-idx-expand ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)

lemma *PJ8sem*:
$(\sigma \models ba(f \longrightarrow g) \longrightarrow (f \bigtriangleup h) \longrightarrow (g \bigtriangleup h))$
using *PJ8semhelp* **by** (*simp add*: *projection-d-def ba-defs powerinterval-def*) *blast*

10.3.9 PJ9

lemma *PJ9sem*:
$(\sigma \models f\ \nabla\ (\ g \longrightarrow h) \longrightarrow f \bigtriangleup g \longrightarrow f \bigtriangleup h)$
by (*simp add*: *uprojection-d-def projection-d-def*, *metis*)

10.4 Axioms

lemma *BpGen*:
assumes $\vdash f$
shows $\vdash bp\ f$
using *assms*
by (*simp add*: *bp-d-def uprojection-d-def projection-d-def Valid-def*)

lemma *PJ1*:
$\vdash f \bigtriangleup (\ g \vee h) \longrightarrow f \bigtriangleup g \vee f \bigtriangleup h$
using *PJ1sem Valid-def* **by** *blast*

lemma *PJ2*:
$\vdash f \bigtriangleup empty = empty$
using *PJ2sem Valid-def* **by** *blast*

lemma *PJ3*:
$\vdash f \bigtriangleup skip = (f \wedge more)$
using *PJ3sem Valid-def* **by** *blast*

lemma *PJ4*:
$\vdash f \bigtriangleup (g;h) = (f \bigtriangleup g)\ ;\ (\ f \bigtriangleup h)$
using *PJ4sem Valid-def* **by** *blast*

lemma *PJ5*:
$\vdash f \bigtriangleup init(g) \longrightarrow init(g)$
using *PJ5sem Valid-def* **by** *blast*

lemma *PJ6*:
$\vdash skip \bigtriangleup g = g$
using *PJ6sem Valid-def* **by** *blast*

lemma *PJ7*:
$\vdash f \bigtriangleup (g \bigtriangleup h) = (f \bigtriangleup g) \bigtriangleup h$

using *PJ7sem Valid-def* **by** *blast*

lemma *PJ8*:
$\vdash ba(f \longrightarrow g) \longrightarrow (f \triangle h) \longrightarrow (g \triangle h)$
using *PJ8sem Valid-def* **by** *blast*

lemma *PJ9*:
$\vdash f \nabla (g \longrightarrow h) \longrightarrow f \triangle g \longrightarrow f \triangle h$
using *PJ9sem Valid-def* **by** *blast*

10.5 Time Reversal

lemma *filt-intapp*:
$filt\ w\ (l \ominus \langle x \rangle) = (filt\ w\ l) \ominus \langle (nth\ w\ x) \rangle$
proof
(*induct l*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a l*)
then show *?case*
by *simp*
qed

lemma *filt-rev*:
assumes $\forall\ i \leq intlen\ l.\ (nth\ l\ i) \leq intlen\ w$
shows $(filt\ (intrev\ w)\ l) = (intrev\ (filt\ w\ (map\ (\lambda x.\ intlen\ w - x)\ (intrev\ l))\ ))$
using *assms*
proof
(*induct l*)
case (*St x*)
then show *?case*
proof −
have *01*: $filt\ (intrev\ w)\ \langle x \rangle = \langle\ nth\ (intrev\ w)\ x\ \rangle$
by *simp*
have *02*: $\langle\ nth\ (intrev\ w)\ x\ \rangle = \langle\ nth\ w\ (intlen\ w - x)\ \rangle$
using *St.prems interval-intrev-nth* **by** *auto*
have *03*: $intrev\ (filt\ w\ (interval.map\ ((-)\ (intlen\ w))\ \langle x \rangle)) = \langle\ nth\ w\ (intlen\ w - x)\ \rangle$
by *simp*
from *01 02 03* **show** *?thesis* **by** *auto*
qed
next
case (*Cons x1a l*)
then show *?case*
proof −
have *1*: $filt\ (intrev\ w)\ (x1a \odot l) =$
$(nth\ (intrev\ w)\ x1a) \odot (filt\ (intrev\ w)\ l)$
by *simp*

```
    have 2: (nth (intrev w) x1a) = (nth w (intlen w − x1a) )
       using Cons.prems interval-intrev-nth by fastforce
    have 3: intrev (filt w (x1a ⊙ l)) = intrev ((nth w x1a) ⊙ (filt w l))
     by simp
    have 4: intrev (filt w (map ((−) (intlen w)) (intrev (x1a ⊙ l)))) =
            intrev (filt w (map ((−) (intlen w)) ( (intrev l) ⊖ ⟨ x1a ⟩ )))

     by simp
    have 5: intrev (filt w (map ((−) (intlen w)) ( (intrev l) ⊖ ⟨ x1a ⟩ ))) =
            intrev ( (filt w (map ((−) (intlen w)) ( (intrev l)))) ⊖ ⟨ nth w (intlen w −x1a) ⟩)
      by (simp add: filt-intapp)
    have 6: intrev ( (filt w (map ((−) (intlen w)) ( (intrev l)))) ⊖ ⟨ nth w (intlen w −x1a) ⟩) =
            (nth w (intlen w −x1a)) ⊙ intrev( (filt w (map ((−) (intlen w)) ( (intrev l)))))
      by auto
    have 7: ∀ i ≤ intlen l. (nth l i) ≤ intlen w
       using local.Cons(2) by auto
    have 8 : (filt (intrev w) l) = intrev(filt w (interval.map ((−) (intlen w)) (intrev l)))
      using 7 Cons.hyps by blast
    show ?thesis
    using 2 5 8 by auto
    qed
qed

lemma ProjectionRevsema:
 assumes (σ |= (f △ g)^r)
 shows    (σ |= (f^r) △ (g^r))
proof −
 have 1: (σ |= (f △ g)^r)
  using assms by auto
 have 2: ∃ l. index-sequence 0 l ∧ Interval.nth l (intlen l) = intlen σ ∧
         powerinterval f (intrev σ) l ∧ g (filt (intrev σ) l)
  using 1 by (simp add: projection-d-def reverse-d-def)
 obtain l where 3: index-sequence 0 l ∧ Interval.nth l (intlen l) = intlen σ ∧
         powerinterval f (intrev σ) l ∧ g (filt (intrev σ) l)
  using 2 by auto
 have 4: index-sequence 0 l
  using 3 by auto
 have 5: Interval.nth l (intlen l) = intlen σ
  using 3 by auto
 have 6: powerinterval f (intrev σ) l
   using 3 by auto
 have 7: g (filt (intrev σ) l)
    using 3 by auto
 have 8: ∀ i ≤ intlen l. (nth l i) ≤ intlen σ
  using 4 5 interval-idx-less-last-1 le-eq-less-or-eq by fastforce
 have 9: g ( intrev(filt σ (map (λx. intlen σ −x) (intrev l)) ))
  using 7 8 filt-rev by fastforce
 have 10: nth (map ((−) (intlen σ)) (intrev l)) 0 = 0
     by (metis 5 diff-self-eq-0 interval-intfirst-intrev interval-nth-intlen-intlast
         interval-nth-map interval-nth-zero-intfirst)
```

have *11*: ($\forall$ *n*<*intlen l*.
nth (*map* (($-$) (*intlen* σ)) (*intrev l*)) *n*
< *nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*Suc n*))
by (*simp add*: *interval-nth-map interval-intrev-nth*)
(*metis* (*no-types*, *lifting*) *4 5 Suc-diff-Suc Suc-less-eq diff-less-Suc diff-less-mono2 index-sequence-def interval-idx-less-last-1*)
have *12*: *index-sequence 0* (*map* (λ*x*. *intlen* σ $-$*x*) (*intrev l*))
by (*simp add*: *10 11 index-sequence-def*)
have *13*: *intlen* (*map* (λ*x*. *intlen* σ $-$*x*) (*intrev l*)) = *intlen l*
by *auto*
have *14*: *nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*intlen* (*map* (($-$) (*intlen* σ)) (*intrev l*))) =
intlen σ
by (*metis 4 diff-zero index-sequence-def interval-intlast-intrev interval-intlen-map interval-nth-intlen-intlast interval-nth-map interval-nth-zero-intfirst*)
have *15*: $\forall$ *i*<*intlen l*.
(*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) *i*) $\leq$
(*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*Suc i*))

by (*simp add*: *11 less-imp-le-nat*)
have *16*: $\forall$ *i*<*intlen l*.
(*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*Suc i*)) $\leq$ *intlen* σ

by (*simp add*: *interval-nth-map*)
have *17*: $\forall$ *i*<*intlen l*.
intrev
(*sub* (*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) *i*)
(*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*Suc i*)) σ) =
sub (*intlen* σ $-$ (*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*Suc i*)))
(*intlen* σ $-$ (*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) *i*))
(*intrev* σ)

using *interval-intrev-sub*
using *15 16* **by** *blast*
have *18*: $\forall$ *i*<*intlen l*.
intlen σ $-$ (*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*Suc i*)) =
(*nth* (*intrev l*) (*Suc i*))

by (*metis 10 12 13 14 Suc-leI diff-diff-cancel interval-idx-less-than interval-intlen-gr-zero interval-nth-map less-le zero-less-Suc zero-less-diff*)
have *19*: $\forall$ *i*<*intlen l*.
intlen σ $-$ (*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*i*)) =
(*nth* (*intrev l*) (*i*))
by (*metis 12 13 5 diff-diff-cancel gr-zeroI interval-idx-greater-first interval-intfirst-intrev interval-intlast-prefix interval-nth-map interval-nth-zero-intfirst interval-prefix-intlen order.order-iff-strict zero-less-diff*)
have *20*: $\forall$ *i*<*intlen l*.
(*sub* (*intlen* σ $-$ (*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) (*Suc i*)))
(*intlen* σ $-$ (*nth* (*map* (($-$) (*intlen* σ)) (*intrev l*)) *i*))
(*intrev* σ)) =
(*sub* (*nth* (*intrev l*) (*Suc i*))

$(nth\ (intrev\ l)\ (i))$
$(intrev\ \sigma))$

using *18 19* **by** *simp*
have *21*: $\forall\, i{<}intlen\ l.$
$(\ sub\ (nth\ (intrev\ l)\ (Suc\ i))$
$(nth\ (intrev\ l)\ (i))$
$(intrev\ \sigma)) =$
$(\ sub\ (nth\ l\ (intlen\ l -\ (Suc\ i)))$
$(nth\ l\ (intlen\ l - i))$
$(intrev\ \sigma))$

by (*simp add*: *interval-intrev-nth*)
have *22*: $\forall\, i{<}intlen\ l.$
$f\ (\ sub\ (nth\ l\ (intlen\ l -\ (Suc\ i)))$
$(nth\ l\ (intlen\ l - i))$
$(intrev\ \sigma))$

by (*metis 6 Suc-diff-Suc Suc-less-eq diff-less-Suc powerinterval-def*)
have *24*: $\forall\, i{<}intlen\ l.$
$f\ (intrev$
$(sub\ (nth\ (map\ ((-)\ (intlen\ \sigma))\ (intrev\ l))\ i)$
$(nth\ (map\ ((-)\ (intlen\ \sigma))\ (intrev\ l))\ (Suc\ i))\ \sigma))$
by (*simp add*: *17 20 21 22*)
have *25*: $powerinterval\ (\lambda s.\ f\ (intrev\ s))\ \sigma\ (map\ ((-)\ (intlen\ \sigma))\ (intrev\ l))$
by (*simp add*: *24 powerinterval-def*)
have *26*: $(\exists\, l.\ index\text{-}sequence\ 0\ l\ \wedge$
$nth\ l\ (intlen\ l) = intlen\ \sigma\ \wedge$
$powerinterval\ (\lambda s.\ f\ (intrev\ s))\ \sigma\ l \wedge g\ (intrev(filt\ \sigma\ l)))$
using *12 14 25 9* **by** *blast*
from *26* **show** *?thesis*
by (*simp add*: *projection-d-def reverse-d-def*)
qed

lemma *ProjectionRevsemb*:
assumes $(\sigma \models (f^r) \bigtriangleup (g^r))$
shows $(\sigma \models (f \bigtriangleup g)^r)$
proof −
have *1*: $(\exists\, l.\ index\text{-}sequence\ 0\ l\ \wedge$
$nth\ l\ (intlen\ l) = intlen\ \sigma\ \wedge$
$powerinterval\ (\lambda s.\ f\ (intrev\ s))\ \sigma\ l \wedge g\ (intrev(filt\ \sigma\ l)))$
using *assms* **by** (*simp add*: *projection-d-def reverse-d-def*)
obtain *l* **where** *2*: $index\text{-}sequence\ 0\ l\ \wedge$
$nth\ l\ (intlen\ l) = intlen\ \sigma\ \wedge$
$powerinterval\ (\lambda s.\ f\ (intrev\ s))\ \sigma\ l \wedge g\ (intrev(filt\ \sigma\ l))$
using *1* **by** *auto*
have *3*: $index\text{-}sequence\ 0\ l$
using *2* **by** *auto*
have *4*: $nth\ l\ (intlen\ l) = intlen\ \sigma$

using *2* **by** *auto*
have *5*: *powerinterval* (*λs*. *f* (*intrev s*)) *σ l*
using *2* **by** *auto*
have *6*: *g* (*intrev*(*filt σ l*))
using *2* **by** *auto*
have *7*: *intlen* (*map* ((−) (*intlen σ*)) (*intrev l*)) = *intlen l*
by *simp*
have *8*: ∀ *i* ≤ *intlen l*. (*nth* (*map* ((−) (*intlen σ*)) (*intrev l*)) *i*) ≤ *intlen σ*
by (*simp add*: *interval-nth-map*)
have *9*: *g* (*filt* (*intrev σ*) (*map* ((−) (*intlen σ*)) (*intrev l*)))
by (*metis 2 filt-rev interval-idx-less-equal interval-intrev-intlen interval-rev-rev-ident le-refl*)
have *10*: *nth* (*map* ((−) (*intlen σ*)) (*intrev l*)) (*intlen* (*map* ((−) (*intlen σ*)) (*intrev l*))) = *intlen σ*
by (*metis 2 diff-zero index-sequence-def interval-intlast-intrev interval-intlen-map interval-nth-intlen-intlast interval-nth-map interval-nth-zero-intfirst*)
have *11*: *nth* (*map* ((−) (*intlen σ*)) (*intrev l*)) *0* = *0*
by (*metis 4 diff-self-eq-0 interval-intfirst-intrev interval-nth-intlen-intlast interval-nth-map interval-nth-zero-intfirst*)
have *12*: (∀ *n*<*intlen l*.
nth (*map* ((−) (*intlen σ*)) (*intrev l*)) *n*
< *nth* (*map* ((−) (*intlen σ*)) (*intrev l*)) (*Suc n*))
by (*simp add*: *interval-nth-map interval-intrev-nth*)
(*metis* (*no-types*, *lifting*) *2 Suc-diff-Suc Suc-less-eq diff-less-Suc diff-less-mono2 index-sequence-def interval-idx-less-last-1*)
have *13*: *index-sequence 0* (*map* (*λx*. *intlen σ* −*x*) (*intrev l*))
by (*simp add*: *11 12 index-sequence-def*)
have *14*: ∀ *i*<*intlen l*. *f* (*intrev* (*sub* (*nth l i*) (*nth l* (*Suc i*)) *σ*))
using *5* **by** (*simp add*:*powerinterval-def*)
have *15*: ∀ *i*<*intlen l*. (*nth l* (*Suc i*)) ≤ *intlen σ*
using *2 interval-idx-expand* **by** *fastforce*
have *16*: ∀ *i*<*intlen l*. (*nth l i*) ≤ (*nth l* (*Suc i*))
using *2 interval-idx-expand* **by** *fastforce*
have *17*: ∀ *i*<*intlen l*.
f (*sub* ((*intlen σ*)−(*nth l* (*Suc i*))) ((*intlen σ*) − (*nth l i*)) (*intrev σ*))

using *14*
by (*simp add*: *interval-intrev-sub 15 16*)
have *18*: ∀ *i*<*intlen l*.
(*nth* (*map* ((−) (*intlen σ*)) (*intrev l*)) *i*) =
intlen σ − (*nth* (*intrev l*) *i*)

using *interval-nth-map* **by** *blast*
have *19*: ∀ *i*<*intlen l*.
(*nth* (*map* ((−) (*intlen σ*)) (*intrev l*)) (*Suc i*)) =
intlen σ − (*nth* (*intrev l*) (*Suc i*))

using *interval-nth-map* **by** *blast*
have *20*: ∀ *i*<*intlen l*.
(*nth* (*intrev l*) *i*) = (*nth l* (*intlen l* − *i*))

by (*simp add*: *interval-intrev-nth*)
have *21*: $\forall\, i{<}intlen\ l.$
$(nth\ (intrev\ l)\ (Suc\ i)) = (nth\ l\ (intlen\ l - (Suc\ i)))$
by (*simp add*: *interval-intrev-nth*)
have *22*: $\forall\, i{<}intlen\ l.$
$f\ (sub\ (intlen\ \sigma - (nth\ l\ (intlen\ l - i)))$
$(intlen\ \sigma - (nth\ l\ (intlen\ l - (Suc\ i)))\ )$
$(intrev\ \sigma))$

by (*metis 17 Suc-diff-Suc Suc-less-eq diff-less-Suc*)
have *23*: $\forall\, i{<}intlen\ l.$
$f\ (sub\ (intlen\ \sigma - (nth\ (intrev\ l)\ i))$
$(intlen\ \sigma - (nth\ (intrev\ l)\ (Suc\ i))\ )$
$(intrev\ \sigma))$

by (*simp add*: *20 21 22*)
have *24*: $\forall\, i{<}intlen\ l.$
$f\ (sub\ (nth\ (map\ ((-)\ (intlen\ \sigma))\ (intrev\ l))\ i)$
$(nth\ (map\ ((-)\ (intlen\ \sigma))\ (intrev\ l))\ (Suc\ i))\ (intrev\ \sigma))$
by (*simp add*: *23 interval-nth-map*)
have *25*: *powerinterval f* $(intrev\ \sigma)\ (map\ ((-)\ (intlen\ \sigma))\ (intrev\ l))$
by (*simp add*: *24 powerinterval-def*)
have *26*: $\exists\, l.$ *index-sequence 0 l* $\wedge$ *Interval.nth l* $(intlen\ l) = intlen\ \sigma\ \wedge$
powerinterval f $(intrev\ \sigma)\ l \wedge g\ (filt\ (intrev\ \sigma)\ l)$
using *10 13 25 9* **by** *blast*
from *26* **show** *?thesis* **by** (*simp add*: *projection-d-def reverse-d-def*)
qed

lemma *ProjectionRev*:
$\vdash (f \bigtriangleup g)^r = f^r \bigtriangleup g^r$
using *ProjectionRevsema ProjectionRevsemb unl-lift2* **by** *blast*

10.6 Theorems

10.6.1 Projection

lemma *PowerProjLen*:
$\vdash f \bigtriangleup len\ n = power\ (f \wedge more)\ n$
proof
(*induct n*)
case *0*
then show *?case* **by** (*metis PJ2 len-d-def pow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis PJ3 PJ4 inteq-reflection len-d-def pow-Suc*)
qed

lemma *ProjLenExist*:

$\vdash f \triangle (\exists n.\ len\ n) = (\exists\ n.\ f \triangle len\ n)$
 by (*simp add*: *Valid-def projection-d-def*, *blast*)

lemma *PowerProjLenExist*:
$\vdash (\exists\ n.\ f \triangle len\ n) = (\exists\ n.\ power\ (f \wedge more)\ n)$
using *PowerProjLen* **by** (*simp add*: *Valid-def PowerProjLen*, *blast*)

lemma *RightProjImpProj*:
 assumes $\vdash g1 \longrightarrow g2$
 shows $\vdash f \triangle g1 \longrightarrow f \triangle g2$
using *assms*
by (*simp add*: *Valid-def projection-d-def*, *blast*)

lemma *LeftProjImpProj*:
 assumes $\vdash f1 \longrightarrow f2$
 shows $\vdash f1 \triangle g \longrightarrow f2 \triangle g$
using *assms*
by (*simp add*: *Valid-def projection-d-def powerinterval-def*, *blast*)

lemma *RightProjEqvProj*:
 assumes $\vdash g1 = g2$
 shows $\vdash f \triangle g1 = f \triangle g2$
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *LeftProjEqvProj*:
 assumes $\vdash f1 = f2$
 shows $\vdash f1 \triangle g = f2 \triangle g$
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *ProjTrueEqvChopstar*:
$\vdash f \triangle \#True = f^\star$
proof –
 have *1*: $\vdash\ \#True = (\ (\exists n.\ len\ n))$
  by (*simp add*: *Valid-def len-defs*)
 have *2*: $\vdash f \triangle \#True = f \triangle (\exists n.\ len\ n)$
  using *1 RightProjEqvProj* **by** *blast*
 have *3*: $\vdash f \triangle (\exists n.\ len\ n) = (\exists\ n.\ power\ (f \wedge more)\ n)$
  using *PowerProjLenExist ProjLenExist* **by** *fastforce*
 have *4*: $\vdash (\exists\ n.\ power\ (f \wedge more)\ n) = f^\star$
  by (*simp add*: *chopstar-d-def powerstar-d-def*)
 show *?thesis* **using** *2 3 4* **by** *fastforce*
qed

lemma *ProjChopstarEqvChopstarProj*:
$\vdash f \triangle (g^\star) = (f \triangle g)^\star$
proof –

have *1*: $\vdash f \triangle (g^\star) = f \triangle (g \triangle \#True)$
by (*metis ProjTrueEqvChopstar RightProjEqvProj inteq-reflection*)
have *2*: $\vdash f \triangle (g \triangle \#True) = (f \triangle g) \triangle \#True$
by (*simp add*: *PJ7*)
have *3*: $\vdash (f \triangle g) \triangle \#True = (f \triangle g)^\star$
by (*simp add*: *ProjTrueEqvChopstar*)
show *?thesis* **using** *1 2 3* **by** *fastforce*
qed

lemma *ProjAndImp*:
$\vdash f \triangle (g1 \wedge g2) \longrightarrow f \triangle g1 \wedge f \triangle g2$
by (*meson Prop12 RightProjImpProj int-iffD1 lift-and-com*)

lemma *ProjOrDist*:
$\vdash \#True \triangle (f \vee g) = (\#True \triangle f \vee \#True \triangle g)$
using *PJ1sema* **by** *blast*

lemma *StateImportProj*:
$\vdash ((init\ w) \wedge f \triangle g) = f \triangle ((init\ w) \wedge g)$
by (*auto simp add*: *Valid-def init-defs projection-d-def filt-nth index-sequence-def*)

lemma *ProjStateAndNextEqvStateAndMoreChopProj*:
$\vdash f \triangle ((init\ w) \wedge \bigcirc g) = ((init\ w) \wedge (f \wedge more);(f \triangle g))$
proof −
have *2*: $\vdash (f \wedge more);(f \triangle g) = f \triangle \bigcirc g$
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)
have *3*: $\vdash f \triangle ((init\ w)) \longrightarrow init\ w$
by (*simp add*: *PJ5*)
have *4*: $\vdash (init\ w \wedge f \triangle \bigcirc g) = f \triangle ((init\ w) \wedge \bigcirc g)$
by (*simp add*: *StateImportProj*)
have *5*: $\vdash f \triangle ((init\ w) \wedge \bigcirc g) \longrightarrow ((init\ w) \wedge (f \wedge more);(f \triangle g))$
using *2 3 ProjAndImp* **by** *fastforce*
from *5 4* **show** *?thesis* **using** *2* **by** *fastforce*
qed

lemma *ProjNext*:
$\vdash f \triangle \bigcirc g = (f \wedge more);(f \triangle g)$
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)

lemma *ProjWnext*:
$\vdash f \triangle (wnext\ g) = (empty \vee (f \wedge more);(f \triangle g))$
proof −
have *1*: $\vdash f \triangle (wnext\ g) = f \triangle (empty \vee \bigcirc g)$
by (*simp add*: *RightProjEqvProj WnextEqvEmptyOrNext*)
have *2*: $\vdash f \triangle (empty \vee \bigcirc g) = (empty \vee f \triangle (\bigcirc g))$
using *PJ1sema PJ2* **by** *fastforce*
have *3*: $\vdash f \triangle (\bigcirc g) = (f \wedge more);(f \triangle g)$
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)
show *?thesis*

```
 using 1 2 3 by fastforce
qed


lemma ProjIntro:
 assumes ⊢  f ∧  more  ⟶ (g ∧  more ); f
 shows  ⊢ f ⟶ g △ #True
using assms CSIntro ProjTrueEqvChopstar by force


lemma RightBoxStateImportProj:
⊢ □(init w) ∧ f △ g ⟶ f △ (□(init w) ∧ g)
by (simp add: Valid-def always-defs init-defs projection-d-def)
  ( metis diff-zero filt-expand interval-idx-bound-1 interval-intlen-gr-zero
   interval-suffix-length-good)

lemma LeftBoxStateImportProjhelp:
 (∀ n≤intlen wa. w ⟨Interval.nth wa n⟩) ∧
        (∃ l. Interval.nth l 0 = 0 ∧
             (∀ n<intlen l. Interval.nth l n < Interval.nth l (Suc n)) ∧
             Interval.nth l (intlen l) = intlen wa ∧
             (∀ i<intlen l. f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa)) ∧
             g (filt wa l)) ⟶
        (∃ l. Interval.nth l 0 = 0 ∧
             (∀ n<intlen l. Interval.nth l n < Interval.nth l (Suc n)) ∧
             Interval.nth l (intlen l) = intlen wa ∧
             (∀ i<intlen l.
                 f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) ∧
                 (∀ n≤intlen (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa).
                     w ⟨Interval.nth (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) n⟩)) ∧
             g (filt wa l))
proof
 assume 0: (∀ n≤intlen wa. w ⟨Interval.nth wa n⟩) ∧
   (∃ l. Interval.nth l 0 = 0 ∧
        (∀ n<intlen l. Interval.nth l n < Interval.nth l (Suc n)) ∧
        Interval.nth l (intlen l) = intlen wa ∧
        (∀ i<intlen l. f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa)) ∧
        g (filt wa l))
 show  ∃ l. Interval.nth l 0 = 0 ∧
       (∀ n<intlen l. Interval.nth l n < Interval.nth l (Suc n)) ∧
       Interval.nth l (intlen l) = intlen wa ∧
       (∀ i<intlen l.
           f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) ∧
           (∀ n≤intlen (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa).
               w ⟨Interval.nth (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) n⟩)) ∧
       g (filt wa l)
  proof −
   have 1: (∀ n≤intlen wa. w ⟨Interval.nth wa n⟩)
    using 0 by auto
   have 2: (∃ l. Interval.nth l 0 = 0 ∧
```

```
        (∀ n<intlen l. Interval.nth l n < Interval.nth l (Suc n)) ∧
        Interval.nth l (intlen l) = intlen wa ∧
        (∀ i<intlen l. f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa)) ∧
        g (filt wa l))
    using 0 by auto
  obtain l where 3: Interval.nth l 0 = 0 ∧
        (∀ n<intlen l. Interval.nth l n < Interval.nth l (Suc n)) ∧
        Interval.nth l (intlen l) = intlen wa ∧
        (∀ i<intlen l. f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa)) ∧
        g (filt wa l)
     using 2 by auto
  have 4: Interval.nth l 0 = 0
    using 3 by auto
  have 5: (∀ n<intlen l. Interval.nth l n < Interval.nth l (Suc n))
    using 3 by auto
  have 6: Interval.nth l (intlen l) = intlen wa
     using 3 by auto
  have 7: (∀ i<intlen l. f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa))
     using 3 by auto
  have 8: g (filt wa l)
     using 3 by auto
  have 9: (∀ i<intlen l.
          f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) ∧
          (∀ n≤(nth l (Suc i)) −(nth l i).
              w ⟨nth wa ((nth l i)+n)⟩))
  by (metis 1 7 interval-nth-last-stutter nat-le-iff-add nat-le-linear)
  have 10: (∀ i<intlen l.
             intlen (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) =
              (nth l (Suc i)) −(nth l i) )
      by (simp add: 3 PJ6help1 index-sequence-def)
  have 11: (∀ i<intlen l.
            (∀ n≤(nth l (Suc i)) −(nth l i).
              Interval.nth (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) n =
               nth wa ((nth l i)+n) ))
      using 3 index-sequence-def interval-idx-expand by fastforce
 have 12: (∀ i<intlen l.
          f (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) ∧
          (∀ n≤intlen (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa).
              w ⟨Interval.nth (sub (Interval.nth l i) (Interval.nth l (Suc i)) wa) n⟩))
   using 9 10 11 by simp
 show ?thesis
 using 12 3 by blast
 qed
qed

lemma LeftBoxStateImportProj:
 ⊢ □(init w) ∧ f △ g ⟶ (f ∧ □ (init w)) △ g
using LeftBoxStateImportProjhelp
 by (simp add: index-sequence-def Valid-def always-defs init-defs projection-d-def powerinterval-def)
 blast
```

10.6.2 dp and bp

lemma *NotDpEqvBpNot*:
$\vdash (\neg(dp\ f)) = \ bp\ (\neg\ f)$
by (*simp add*: *bp-d-def dp-d-def uprojection-d-def*)

lemma *NotBpEqvDpNot*:
$\vdash (\neg(bp\ f)) = dp(\neg\ f)$
by (*simp add*: *bp-d-def dp-d-def uprojection-d-def*)

lemma *NowImpDp*:
$\vdash f \longrightarrow dp\ f$
proof –
have *1*: $\vdash\ (skip \longrightarrow \#True)$
by *simp*
have *2*: $\vdash ba(skip \longrightarrow \#True)$
using *1* **by** (*simp add*: *BaGen*)
have *3*: $\vdash ba(skip \longrightarrow \#True) \longrightarrow (skip \bigtriangleup f \longrightarrow \#True \bigtriangleup f)$
using *PJ8* **by** *blast*
have *4*: $\vdash (skip \bigtriangleup f \longrightarrow \#True \bigtriangleup f)$
using *2 3 MP* **by** *blast*
show *?thesis*
by (*metis 4 PJ6 dp-d-def inteq-reflection*)
qed

lemma *BpElim*:
$\vdash bp\ f \longrightarrow f$
proof –
have *1*: $\vdash \neg\ f \longrightarrow dp\ (\neg\ f)$
by (*simp add*: *NowImpDp*)
hence *2*: $\vdash \neg(dp\ (\neg\ f)) \longrightarrow f$
by *auto*
from *2* **show** *?thesis*
by (*simp add*: *bp-d-def dp-d-def uprojection-d-def*)
qed

lemma *BpImpDpImpDp*:
$\vdash bp\ (f \longrightarrow g) \longrightarrow\ dp\ \ f \longrightarrow\ dp\ \ g$
proof –
have *1*: $\vdash bp\ (f \longrightarrow g)\ \longrightarrow (\#True \bigtriangleup f) \longrightarrow (\#True \bigtriangleup g)$
by (*simp add*: *PJ9 bp-d-def*)
from *1* **show** *?thesis* **by** (*simp add*: *dp-d-def*)
qed

lemma *BpContraPosImpDist*:
$\vdash bp\ (\neg\ \ g \longrightarrow \neg\ \ f) \longrightarrow (bp\ \ f) \longrightarrow (bp\ \ g)$
proof –

have *1*: $\vdash bp\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (\ dp\ (\neg\ g)) \longrightarrow (\ dp\ (\neg\ f))$
by (*rule BpImpDpImpDp*)
hence *2*: $\vdash bp\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (\neg\ (\ dp\ (\neg\ f))) \longrightarrow (\neg\ (\ dp\ (\neg\ g)))$ **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *bp-d-def dp-d-def uprojection-d-def*)
qed

lemma *BpImpDist*:
$\vdash bp\ (f \longrightarrow g) \longrightarrow (bp\ \ f) \longrightarrow (bp\ \ g)$
proof −
have *1*: $\vdash (f \longrightarrow g) \longrightarrow (\neg\ g \longrightarrow \neg\ f)$ **by** *auto*
hence *2*: $\vdash \neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g)$ **by** *auto*
hence *3*: $\vdash bp\ (\neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g))$ **by** (*rule BpGen*)
have *4*: $\vdash bp\ (\neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g))$
$\longrightarrow$
$bp\ (f \longrightarrow g) \longrightarrow bp\ (\neg\ g \longrightarrow \neg\ f)$ **by** (*rule BpContraPosImpDist*)
have *5*: $\vdash bp\ (f \longrightarrow g) \longrightarrow bp\ (\neg\ g \longrightarrow \neg\ f)$ **using** *3 4 MP* **by** *blast*
have *6*: $\vdash bp\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (bp\ \ f) \longrightarrow (bp\ \ g)$ **by** (*rule BpContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *DpImpDpRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash\ dp\ \ f \longrightarrow\ dp\ \ g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \#True \bigtriangleup f \longrightarrow \#True \bigtriangleup g$
by (*metis BpGen MP PJ9 bp-d-def*)
from *2* **show** *?thesis* **by** (*simp add*: *dp-d-def*)
qed

lemma *BpImpBpRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash bp\ \ f \longrightarrow bp\ \ g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ g \longrightarrow \neg\ f$ **by** *auto*
hence *3*: $\vdash\ dp\ (\neg\ g) \longrightarrow\ dp\ (\neg\ f)$ **by** (*rule DpImpDpRule*)
hence *4*: $\vdash \neg\ (\ dp\ (\neg\ f)) \longrightarrow \neg\ (\ dp\ (\neg\ g))$ **by** *auto*
from *4* **show** *?thesis*
by (*meson BpGen BpImpDist MP assms*)
qed

lemma *DpEqvDpRule*:
assumes $\vdash f = g$
shows $\vdash\ dp\ f =\ dp\ g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \#True \bigtriangleup f = \#True \bigtriangleup g$

using *RightProjEqvProj* **by** *blast*
from *2* **show** *?thesis* **by** (*simp add*: *dp-d-def*)
qed

lemma *BpEqvBpRule*:
assumes $\vdash f = g$
shows $\vdash bp\ f = bp\ g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence *3*: $\vdash dp\ (\neg f) = dp\ (\neg g)$ **by** (*rule DpEqvDpRule*)
hence *4*: $\vdash (\neg (dp\ (\neg f))) = (\neg (dp\ (\neg g)))$ **by** *auto*
from *4* **show** *?thesis*
by (*metis BpImpBpRule assms int-iffD1 int-iffI inteq-reflection*)
qed

lemma *DpState*:
$\vdash dp\ (init\ w) = (init\ w)$
by (*metis NowImpDp PJ5 dp-d-def int-iffI*)

lemma *StateEqvBp*:
$\vdash (init\ w) = bp\ (init\ w)$
proof −
have *1*: $\vdash (init\ w) \longrightarrow bp\ (init\ w)$
by (*metis* (*no-types*, *lifting*) *DiState DpState Initprop*(*2*) *StateEqvBi bi-d-def bp-d-def dp-d-def int-iffD1 inteq-reflection uprojection-d-def*)
have *2*: $\vdash bp\ (init\ w) \longrightarrow (init\ w)$ **by** (*rule BpElim*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DpDpEqvDp*:
$\vdash dp\ (dp\ f) = dp\ f$
proof −
have *2*: $\vdash \#True \triangle (\#True \triangle f) = (\#True \triangle \#True) \triangle f$
by (*simp add*: *PJ7*)
have *3*: $\vdash (\#True \triangle \#True) = \#True$
by (*metis DpState Initprop*(*4*) *dp-d-def int-eq-true inteq-reflection*)
show *?thesis* **by** (*metis 2 3 dp-d-def inteq-reflection*)
qed

lemma *BpBpEqvBp*:
$\vdash bp\ (bp\ f) = bp\ f$
proof −
have *1*: $\vdash dp\ (dp\ (\neg f)) = dp\ (\neg f)$
using *DpDpEqvDp* **by** *blast*
have *2*: $\vdash (\neg (dp\ (dp\ (\neg f)))) = (\neg (dp\ (\neg f)))$
using *1* **by** *auto*
have *3*: $\vdash (\neg (dp\ (\neg f))) = bp\ f$

by (*simp add*: *bp-d-def dp-d-def uprojection-d-def*)
have *4*: $\vdash (\neg\ (dp\ (dp\ (\neg\ f)))) = bp\ (bp\ f)$
by (*simp add*: *bp-d-def dp-d-def uprojection-d-def*)
from *2 3 4* **show** *?thesis*
by *fastforce*
qed

lemma *DpOrEqv*:
$\vdash\ \ dp\ (f \lor\ g) = (dp\ f \lor\ dp\ g)$
proof –
have *1*: $\vdash \#True \triangle (f \lor\ g) = (\#True \triangle f \lor \#True \triangle g)$
using *ProjOrDist* **by** *auto*
from *1* **show** *?thesis* **by** (*simp add*: *dp-d-def*)
qed

lemma *BpAndEqv*:
$\vdash bp(f \land g) = (bp\ f \land bp\ g)$
proof –
have *1*: $\vdash dp\ ((\neg\ f) \lor\ (\neg g)) = (dp\ (\neg f) \lor\ dp\ (\neg g))$
using *DpOrEqv* **by** *auto*
hence *2*: $\vdash (\neg\ (dp\ ((\neg\ f) \lor\ (\neg g)))) = (\neg(dp\ (\neg f) \lor\ dp\ (\neg g)))$
by *auto*
have *3*: $\vdash (\neg\ (dp\ ((\neg\ f) \lor\ (\neg g)))) = bp\ (\neg((\neg\ f) \lor\ (\neg g)))$
using *NotDpEqvBpNot* **by** *blast*
have *4*: $\vdash (\neg((\neg\ f) \lor\ (\neg g))) = (f \land g)$
by *auto*
hence *5*: $\vdash bp(\neg((\neg\ f) \lor\ (\neg g))) = bp(f \land g)$
by (*simp add*: *BpEqvBpRule*)
have *6*: $\vdash (\neg(dp\ (\neg f) \lor\ dp\ (\neg g))) = ((\neg(dp\ (\neg\ f))) \land (\neg(dp\ (\neg\ g))))$
by *auto*
have *7*: $\vdash ((\neg(dp\ (\neg\ f))) \land (\neg(dp\ (\neg\ g)))) = (bp\ f \land bp\ g)$
by (*simp add*: *bp-d-def dp-d-def uprojection-d-def*)
show *?thesis*
by (*metis 2 3 4 6 7 inteq-reflection*)
qed

lemma *DpAndA*:
$\vdash\ \ dp\ (f \land g) \longrightarrow\ dp\ f$
proof –
have *1*: $\vdash \#True \triangle (f \land g) \longrightarrow \#True \triangle f$
by (*meson Prop12 RightProjImpProj int-iffD1 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *dp-d-def*)
qed

lemma *BpOrA*:
$\vdash bp\ f \longrightarrow\ bp(f \lor g)$
by (*simp add*: *BpImpBpRule intI*)

lemma *BpOrB*:
$\vdash$ *bp g* $\longrightarrow$ *bp*(*f* $\vee$ *g*)
by (*simp add*: *BpImpBpRule intI*)

lemma *BpOrImpOr*:
$\vdash$ *bp f* $\vee$ *bp g* $\longrightarrow$ *bp*(*f* $\vee$ *g*)
using *BpOrA BpOrB* **by** *fastforce*

lemma *DpAndB*:
$\vdash$ *dp* (*f* $\wedge$ *g*) $\longrightarrow$ *dp g*
proof −
have *1*: $\vdash$ #*True* $\triangle$ (*f* $\wedge$ *g*) $\longrightarrow$ #*True* $\triangle$ *g*
by (*meson Prop12 RightProjImpProj int-iffD2 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *dp-d-def*)
qed

lemma *DpAndImpAnd*:
$\vdash$ *dp* (*f* $\wedge$ *g*) $\longrightarrow$ *dp f* $\wedge$ *dp g*
proof −
have *1*: $\vdash$ *dp* (*f* $\wedge$ *g*) $\longrightarrow$ *dp f* **by** (*rule DpAndA*)
have *2*: $\vdash$ *dp* (*f* $\wedge$ *g*) $\longrightarrow$ *dp g* **by** (*rule DpAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DpSkipEqvMore*:
$\vdash$ *dp skip* = *more*
proof −
have *1*: $\vdash$ *dp skip* = #*True* $\triangle$ *skip*
by (*simp add*: *dp-d-def*)
have *2*: $\vdash$ #*True* $\triangle$ *skip* = (#*True* $\wedge$ *more*)
using *PJ3* **by** *blast*
have *3*: $\vdash$ (#*True* $\wedge$ *more*) = *more*
by *auto*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DpMoreEqvMore*:
$\vdash$ *dp more* = *more*
by (*metis DpDpEqvDp DpSkipEqvMore inteq-reflection*)

lemma *BpEmptyEqvEmpty*:
$\vdash$ *bp empty* = *empty*
by (*metis DpMoreEqvMore NotDpEqvBpNot empty-d-def inteq-reflection*)

lemma *DpEmptyEqvEmpty*:
$\vdash$ *dp empty* = *empty*
proof −

have *1*: ⊢ *dp empty* = #*True* △ *empty*
by (*simp add*: *dp-d-def*)
have *2*: ⊢ #*True* △ *empty* = *empty*
by (*simp add*: *PJ2*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BpMoreEqvMore*:
⊢ *bp more* =*more*
by (*metis DpEmptyEqvEmpty NotDpEqvBpNot NotEmptyEqvMore inteq-reflection*)

lemma *NextDpImpDpNext*:
⊢ ○ (*dp f*) ⟶ *dp* (○ *f*)
proof −
have *1*: ⊢ *dp*(○ *f*) = #*True* △ (*skip*;*f*)
by (*simp add*: *dp-d-def next-d-def*)
have *2*: ⊢ #*True* △ (*skip*;*f*) = (#*True* △ *skip*);(#*True* △ *f*)
by (*simp add*: *PJ4*)
have *3*: ⊢ (#*True* △ *skip*) = (#*True* ∧ *more*)
using *PJ3* **by** *blast*
have *4*: ⊢ (#*True* ∧ *more*) = *more*
by *auto*
have *5*: ⊢ *skip*;(#*True* △ *f*) ⟶ *more*;(#*True* △ *f*)
by (*metis DpSkipEqvMore LeftChopImpChop NowImpDp inteq-reflection*)
show *?thesis*
by (*metis 2 3 4 5 dp-d-def inteq-reflection next-d-def*)
qed

lemma *BoxStateImportBp*:
⊢ □(*init w*) ⟶ *bp* (□(*init w*))
by (*simp add*: *Valid-def always-defs init-defs projection-d-def bp-d-def uprojection-d-def powerinterval-def*)
(*metis* (*mono-tags*, *lifting*) *filt-expand interval-idx-bound-1 interval-suffix-zero*)

lemma *BoxStateEqvBpBoxState*:
⊢ □ (*init w*) = *bp*(□ (*init w*))
proof −
have *1*: ⊢ *bp*(□ (*init w*)) ⟶ □ (*init w*)
by (*simp add*: *BpElim*)
have *2*: ⊢ *bp*(□ (*init w*)) = (¬(#*True* △ (¬ □ (*init w*))))
by (*simp add*: *bp-d-def uprojection-d-def*)
have *2*: ⊢ □(*init w*) ⟶ *dp* (□(*init w*))
by (*metis NowImpDp*)
have *2*: ⊢ □ (*init w*) ⟶ *bp*(□ (*init w*))
using *BoxStateImportBp* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

end

11 Infinite ITL Semantics

theory *InfiniteSemantics*
imports *InfiniteInterval HOL−TLA.Intensional*
begin

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

11.1 Types of Formulas

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

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

type-synonym (*'a*,*'b*) *formfun* = *'a intervals* ⇒ *'b*
type-synonym (*'a*,*'b*) *finformfun* = *'a interval* ⇒ *'b*
type-synonym (*'a*,*'b*) *infformfun* = *'a infinterval* ⇒ *'b*
type-synonym *'a formula* = (*'a*,*bool*) *formfun*
type-synonym *'a finformula* = (*'a*,*bool*) *finformfun*
type-synonym *'a infformula* = (*'a*,*bool*) *infformfun*
type-synonym (*'a*,*'b*) *stfun* = *'a* ⇒ *'b*
type-synonym *'a stpred* = (*'a*,*bool*) *stfun*

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

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

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

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

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

11.2 Semantics of ITL

The semantics of ITL is defined.

definition *skip-d* :: (*'a* ::*world*) *formula*
where
 skip-d ≡ (λ*s*. (*case s of* (*Inl s*) ⇒ (*intlen s* =*1*) | (*Inr s*) ⇒ *False*))

definition *chop-d* :: ('a ::world) formula ⇒ ('a ::world) formula ⇒ ('a ::world) formula
where
```
 chop-d F1 F2 ≡
  (λs.
    (case s of (Inl s) ⇒
      (∃ n. n≥0 ∧ n≤intlen s ∧ ((Inl (prefix n s)) ⊨ F1) ∧ ((Inl (suffix n s)) ⊨ F2))
     |
             (Inr s) ⇒
      ( (∃ n. ((Inl (iprefix n s)) ⊨ F1) ∧ ((Inr (isuffix n s)) ⊨ F2) )
        ∨ ((Inr s) ⊨ F1)
      )
    )
  )
```

definition *current-val-d* :: ('a::world,'b) stfun ⇒ ('a,'b) formfun
where
```
 current-val-d f = (λs. (case s of (Inl s) ⇒ ((nth s 0) ⊨ f) | (Inr s) ⇒ ((s 0) ⊨ f)))
```

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

definition *fin-val-d* :: ('a::world,'b) stfun ⇒ ('a,'b) formfun
```
where fin-val-d f ≡ λ s. (case s of (Inl s) ⇒ (nth s (intlen s)) ⊨ f
                                  | (Inr s) ⇒ (ϵ (x::'b ). x=x))
```

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

syntax
```
 -skip-d          :: lift                ((skip))
 -chop-d          :: [lift,lift] ⇒ lift ((-;-) [84,84] 83)
 -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*)
-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*
-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*:: (- *intervals*) ⇒ -)

11.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

definition *infinite-d* :: (*'a* ::*world*) *formula*
where
infinite-d ≡ *LIFT*(#*True*;#*False*)

syntax
-infinite-d :: *lift* (*inf*)

syntax (*ASCII*)
-infinite-d :: *lift* (*inf*)

translations
-infinite-d ⇌ *CONST infinite-d*

definition *finite-d* :: (*'a* ::*world*) *formula*
where
finite-d ≡ *LIFT*(¬(*inf*))

syntax
-finite-d :: *lift* (*finite*)

syntax (*ASCII*)
-finite-d :: *lift* (*finite*)

translations
-finite-d ⇌ *CONST finite-d*

definition *schop-d* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *schop-d F1 F2* ≡ *LIFT*((*F1* ∧ *finite*);*F2*)

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

syntax

-schop-d :: [*lift*,*lift*] ⇒ *lift* ((- ⌢ -) [*84*,*84*] *83*)
-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*)
-schop-d :: [*lift*,*lift*] ⇒ *lift* ((- *schop* -) [*84*,*84*] *83*)
-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
-schop-d ⇌ *CONST schop-d*
-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 *df-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *df-d F* ≡ *LIFT*(*F*⌢#*True*)

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

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
-df-d :: *lift* ⇒ *lift* ((*df* -) [*88*] *87*)
-sda-d :: *lift* ⇒ *lift* ((*sda* -) [*88*] *87*)
-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*)
-df-d :: *lift* ⇒ *lift* ((*df* -) [*88*] *87*)
-sda-d :: *lift* ⇒ *lift* ((*sda* -) [*88*] *87*)
-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
-df-d ⇌ *CONST df-d*
-sda-d ⇌ *CONST sda-d*
-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 *bf-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *bf-d F* ≡ *LIFT*(¬(*df*(¬*F*)))

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

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

definition *fmore-d* :: (*'a*::*world*) *formula*
where *fmore-d* ≡ *LIFT*(*more* ∧ *finite*)

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

syntax
-bf-d :: *lift* ⇒ *lift* ((*bf* -) [*88*] *87*)
-sba-d :: *lift* ⇒ *lift* ((*sba* -) [*88*] *87*)
-empty-d :: *lift* ((*empty*))
-fmore-d :: *lift* ((*fmore*))
-dm-d :: *lift* ⇒ *lift* ((*dm* -) [*88*] *87*)

syntax (*ASCII*)
-bf-d :: *lift* ⇒ *lift* ((*bf* -) [*88*] *87*)
-sba-d :: *lift* ⇒ *lift* ((*sba* -) [*88*] *87*)
-empty-d :: *lift* ((*empty*))
-fmore-d :: *lift* ((*fmore*))
-dm-d :: *lift* ⇒ *lift* ((*dm* -) [*88*] *87*)

translations
-bf-d ⇌ *CONST bf-d*
-sba-d ⇌ *CONST sba-d*
-empty-d ⇌ *CONST empty-d*
-fmore-d ⇌ *CONST fmore-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* $\Rightarrow$ $'a$ *formula*
where *initonly-d F* $\equiv$ *LIFT*(*bi*(*empty* = *F*))

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

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

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

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

primrec *power-d* :: ($'a$::*world*) *formula* $\Rightarrow$ *nat* $\Rightarrow$ $'a$ *formula*
where *pow-0* : (*power-d F 0*) = *LIFT*(*empty*)
| *pow-Suc*: (*power-d F* (*Suc n*)) = *LIFT*((*F* $\wedge$ *finite*);(*power-d F n*))

primrec *spower-d* :: ($'a$::*world*) *formula* $\Rightarrow$ *nat* $\Rightarrow$ $'a$ *formula*
where *spow-0* : (*spower-d F 0*) = *LIFT*(*empty*)
| *spow-Suc*: (*spower-d F* (*Suc n*)) = *LIFT*(*F*$\frown$(*spower-d F n*))

syntax
-*bm-d* :: *lift* $\Rightarrow$ *lift* ((*bm* -) [*88*] *87*)
-*init-d* :: *lift* $\Rightarrow$ *lift* ((*init* -) [*88*] *87*)
-*fin-d* :: *lift* $\Rightarrow$ *lift* ((*fin* -) [*88*] *87*)
-*halt-d* :: *lift* $\Rightarrow$ *lift* ((*halt* -) [*88*] *87*)
-*initonly-d* :: *lift* $\Rightarrow$ *lift* ((*initonly* -) [*88*] *87*)
-*keep-d* :: *lift* $\Rightarrow$ *lift* ((*keep* -) [*88*] *87*)
-*yields-d* :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *yields* -) [*88*,*88*] *87*)
-*syields-d* :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *syields* -) [*88*,*88*] *87*)
-*ifthenelse-d* :: [*lift*,*lift*,*lift*] $\Rightarrow$ *lift* ((*if* $_i$ - *then* - *else* -) [*88*,*88*,*88*] *87*)
-*power-d* :: [*lift*,*nat*] $\Rightarrow$ *lift* ((*power* - -) [*88*,*88*] *87*)
-*spower-d* :: [*lift*,*nat*] $\Rightarrow$ *lift* ((*spower* - -) [*88*,*88*] *87*)

syntax (*ASCII*)
-*bm-d* :: *lift* $\Rightarrow$ *lift* ((*bm* -) [*88*] *87*)
-*init-d* :: *lift* $\Rightarrow$ *lift* ((*init* -) [*88*] *87*)
-*fin-d* :: *lift* $\Rightarrow$ *lift* ((*fin* -) [*88*] *87*)
-*halt-d* :: *lift* $\Rightarrow$ *lift* ((*halt* -) [*88*] *87*)
-*initonly-d* :: *lift* $\Rightarrow$ *lift* ((*initonly* -) [*88*] *87*)
-*keep-d* :: *lift* $\Rightarrow$ *lift* ((*keep* -) [*88*] *87*)
-*yields-d* :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *yields* -) [*88*,*88*] *87*)
-*syields-d* :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *syields* -) [*88*,*88*] *87*)

-ifthenelse-d :: [*lift*,*lift*,*lift*] ⇒ *lift* ((*if* $_i$ *- then - else -*) [*88*,*88*,*88*] *87*)
-power-d :: [*lift*,*nat*] ⇒ *lift* ((*power - -*) [*88*,*88*] *87*)
-spower-d :: [*lift*,*nat*] ⇒ *lift* ((*spower - -*) [*88*,*88*] *87*)

translations
-bm-d ⇌ *CONST bm-d*
-init-d ⇌ *CONST init-d*
-fin-d ⇌ *CONST fin-d*
-halt-d ⇌ *CONST halt-d*
-initonly-d ⇌ *CONST initonly-d*
-keep-d ⇌ *CONST keep-d*
-yields-d ⇌ *CONST yields-d*
-syields-d ⇌ *CONST syields-d*
-ifthenelse-d ⇌ *CONST ifthenelse-d*
-power-d ⇌ *CONST power-d*
-spower-d ⇌ *CONST spower-d*

definition *len-d* :: *nat* ⇒ (*'a*::*world*) *formula*
where *len-d n* ≡ *LIFT*(*power skip n*)

definition *fpowerstar-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *fpowerstar-d F* ≡ *LIFT*(∃ *k*. *power F k*)

definition *spowerstar-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *spowerstar-d F* ≡ *LIFT*(∃ *k*. *spower F k*)

definition *omega-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *omega-d F* ≡ (λ*s*.
(*case s of* (*Inl s*) ⇒ *False*
| (*Inr s*) ⇒
(∃(*l*::*infiniteindex*). *infinite-index-sequence 0 l* ∧
(∀ *i*.
((*Inl* (*subinterval s* (*l i*) (*l* (*Suc i*)))) ⊨ *F*)
)
)
))

syntax
-len-d :: *nat* ⇒ *lift* ((*len -*) [*88*] *87*)
-fpowerstar-d :: *lift* ⇒ *lift* ((*fpowerstar -*) [*85*] *85*)
-spowerstar-d :: *lift* ⇒ *lift* ((*spowerstar -*) [*85*] *85*)
-omega-d :: *lift* ⇒ *lift* ((-$^\omega$) [*85*] *85*)

syntax (*ASCII*)
-len-d :: *nat* ⇒ *lift* ((*len -*) [*88*] *87*)
-fpowerstar-d :: *lift* ⇒ *lift* ((*fpowerstar -*) [*85*] *85*)

-spowerstar-d :: *lift* ⇒ *lift* ((*spowerstar* -) [*85*] *85*)
-omega-d :: *lift* ⇒ *lift* ((*omega* -) [*85*] *85*)

translations
-len-d ⇌ *CONST len-d*
-fpowerstar-d ⇌ *CONST fpowerstar-d*
-spowerstar-d ⇌ *CONST spowerstar-d*
-omega-d ⇌ *CONST omega-d*

definition *powerstar-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *powerstar-d F* ≡ *LIFT*((∃ *k*. *power F k*);(*empty* ∨ (*F* ∧ *inf*)))

syntax
-powerstar-d :: *lift* ⇒ *lift* ((*powerstar* -) [*85*] *85*)

syntax (*ASCII*)
-powerstar-d :: *lift* ⇒ *lift* ((*powerstar* -) [*85*] *85*)

translations
-powerstar-d ⇌ *CONST powerstar-d*

definition *chopstar-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *chopstar-d F* ≡ *LIFT*(*powerstar* (*F* ∧ *more*))

definition *schopstar-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *schopstar-d F* ≡ *LIFT*(*spowerstar* (*F* ∧ *more*))

syntax
-chopstar-d :: *lift* ⇒ *lift* ((-*) [*85*] *85*)
-schopstar-d ::*lift* ⇒ *lift* ((*schopstar* -) [*85*] *85*)

syntax (*ASCII*)
-chopstar-d :: *lift* ⇒ *lift* ((*chopstar* -) [*85*] *85*)
-schopstar-d ::*lift* ⇒ *lift* ((*schopstar* -) [*85*] *85*)

translations
-chopstar-d ⇌ *CONST chopstar-d*
-schopstar-d ⇌ *CONST schopstar-d*

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

definition *ifthen-d* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*

where $ifthen\text{-}d\ F\ G \equiv LIFT(if_{\,i}\ F\ then\ G\ else\ \#True\ )$

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

syntax
$-ifthen\text{-}d$ :: $[lift,lift] \Rightarrow\ lift\ ((if_{\,i}\ \text{-}\ then\ \text{-}\ )\ \ [88,88]\ 87)$
$-while\text{-}d$ :: $[lift,lift] \Rightarrow\ lift\ ((while\ \text{-}\ do\ \text{-}\ )\ \ [88,88]\ 87)$
$-sfin\text{-}d$ :: $lift \Rightarrow\ lift\ \ ((sfin\ \text{-})\ [88]\ 87)$

syntax $(ASCII)$
$-ifthen\text{-}d$ :: $[lift,lift] \Rightarrow\ lift\ ((if_{\,i}\ \text{-}\ then\ \text{-}\ )\ \ [88,88]\ 87)$
$-while\text{-}d$ :: $[lift,lift] \Rightarrow\ lift\ ((while\ \text{-}\ do\ \text{-}\ )\ \ [88,88]\ 87)$
$-sfin\text{-}d$ :: $lift \Rightarrow\ lift\ \ ((sfin\ \text{-})\ [88]\ 87)$

translations
$-ifthen\text{-}d \rightleftharpoons CONST\ ifthen\text{-}d$
$-while\text{-}d \rightleftharpoons CONST\ while\text{-}d$
$-sfin\text{-}d \rightleftharpoons CONST\ sfin\text{-}d$

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

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

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

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

translations
$-swhile\text{-}d \rightleftharpoons CONST\ swhile\text{-}d$
$-repeat\text{-}d \rightleftharpoons CONST\ repeat\text{-}d$

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

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

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

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

definition *temporal-assign-d* :: ('a::world,'b) stfun ⇒ ('a,'b) formfun ⇒ 'a formula
where *temporal-assign-d v e* ≡ λ s. s ⊨ finite ⟶ !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
-srepeat-d :: [lift,lift] ⇒ lift ((srepeat - until -) [88,88] 87)
-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*)
-srepeat-d :: [lift,lift] ⇒ lift ((srepeat - until -) [88,88] 87)
-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
-srepeat-d ⇌ CONST srepeat-d
-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

11.4 Properties of Operators

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

lemma *skip-defs* :
$(w \models skip) = (case\ w\ of\ (Inl\ w) \Rightarrow (intlen\ w = 1) \mid (Inr\ w) \Rightarrow False)$
by (*simp add*: *skip-d-def*)

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

lemma *skip-defs-infinite* :
$\neg\ ((Inr\ w) \models skip)$
by (*simp add*: *skip-d-def*)

lemma *chop-defs* :
$(w \models F1\ ;\ F2) =$
$(case\ w\ of\ (Inl\ w) \Rightarrow$
$(\exists\, n.\ n \geq 0 \wedge n \leq intlen\ w \wedge ((Inl\ (prefix\ n\ w)) \models F1\ ) \wedge ((Inl\ (suffix\ n\ w)) \models F2\ )\ )$
$\mid (Inr\ w) \Rightarrow$
$(\ (\exists\, n.\ ((Inl\ (iprefix\ n\ w)) \models F1\ ) \wedge ((Inr\ (isuffix\ n\ w)) \models F2))$
$\vee ((Inr\ w) \models F1)$
$)$
$)$

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

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

lemma *chop-defs-infinite*:
$(\ (Inr\ w) \models F1;F2) =$
$(\ (\exists\, n.\ (Inl\ (iprefix\ n\ w) \models F1\ ) \wedge (Inr\ (isuffix\ n\ w) \models F2\ ))$
$\vee (Inr\ w \models F1\ )$
$)$
by (*simp add*: *chop-d-def*)

lemma *infinite-defs*:
$(w \models inf) = (case\ w\ of\ (Inr\ w) \Rightarrow True \mid (Inl\ w) \Rightarrow False)$
by (*simp add*: *infinite-d-def chop-d-def sum.case-eq-if*)

lemma *infinite-defs-1*:
$((Inr\ w) \models inf)$
by (*simp add*: *infinite-d-def chop-d-def sum.case-eq-if*)

lemma *finite-defs* :
$(w \models \mathit{finite}) = (\mathit{case}\ w\ \mathit{of}\ \ (\mathit{Inl}\ w) \Rightarrow\ \mathit{True} \mid (\mathit{Inr}\ w) \Rightarrow\ \mathit{False})$
by (*simp add*: *finite-d-def infinite-defs chop-d-def sum.case-eq-if*)

lemma *finite-defs-1* :
$((\mathit{Inl}\ w) \models \mathit{finite})$
by (*simp add*: *finite-defs sum.case-eq-if*)

lemma *schop-defs* :
$(w \models F1 \frown F2) =$
$(\mathit{case}\ w\ \mathit{of}\ \ (\mathit{Inl}\ w) \Rightarrow$
$(\exists\, n.\ n{\geq}0 \wedge n{\leq}\mathit{intlen}\ w \wedge ((\mathit{Inl}\ (\mathit{prefix}\ n\ w)) \models F1\ )\ \wedge ((\mathit{Inl}\ (\mathit{suffix}\ n\ w)) \models F2\ )\ )$
$\mid (\mathit{Inr}\ w) \Rightarrow$
$(\ (\exists\, n.\ \ ((\mathit{Inl}\ (\mathit{iprefix}\ n\ w)) \models F1\ ) \wedge ((\mathit{Inr}\ (\mathit{isuffix}\ n\ w)) \models F2))$
$)$
$)$

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

lemma *schop-defs-finite* :
$((\mathit{Inl}\ w) \models F1 \frown F2) =$
$(\exists\, n.\ n{\geq}0 \wedge n{\leq}\mathit{intlen}\ w \wedge ((\mathit{Inl}\ (\mathit{prefix}\ n\ w)) \models F1\ )\ \wedge ((\mathit{Inl}\ (\mathit{suffix}\ n\ w)) \models F2\ )\ )$
by (*simp add*: *schop-defs*)

lemma *schop-defs-infinite* :
$((\mathit{Inr}\ w) \models F1 \frown F2) =$
$(\exists\, n.\ \ ((\mathit{Inl}\ (\mathit{iprefix}\ n\ w)) \models F1\ ) \wedge ((\mathit{Inr}\ (\mathit{isuffix}\ n\ w)) \models F2))$

by (*simp add*: *schop-defs*)

lemma *sometimes-defs* :
$(w \models \Diamond\ F) =$
$(\mathit{case}\ w\ \mathit{of}\ \ (\mathit{Inl}\ w) \Rightarrow (\exists\ \ n.\ 0{\leq}n \wedge n{\leq}\mathit{intlen}\ w \wedge ((\mathit{Inl}\ (\mathit{suffix}\ n\ w)) \models F))$
$\mid (\mathit{Inr}\ w) \Rightarrow (\exists\ \ n.\ ((\mathit{Inr}\ (\mathit{isuffix}\ n\ w)) \models F))$
$)$

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

lemma *always-defs* :
$(w \models \Box\ F) =$
$(\mathit{case}\ w\ \mathit{of}\ \ (\mathit{Inl}\ w) \Rightarrow (\forall\ \ n.\ 0{\leq}n \wedge n{\leq}\mathit{intlen}\ w \longrightarrow ((\mathit{Inl}\ (\mathit{suffix}\ n\ w)) \models F))$
$\mid (\mathit{Inr}\ w) \Rightarrow (\forall\ \ n.\ \ ((\mathit{Inr}\ (\mathit{isuffix}\ n\ w)) \models F\ ))$
$)$
by (*simp add*: *always-d-def sometimes-defs sum.case-eq-if*)

lemma *di-defs* :
$(w \models \mathit{di}\ F) =$
$(\mathit{case}\ w\ \mathit{of}\ \ (\mathit{Inl}\ w) \Rightarrow (\exists\ \ n.\ 0{\leq}n \wedge n{\leq}\mathit{intlen}\ w \wedge ((\mathit{Inl}\ (\mathit{prefix}\ n\ w)) \models F))$
$\mid (\mathit{Inr}\ w) \Rightarrow (\exists\ \ n.\ ((\mathit{Inl}\ (\mathit{iprefix}\ n\ w)) \models F)) \vee ((\mathit{Inr}\ w) \models F)$

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

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

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

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

lemma *da-defs* :
$(w \models da\ F) =$
$(case\ w\ of\ (Inl\ w) \Rightarrow (\exists\ n\ na.\ \ n+na \leq intlen\ w \wedge ((Inl\ (sub\ n\ (n+ na)\ w)) \models F))$
$\quad \mid (Inr\ w) \Rightarrow (\exists\ n\ na.\ ((Inl\ (subinterval\ w\ n\ (n+na)\ )) \models F)$
$\qquad \vee\ (\ (Inr\ (isuffix\ n\ w)) \models F))$
)
proof
(*auto simp add*: *da-d-def chop-defs finite-d-def infinite-d-def iprefix-isuffix sum.case-eq-if*)
show $\bigwedge n\ na.$
$isl\ w \Longrightarrow$
$n \leq intlen\ (projl\ w) \Longrightarrow$
$na \leq intlen\ (projl\ w) - n \Longrightarrow F\ (Inl\ (prefix\ na\ (suffix\ n\ (projl\ w)))) \Longrightarrow$
$\exists n\ na.\ n + na \leq intlen\ (projl\ w) \wedge F\ (Inl\ (sub\ n\ (n + na)\ (projl\ w)))$
by (*metis Nat.le-diff-conv2 add.commute interval-sub-prefix-suffix-0 zero-le*)
show $\bigwedge n\ na.$
$isl\ w \Longrightarrow$
$n + na \leq intlen\ (projl\ w) \Longrightarrow$
$F\ (Inl\ (sub\ n\ (n + na)\ (projl\ w))) \Longrightarrow$
$\exists n \leq intlen\ (projl\ w).\ \exists na \leq intlen\ (projl\ w) - n.\ F\ (Inl\ (prefix\ na\ (suffix\ n\ (projl\ w))))$
by (*metis Interval.sub-def add-leD1 interval-intlen-sub interval-pref-intlen-bound interval-suffix-length le-add1*)
show $\bigwedge n\ na.$
$\neg\ isl\ w \Longrightarrow F\ (Inl\ (subinterval\ (projr\ w)\ n\ (na + n))) \Longrightarrow$
$\exists n.\ (\exists na.\ F\ (Inl\ (subinterval\ (projr\ w)\ n\ (n + na)))) \vee F\ (Inr\ (isuffix\ n\ (projr\ w)))$
by (*metis add.commute*)

show $\bigwedge n\ na.$
$\neg\ isl\ w \Longrightarrow F\ (Inl\ (subinterval\ (projr\ w)\ n\ (n + na))) \Longrightarrow$
$\exists\, n.\ (\exists\, na.\ F\ (Inl\ (subinterval\ (projr\ w)\ n\ (na + n)))) \lor F\ (Inr\ (isuffix\ n\ (projr\ w)))$
by (*metis add.commute*)
qed

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

lemma *sda-defs* :
$(w \models sda\ F) =$
$(case\ w\ of\ (Inl\ w) \Rightarrow (\exists\ n\ na.\ n{+}na \leq intlen\ w \land ((Inl\ (sub\ n\ (n{+}na)\ w)) \models F))$
$\mid (Inr\ w) \Rightarrow (\exists\ n\ na.\ ((Inl\ (subinterval\ w\ n\ (n{+}na)\ )) \models F))$
)
proof
(*auto simp add*: *sda-d-def schop-defs iprefix-isuffix sum.case-eq-if*)
show $\bigwedge n\ na.$
$isl\ w \Longrightarrow$
$n \leq intlen\ (projl\ w) \Longrightarrow$
$na \leq intlen\ (projl\ w) - n \Longrightarrow F\ (Inl\ (prefix\ na\ (suffix\ n\ (projl\ w)))) \Longrightarrow$
$\exists\, n\ na.\ n + na \leq intlen\ (projl\ w) \land F\ (Inl\ (sub\ n\ (n + na)\ (projl\ w)))$
by (*metis Nat.le-diff-conv2 interval-sub-prefix-suffix-0 le-add-diff-inverse*
nat-add-left-cancel-le zero-le)
show $\bigwedge n\ na.$
$isl\ w \Longrightarrow$
$n + na \leq intlen\ (projl\ w) \Longrightarrow$
$F\ (Inl\ (sub\ n\ (n + na)\ (projl\ w))) \Longrightarrow$
$\exists\, n{\leq}intlen\ (projl\ w).\ \exists\, na{\leq}intlen\ (projl\ w) - n.\ F\ (Inl\ (prefix\ na\ (suffix\ n\ (projl\ w))))$
by (*metis Interval.sub-def add-leD1 interval-intlen-sub interval-pref-intlen-bound*
interval-suffix-length le-add1)
show $\bigwedge n\ na.\ \neg\ isl\ w \Longrightarrow F\ (Inl\ (subinterval\ (projr\ w)\ n\ (na + n))) \Longrightarrow$
$\exists\, n\ na.\ F\ (Inl\ (subinterval\ (projr\ w)\ n\ (n + na)))$
by (*metis add.commute*)
show $\bigwedge n\ na.\ \neg\ isl\ w \Longrightarrow F\ (Inl\ (subinterval\ (projr\ w)\ n\ (n + na))) \Longrightarrow$
$\exists\, n\ na.\ F\ (Inl\ (subinterval\ (projr\ w)\ n\ (na + n)))$
by (*metis add.commute*)
qed

lemma *sba-defs* :
$(w \models sba\ F) =$
$(case\ w\ of\ (Inl\ w) \Rightarrow (\forall\ n\ na.\ n{+}na \leq intlen\ w \longrightarrow ((Inl\ (sub\ n\ (n{+}na)\ w)) \models F))$
$\mid (Inr\ w) \Rightarrow (\forall\ n\ na.\ ((Inl\ (subinterval\ w\ n\ (n{+}na)\ )) \models F))$
)
by (*simp add*: *sba-d-def sda-defs sum.case-eq-if*)

lemma *next-defs* :
 $(w \models \bigcirc\ F) =$
 $(case\ w\ of\ (Inl\ w) \Rightarrow (intlen\ w > 0 \wedge ((Inl\ (suffix\ 1\ w)) \models F)\ )$
 $\quad\quad | \ (Inr\ w) \Rightarrow ((Inr\ (isuffix\ 1\ w)) \models F)$
 $)$
using *Suc-le-eq min.absorb1*
by (*simp add*: *next-d-def chop-defs skip-d-def iprefix-length sum.case-eq-if*)
 force

lemma *wnext-defs* :
 $(w \models wnext\ F) =$
 $(case\ w\ of\ (Inl\ w) \Rightarrow (intlen\ w = 0 \vee ((Inl\ (suffix\ 1\ w)) \models F)\ )$
 $\quad\quad | \ (Inr\ w) \Rightarrow ((Inr\ (isuffix\ 1\ w)) \models F)$
 $)$
by (*simp add*: *wnext-d-def next-defs sum.case-eq-if*)

lemma *prev-defs* :
 $(w \models prev\ F) =$
 $(case\ w\ of\ (Inl\ w) \Rightarrow (intlen\ w > 0 \wedge ((Inl\ (prefix\ ((intlen\ w)-1)\ w)) \models F)\ )$
 $\quad\quad | \ (Inr\ w) \Rightarrow (Inr\ w) \models F$
 $)$
by (*simp add*: *prev-d-def chop-defs skip-d-def sum.case-eq-if*)
 (*metis One-nat-def Suc-leI diff-diff-cancel diff-le-self interval-suffix-length-good*
 le-zero-eq neq0-conv zero-neq-one)

lemma *wprev-defs* :
 $(w \models wprev\ F) =$
 $(case\ w\ of\ (Inl\ w) \Rightarrow (intlen\ w = 0 \vee ((Inl\ (prefix\ ((intlen\ w)-1)\ w)) \models F)\ )$
 $\quad\quad | \ (Inr\ w) \Rightarrow (Inr\ w) \models F$
 $)$
by (*simp add*: *wprev-d-def prev-defs sum.case-eq-if*)

lemma *more-defs* :
 $(w \models more) =$
 $(case\ w\ of\ (Inl\ w) \Rightarrow (intlen\ w > 0)$
 $\quad\quad | \ (Inr\ w) \Rightarrow True$
 $)$
by (*simp add*: *more-d-def next-defs sum.case-eq-if*)

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

lemma *empty-defs* :
 $(w \models empty) =$
 $(case\ w\ of\ (Inl\ w) \Rightarrow (intlen\ w = 0)$

| (*Inr w*) ⇒ *False*
)
by (*simp add*: *empty-d-def more-defs sum.case-eq-if*)

lemma *init-defs* :
(*w* ⊨ *init F*) =
(*case w of* (*Inl w*) ⇒ ((*Inl* (*prefix 0 w*)) ⊨ *F*)
| (*Inr w*) ⇒ ((*Inl* (*iprefix 0 w*)) ⊨ *F*)
)
using *min.absorb1*
by (*simp add*: *init-d-def chop-defs empty-defs iprefix-length sum.case-eq-if*)
force

lemma *init-defs-finite*:
((*Inl w*) ⊨ *init F*) = ((*Inl* (*prefix 0 w*)) ⊨ *F*)
by (*simp add*: *init-defs*)

lemma *init-defs-infinite*:
((*Inr w*) ⊨ *init F*) = ((*Inl* (*iprefix 0 w*)) ⊨ *F*)
by (*simp add*: *init-defs*)

lemma *initalt-defs* :
(*w* ⊨ *bi*(*empty* ⟶ *F*)) =
(*case w of* (*Inl w*) ⇒ ((*Inl* (*prefix 0 w*)) ⊨ *F*)
| (*Inr w*) ⇒ ((*Inl* (*iprefix 0 w*)) ⊨ *F*)
)
using *min.absorb1*
by (*simp add*: *bi-defs empty-defs iprefix-length sum.case-eq-if*)
force

lemma *fin-defs* :
(*w* ⊨ *fin F*) =
(*case w of* (*Inl w*) ⇒ ((*Inl* (*suffix* (*intlen w*) *w*)) ⊨ *F*)
| (*Inr w*) ⇒ *True*
)
by (*simp add*: *fin-d-def empty-defs always-defs sum.case-eq-if*)

lemma *finalt-defs* :
(*w* ⊨ #*True*;(*F* ∧ *empty*)) =
(*case w of* (*Inl w*) ⇒ ((*Inl* (*suffix* (*intlen w*) *w*)) ⊨ *F*)
| (*Inr w*) ⇒ *True*
)
by (*simp add*: *chop-defs empty-defs sum.case-eq-if*) *fastforce*

lemma *sfin-defs* :
(*w* ⊨ *sfin F*) =
(*case w of* (*Inl w*) ⇒ ((*Inl* (*suffix* (*intlen w*) *w*)) ⊨ *F*)
| (*Inr w*) ⇒ *False*
)

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

lemma halt-defs :
  (w ⊨ halt(F)) =
    (case w of (Inl w) ⇒ (∀ n≤intlen w. (intlen w = n) = ( (Inl (suffix n w)) ⊨ F))
            | (Inr w) ⇒ (∀ n. ¬( (Inr (isuffix n w)) ⊨ F))
    )
by (simp add: halt-d-def empty-defs always-defs sum.case-eq-if)

lemma initonly-defs :
  (w ⊨ initonly(F)) =
   (case w of (Inl w) ⇒ (∀ n≤intlen w. (n = 0) = ( (Inl (prefix n w)) ⊨ F ))
           | (Inr w) ⇒ (∀ n. (n = 0) = ( (Inl (iprefix n w)) ⊨ F )) ∧ ¬((Inr w) ⊨ F)
   )
by (simp add: min.absorb1 initonly-d-def bi-defs empty-defs iprefix-length sum.case-eq-if)

lemma ifthenelse-defs:
  (w ⊨ if_i F then G else H) =
   ( ((w ⊨ F) ∧ (w ⊨ G)) ∨ (( ¬(w ⊨ F) ∧ (w ⊨ H))) )
by (simp add: ifthenelse-d-def)

lemma len-defs :
  (w ⊨ len n) =
    (case w of (Inl w) ⇒ (intlen w =n)
            | (Inr w) ⇒ False
    )
proof
   (simp add: len-d-def  sum.case-eq-if)
   show (isl w ⟶ (w ⊨ (power skip n)) = (intlen (projl w) = n)) ∧
         (¬ isl w ⟶ ¬ (w ⊨ (power skip n)))
   proof (induct n arbitrary:w)
   case 0
   then show ?case by (simp add: empty-defs sum.case-eq-if)
   next
   case (Suc n)
   then show ?case
      by (auto simp add: len-d-def chop-defs skip-defs finite-defs sum.case-eq-if)
          auto
   qed
qed

lemma currentval-defs :
  (s ⊨ $v) =
   (case s of (Inl s) ⇒ (v (nth s 0))
           | (Inr s) ⇒ (v (s 0))
   )
by (simp add: current-val-d-def)

lemma nextval-defs :
```

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

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

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

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

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

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

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

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

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

lemma gets-defs :
  (s ⊨ v gets e) =
  (case s of (Inl s) ⇒ (∀ i < intlen s. v (Interval.nth s (Suc i)) = e (Inl (sub i (i+1) s)) )
          | (Inr s) ⇒ (∀ i. v ( s (Suc i)) = e (Inl (subinterval s i (i+1))) )
  )
using Suc-leI Suc-le-lessD
by (simp add: min.absorb1 finite-defs gets-d-def keep-d-def ba-defs skip-defs sub-def
           temporal-assign-d-def fin-val-d-def subinterval-length subinterval-nth sum.case-eq-if)
 auto

lemma stable-defs-helpa:
assumes (∀ i<intlen s. v (Interval.nth s (Suc i)) = v (Interval.nth s i))
        i ≤ intlen s
 shows (v (Interval.nth s i) = v (Interval.nth s 0))
using assms
proof (induct s arbitrary:i)
case (St x)
then show ?case by simp
next
case (Cons x1a s)
then show ?case
  proof (cases i)
  case 0
  then show ?thesis by blast
  next
  case (Suc nat)
  then show ?thesis
     by (metis Cons.hyps Cons.prems(1) Cons.prems(2) Suc-le-mono Suc-mono interval-nth-Suc
         interval-nth-zero intlen.simps(2) plus-1-eq-Suc zero-less-Suc)
  qed
qed

lemma stable-defs-helpb:
assumes (∀ i≤intlen s. v (Interval.nth s i) = v (Interval.nth s 0))
        i < intlen s
 shows v (Interval.nth s (Suc i)) = v (Interval.nth s i)
using assms
proof (induct s arbitrary:i)
case (St x)
then show ?case by simp
next
case (Cons x1a s)
then show ?case
   proof (cases i)
   case 0
   then show ?thesis using Suc-leI Cons.prems(1) Cons.prems(2) by blast
   next
   case (Suc nat)
```

then show *?thesis* **using** *Cons.prems*(*1*) *Cons.prems*(*2*) *Suc-leI less-imp-le-nat* **by** *presburger*
qed
qed

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 −
have *1*: ($\forall i<intlen\ s.\ v\ (Interval.nth\ s\ (Suc\ i)) = v\ (Interval.nth\ s\ i)) \longrightarrow$
($\forall i\leq intlen\ s.\ v\ (Interval.nth\ s\ i) = v\ (Interval.nth\ s\ 0)$)
using *stable-defs-helpa* **by** *auto*
have *2*: ($\forall i\leq intlen\ s.\ v\ (Interval.nth\ s\ i) = v\ (Interval.nth\ s\ 0)) \longrightarrow$
($\forall i<intlen\ s.\ v\ (Interval.nth\ s\ (Suc\ i)) = v\ (Interval.nth\ s\ i)$)
using *stable-defs-helpb* **by** *blast*
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *stable-defs-infinite*:
($\forall i.\ v\ (s\ (Suc\ i)) = v\ (s\ i)) = (\forall i.\ v\ (s\ i) = v\ (s\ 0)$)
proof −
have *1*: ($\forall i.\ v\ (s\ (Suc\ i)) = v\ (s\ i)) \Longrightarrow (\bigwedge j.\ v\ (s\ j) = v\ (s\ 0)$)
proof −
assume *A1*: ($\forall i.\ v\ (s\ (Suc\ i)) = v\ (s\ i)$)
show ($\bigwedge j.\ v\ (s\ j) = v\ (s\ 0)$)
proof −
fix *j*
show $v\ (s\ j) = v\ (s\ 0)$
using *A1* **by** (*induct j*) *simp-all*
qed
qed
have *2*: ($\forall i.\ v\ (s\ (Suc\ i)) = v\ (s\ i)) \Longrightarrow (\forall j.\ v\ (s\ j) = v\ (s\ 0)$)
using *1* **by** *blast*
have *3*: ($\forall j.\ v\ (s\ j) = v\ (s\ 0)) \Longrightarrow (\bigwedge j.\ v\ (s\ (Suc\ j)) = v\ (s\ j)$)
proof −
assume *A2*: ($\forall j.\ v\ (s\ j) = v\ (s\ 0)$)
show ($\bigwedge j.\ v\ (s\ (Suc\ j)) = v\ (s\ j)$)
proof −
fix *j*
show $v\ (s\ (Suc\ j)) = v\ (s\ j)$
using *A2*
proof (*induct j*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc j*)
then show *?case* **by** *metis*
qed
qed
qed

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

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

lemma *padded-defs* :
$(s \models padded\ v) =$
$(case\ s\ of\ (Inl\ s) \Rightarrow ((\forall\ i{<}\ intlen\ s.\ (v\ (nth\ s\ i)) = (v\ (nth\ s\ 0))) \vee intlen\ s = 0)$
$\mid (Inr\ s) \Rightarrow ((\forall\ i.\ (v\ (\ s\ i)) = (v\ (\ s\ 0)))\ )$
$)$
proof
(*simp add*: *padded-d-def stable-defs chop-d-def skip-defs empty-defs sum.case-eq-if*)
show $isl\ s \longrightarrow$
$((\exists\ n{\le}intlen\ (projl\ s).$
$(\forall i.\ i \le n \wedge i \le intlen\ (projl\ s) \longrightarrow$
$v\ (nth\ (projl\ s)\ i) = v\ (nth\ (projl\ s)\ 0)) \wedge intlen\ (projl\ s) - n = Suc\ 0) \vee$
$intlen\ (projl\ s) = 0) =$
$((\forall i{<}intlen\ (projl\ s).\ v\ (nth\ (projl\ s)\ i) = v\ (nth\ (projl\ s)\ 0)) \vee intlen\ (projl\ s) = 0)$
proof *rule+*
show $\bigwedge i.\ isl\ s \Longrightarrow$
$(\exists\ n{\le}intlen\ (projl\ s).$
$(\forall i.\ i \le n \wedge i \le intlen\ (projl\ s) \longrightarrow v\ (nth\ (projl\ s)\ i) = v\ (nth\ (projl\ s)\ 0))$
$\wedge\ intlen\ (projl\ s) - n = Suc\ 0) \vee$
$intlen\ (projl\ s) = 0 \Longrightarrow$
$i < intlen\ (projl\ s) \Longrightarrow v\ (Interval.nth\ (projl\ s)\ i) = v\ (Interval.nth\ (projl\ s)\ 0)$
by (*metis One-nat-def Suc-leI Suc-le-mono le-add-diff-inverse2 less-imp-le-nat not-less-zero plus-1-eq-Suc*)
show $isl\ s \Longrightarrow$
$(\forall i{<}intlen\ (projl\ s).$
$v\ (nth\ (projl\ s)\ i) = v\ (nth\ (projl\ s)\ 0)) \vee intlen\ (projl\ s) = 0 \Longrightarrow$
$(\exists\ n{\le}intlen\ (projl\ s).$
$(\forall i.\ i \le n \wedge i \le intlen\ (projl\ s) \longrightarrow$
$v\ (nth\ (projl\ s)\ i) = v\ (nth\ (projl\ s)\ 0)) \wedge intlen\ (projl\ s) - n = Suc\ 0) \vee$
$intlen\ (projl\ s) = 0$
by (*metis Suc-leI Suc-pred diff-diff-cancel diff-le-self gr-zeroI le-imp-less-Suc*)
qed
qed

lemma *padded-temporal-assign-defs* :
$(s \models v <\sim e) =$
$((s \models padded\ v) \wedge$

(*case s of* (*Inl s*) ⇒ (*v* (*Interval.nth s* (*intlen s*))) = *e* (*Inl s*)
| (*Inr s*) ⇒ *True*)
)
by (*auto simp add*: *padded-temp-assign-d-def padded-defs temporal-assign-defs*)

11.5 Soundness Axioms

11.5.1 ChopAssoc

lemma *ChopAssocSemHelpa*:
assumes (∃ *i ia* . *i*≤*intlen σ* ∧ *ia* ≤ *intlen σ* −*i* ∧ (*Inl* (*prefix i σ*) ⊨ *f*) ∧
(*Inl* (*prefix ia* (*suffix i σ*)) ⊨ *g*) ∧ (*Inl* (*suffix* (*ia* + *i*) *σ*) ⊨ *h*))
shows (∃ *j ja* . *j* ≤*intlen σ* ∧ *ja* ≤ *j* ∧ (*Inl* (*prefix ja* (*prefix j σ*)) ⊨ *f*) ∧
(*Inl* (*suffix ja* (*prefix j σ*)) ⊨ *g*) ∧ (*Inl* (*suffix j σ*) ⊨ *h*))
proof −
have *1*: (∃ *i ia* . *i*≤*intlen σ* ∧ *ia* ≤ *intlen σ* −*i* ∧ (*Inl* (*prefix i σ*) ⊨ *f*) ∧
(*Inl* (*prefix ia* (*suffix i σ*)) ⊨ *g*) ∧ (*Inl* (*suffix* (*ia* + *i*) *σ*) ⊨ *h*))
using *assms* **by** *auto*
obtain *i ia* **where** *2*: *i*≤*intlen σ* ∧ *ia* ≤ *intlen σ* −*i* ∧ (*Inl* (*prefix i σ*) ⊨ *f*) ∧
(*Inl* (*prefix ia* (*suffix i σ*)) ⊨ *g*) ∧ (*Inl* (*suffix* (*ia* + *i*) *σ*) ⊨ *h*)
using *1* **by** *auto*
have *3*: (*Inl* (*suffix* (*ia*+*i*) *σ*) ⊨ *h*)
using *2* **by** *auto*
have *4*: *ia* +*i* ≤*intlen σ*
using *2 Nat.le-diff-conv2* **by** *blast*
have *5*: *i* ≤ *ia*+*i*
by *simp*
have *6*: (*Inl* (*suffix i* (*prefix* (*ia* +*i*) *σ*)) ⊨ *g*)
using *2 4 interval-suffix-prefix-swap* **by** *force*
have *7*: (*Inl* (*prefix i* (*prefix* (*ia* +*i*) *σ*)) ⊨ *f*)
by (*simp add*: *2 add.commute*)
show *?thesis* **using** *2 4 5 6 7* **by** *blast*
qed

lemma *ChopAssocSemHelpb*:
assumes (∃ *j ja* . *j* ≤*intlen σ* ∧ *ja* ≤ *j* ∧ (*Inl* (*prefix ja* (*prefix j σ*)) ⊨ *f*) ∧
(*Inl* (*suffix ja* (*prefix j σ*)) ⊨ *g*) ∧ (*Inl* (*suffix j σ*) ⊨ *h*))
shows (∃ *i ia* . *i*≤*intlen σ* ∧ *ia* ≤ *intlen σ* −*i* ∧ (*Inl* (*prefix i σ*) ⊨ *f*) ∧
(*Inl* (*prefix ia* (*suffix i σ*)) ⊨ *g*) ∧ (*Inl* (*suffix* (*ia* + *i*) *σ*) ⊨ *h*))
proof −
have *1*: (∃ *j ja* . *j* ≤*intlen σ* ∧ *ja* ≤ *j* ∧ (*Inl* (*prefix ja* (*prefix j σ*)) ⊨ *f*) ∧
(*Inl* (*suffix ja* (*prefix j σ*)) ⊨ *g*) ∧ (*Inl* (*suffix j σ*) ⊨ *h*))
using *assms* **by** *auto*
obtain *j ja* **where** *2*: *j* ≤*intlen σ* ∧ *ja* ≤ *j* ∧ (*Inl* (*prefix ja* (*prefix j σ*)) ⊨ *f*) ∧
(*Inl* (*suffix ja* (*prefix j σ*)) ⊨ *g*) ∧ (*Inl* (*suffix j σ*) ⊨ *h*)
using *1* **by** *auto*
have *3*: *ja*≤*intlen σ*
using *2 le-trans* **by** *blast*
have *4*: *j* −*ja* ≤ *intlen σ* −*ja*
by (*simp add*: *2 diff-le-mono*)
have *5*: (*Inl* (*prefix ja σ*) ⊨ *f*)

by (*metis 2 interval-pref-pref-3 le-add-diff-inverse*)
have *6*: (*Inl* (*prefix* (j −*ja*) (*suffix ja* σ)) $\models$ *g*)
by (*simp add*: *2 interval-suffix-prefix-swap*)
have *7*: (*Inl* (*suffix* ((j −*ja*) + *ja*) σ) $\models$ *h*)
by (*simp add*: *2*)
show *?thesis* **using** *3 4 5 6 7* **by** *blast*
qed

lemma *ChopAssocSemHelp*:
($\exists$ *i ia* . *i*$\leq$*intlen* σ $\wedge$ *ia* $\leq$ *intlen* σ −*i* $\wedge$ (*Inl* (*prefix i* σ) $\models$ *f*) $\wedge$
(*Inl* (*prefix ia* (*suffix i* σ)) $\models$ *g*) $\wedge$ (*Inl* (*suffix* (*ia* + *i*) σ) $\models$ *h*)) =
($\exists$ *j ja* . *j* $\leq$*intlen* σ $\wedge$ *ja* $\leq$ *j* $\wedge$ (*Inl* (*prefix ja* (*prefix j* σ)) $\models$ *f*) $\wedge$
(*Inl* (*suffix ja* (*prefix j* σ)) $\models$ *g*) $\wedge$ (*Inl* (*suffix j* σ) $\models$ *h*))
using *ChopAssocSemHelpa*[*of* σ *f g h*]
ChopAssocSemHelpb[*of* σ *f g h*] **by** *auto*

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

lemma *ChopAssocSemHelpInFinite*:
((*Inr* σ) $\models$ *f* ; (*g* ; *h*)) = ((*Inr* σ) $\models$ (*f*;*g*);*h*)
proof −
have ((*Inr* σ) $\models$ *f* ; (*g* ; *h*)) =
(($\exists$ *n*.
f (*Inl* (*iprefix n* σ)) $\wedge$
(($\exists$ *na*. *g* (*Inl* (*iprefix na* (*isuffix n* σ))) $\wedge$ *h* (*Inr* (*isuffix na* (*isuffix n* σ)))) $\vee$
g (*Inr* (*isuffix n* σ)))) $\vee$

f $(Inr\ \sigma))$
by (*metis chop-defs-infinite*)
also have ... =
$((\exists\, n\ na.$
$f\ (Inl\ (iprefix\ n\ \sigma)) \wedge$
$((\ g\ (Inl\ (iprefix\ na\ (isuffix\ n\ \sigma))) \wedge h\ (Inr\ (isuffix\ na\ (isuffix\ n\ \sigma)))) \vee$
$g\ (Inr\ (isuffix\ n\ \sigma)))) \vee$
$f\ (Inr\ \sigma))$
by *fastforce*
also have ... =
$((\exists\, n\ na.\ na \le intlen\ (iprefix\ n\ \sigma) \wedge$
$(\ f\ (Inl\ (prefix\ na\ (iprefix\ n\ \sigma))) \wedge g\ (Inl\ (suffix\ na\ (iprefix\ n\ \sigma)))) \wedge$
$h\ (Inr\ (isuffix\ n\ \sigma))) \vee$
$(\exists\, n{::}nat.\ f\ (Inl\ (iprefix\ n\ \sigma)) \wedge g\ (Inr\ (isuffix\ n\ \sigma))) \vee f\ (Inr\ \sigma))$

by (*auto simp add*: *add.commute interval-iprefix-isuffix-swap iprefix-length isuffix-isuffix*)
(*metis interval-pref-ipref-3 ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
also have ... =
$((Inr\ \sigma) \models (f;g);h)$
by (*metis chop-defs-finite chop-defs-infinite le-add2 le-add-same-cancel2*)
finally show $((Inr\ \sigma) \models\ f\ ;\ (g\ ;\ h)) = ((Inr\ \sigma) \models (f;g);h)$ **.**
qed

lemma *ChopAssocSem*:
$(\sigma \models\ f\ ;\ (g\ ;\ h) =\ (f;g);h)$
by *auto*
(*metis ChopAssocSemHelpFinite ChopAssocSemHelpInFinite sum.collapse*(*1*) *sum.collapse*(*2*))+

11.5.2 OrChopImp

lemma *OrChopImpSem*:
$(\sigma \models\ (\ f \vee g);h\ \ \longrightarrow\ f;h \vee g;h\ )$
by (*auto simp add*: *chop-defs sum.case-eq-if*)

11.5.3 ChopOrImp

lemma *ChopOrImpSem*:
$(\sigma \models f;(g \vee h) \longrightarrow\ \ f;g \vee\ f;h\ )$
by (*auto simp add*: *chop-defs sum.case-eq-if*)

11.5.4 EmptyChop

lemma *EmptyChopSemFinite*:
$((Inl\ \sigma) \models empty\ ;\ f = f\ )$
using *min.absorb1* **by** (*simp add*: *empty-defs chop-defs*) *force*

lemma *EmptyChopSemInfinite*:
$((Inr\ \sigma) \models empty\ ;\ f = f\ )$
by (*simp add*: *chop-defs empty-defs iprefix-length isuffix-0*)

lemma *EmptyChopSem*:

$(\sigma \models empty\ ;\ f = f\ )$
using *EmptyChopSemFinite EmptyChopSemInfinite*
by (*metis sum.collapse*(*1*) *sum.collapse*(*2*))

11.5.5 ChopEmpty

lemma *ChopEmptySemFinite*:
$((Inl\ \sigma) \models f;empty = f\ )$
by (*simp add*: *empty-defs chop-defs*) *auto*

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

lemma *ChopEmptySem*:
$(\sigma \models f;empty = f\ )$
using *ChopEmptySemFinite ChopEmptySemInfinite*
by (*metis sum.collapse*(*1*) *sum.collapse*(*2*))

11.5.6 StateImpBi

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

11.5.7 NextImpNotNextNot

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

11.5.8 BiBoxChopImpChop

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

11.5.9 BoxInduct

lemma *box-induct-help-1* :
$((Inl\ \sigma) \models f) \wedge (\forall i.\ Suc\ 0 \leq intlen\ \sigma - i \longrightarrow$
$i \leq intlen\ \sigma \longrightarrow (Inl\ (suffix\ i\ \sigma) \models f) \longrightarrow (Inl\ (suffix\ (Suc\ i)\ \sigma) \models f))$
$\Longrightarrow (\forall\ j.\ j \leq intlen\ \sigma \longrightarrow (Inl\ (suffix\ j\ \sigma) \models f))$
proof
fix j
show $((Inl\ \sigma) \models f) \wedge (\forall i.\ Suc\ 0 \leq intlen\ \sigma - i \longrightarrow$
$i \leq intlen\ \sigma \longrightarrow (Inl\ (suffix\ i\ \sigma) \models f) \longrightarrow (Inl\ (suffix\ (Suc\ i)\ \sigma) \models f))$
$\Longrightarrow\ j \leq intlen\ \sigma \longrightarrow (Inl\ (suffix\ j\ \sigma) \models f)$
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 *box-induct-help-infinite* :
$((Inr\ \sigma) \models f) \wedge (\forall\, i.$
$((Inr\ (isuffix\ i\ \sigma) \models f) \longrightarrow (Inr\ (isuffix\ (Suc\ i)\ \sigma) \models f)))$
$\Longrightarrow (\forall\ j.\ \ (Inr\ (isuffix\ j\ \sigma) \models f))$
proof
fix *j*
show $((Inr\ \sigma) \models f) \wedge (\forall\, i.$
$(Inr\ (isuffix\ i\ \sigma) \models f) \longrightarrow (Inr\ (isuffix\ (Suc\ i)\ \sigma) \models f))$
$\Longrightarrow\ (Inr\ (isuffix\ j\ \sigma) \models f)$
proof (*induct j arbitrary*: σ)
case *0*
then show *?case* **by** (*simp add*: *isuffix-0*)
next
case (*Suc j*)
then show *?case* **by** *blast*
qed
qed

lemma *BoxInductSem*:
$(\sigma \models \Box\ (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box\ f)$
proof
(*auto simp add*: *always-defs wnext-defs sum.case-eq-if*)
show $\bigwedge n.\ isl\ \sigma \Longrightarrow$
$\forall\, n \le intlen\ (projl\ \sigma).\ f\ (Inl\ (suffix\ n\ (projl\ \sigma))) \longrightarrow$
$intlen\ (projl\ \sigma) = n \vee f\ (Inl\ (suffix\ (Suc\ n)\ (projl\ \sigma))) \Longrightarrow$
$f\ \sigma \Longrightarrow n \le intlen\ (projl\ \sigma) \Longrightarrow f\ (Inl\ (suffix\ n\ (projl\ \sigma)))$
by (*metis One-nat-def box-induct-help-1 cancel-comm-monoid-add-class.diff-cancel not-one-le-zero sum.collapse*(*1*))
show $\bigwedge n.\ \neg\ isl\ \sigma \Longrightarrow$
$\forall\, n.\ f\ (Inr\ (isuffix\ n\ (projr\ \sigma))) \longrightarrow f\ (Inr\ (isuffix\ (Suc\ 0)\ (isuffix\ n\ (projr\ \sigma)))) \Longrightarrow$
$f\ \sigma \Longrightarrow f\ (Inr\ (isuffix\ n\ (projr\ \sigma)))$
by (*metis* (*mono-tags*) *add.right-neutral add-Suc-right box-induct-help-infinite isuffix-isuffix sum.collapse*(*2*))
qed

11.5.10 ChopStarEqv

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

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

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

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

lemma *powersem1*:
($\sigma \models$ ($\exists$ *k*. *power f k*) = (*empty* $\vee$ ($\exists$ *k*. *power f* (*Suc k*))))
proof *auto*
show $\bigwedge$*x*. $\sigma \models$ (*power f x*) $\Longrightarrow$ $\forall$ *k*. $\neg$ ($\sigma \models$ (*f* $\wedge$ *finite*);*power f k*) $\Longrightarrow$ $\sigma \models$ *empty*
by (*metis not0-implies-Suc pow-0 pow-Suc*)
show $\sigma \models$ *empty* $\Longrightarrow$ $\exists$ *x*. $\sigma \models$ (*power f x*)
by (*metis pow-0*)
show $\bigwedge$*k*. $\sigma \models$ ((*f* $\wedge$ *finite*);*power f k*) $\Longrightarrow$ $\exists$ *x*. $\sigma \models$ (*power f x*)
by (*metis pow-Suc*)
qed

lemma *spowersem1*:
($\sigma \models$ ($\exists$ *k*. *spower f k*) = (*empty* $\vee$ ($\exists$ *k*. *spower f* (*Suc k*))))
proof *auto*
show $\bigwedge$*x*. $\sigma \models$ (*spower f x*) $\Longrightarrow$ $\forall$ *k*. $\neg$ ($\sigma \models$ *f* $\frown$*spower f k*) $\Longrightarrow$ $\sigma \models$ *empty*
by (*metis not0-implies-Suc spow-0 spow-Suc*)
show $\sigma \models$ *empty* $\Longrightarrow$ $\exists$ *x*. $\sigma \models$ (*spower f x*)
by (*metis spow-0*)
show $\bigwedge$*k*. $\sigma \models$ (*f* $\frown$ *spower f k*) $\Longrightarrow$ $\exists$ *x*. $\sigma \models$ (*spower f x*)
by (*metis spow-Suc*)
qed

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

lemma *spowersem*:
$\vdash$ ($\exists$ *k*. *spower f k*) = (*empty* $\vee$ *f*$\frown$($\exists$ *k*. (*spower f k*)))
proof −
have *1*: $\vdash$ ($\exists$ *k*. *spower f k*) = (*empty* $\vee$ ($\exists$ *k*. *spower f* (*Suc k*)))

using *spowersem1* **by** *blast*
have *2*: $\vdash$ ($\exists$ *k*. *spower f* (*Suc k*)) = ($\exists$ *k*. *f*$\frown$*spower f k*)
by *simp*
have *3*: $\vdash$ ($\exists$ *k*. *f*$\frown$(*spower f k*)) = *f*$\frown$($\exists$ *k*. (*spower f k*))
using *SChopExist* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *PowerstarEqvSemhelp1*:
$\vdash$ *empty*;(*empty* $\vee$ (*f* $\wedge$ *inf*)) = (*empty* $\vee$ (*f* $\wedge$ *inf*))
using *EmptyChopSem* **by** *blast*

lemma *PowerstarEqvSemhelp2*:
$\vdash$ (*f* $\wedge$ *inf*) = (*f* $\wedge$ *inf*);*g*
by (*simp add*: *Valid-def infinite-defs chop-defs sum.case-eq-if*)

lemma *PowerstarEqvSemhelp3*:
$\vdash$ ((*f* $\wedge$ *inf*);*g* $\vee$ (*f* $\wedge$ *finite*);*g*) = (*f* ;*g*)
by (*auto simp add*: *Valid-def finite-defs infinite-defs chop-defs sum.case-eq-if*)

lemma *PowerstarEqvSem*:
(σ $\models$ (*powerstar f*) = (*empty* $\vee$ *f*;(*powerstar f*)))
proof −
have *1*: (σ $\models$ (*powerstar f*)) =
(σ $\models$ ($\exists$ *k*. *power f k*);(*empty* $\vee$ *f* $\wedge$ *inf*))
by (*simp add*: *powerstar-d-def*)
have *2*: (σ $\models$ ($\exists$ *k*. *power f k*);(*empty* $\vee$ *f* $\wedge$ *inf*)) =
(σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*))
using *powersem* **by** (*metis inteq-reflection*)
have *3*: (σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*)) =
(σ $\models$ *empty*;(*empty* $\vee$ (*f* $\wedge$ *inf*)) $\vee$
((*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*))
by (*auto simp add*: *chop-defs sum.case-eq-if*)
have *4*: (σ $\models$ *empty*;(*empty* $\vee$ (*f* $\wedge$ *inf*)) $\vee$
((*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*)) =
(σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*)))
using *PowerstarEqvSemhelp1*
by (*metis* (*mono-tags*, *lifting*) *inteq-reflection unl-lift2*)
have *5*: (σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*))) =
(σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *inf*);(($\exists$ *k*. (*power f k*));(*empty* $\vee$ *f* $\wedge$ *inf*)) $\vee$
((*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*)))
using *PowerstarEqvSemhelp2*
by (*metis* (*mono-tags*, *lifting*) *inteq-reflection*)
have *6*: (σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *inf*);(($\exists$ *k*. (*power f k*));(*empty* $\vee$ *f* $\wedge$ *inf*)) $\vee$
((*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*))) =
(σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *inf*);(($\exists$ *k*. (*power f k*));(*empty* $\vee$ *f* $\wedge$ *inf*)) $\vee$
(*f* $\wedge$ *finite*);(($\exists$ *k*. (*power f k*));(*empty* $\vee$ *f* $\wedge$ *inf*))))

by *auto*
(*metis ChopAssocSemHelpFinite ChopAssocSemHelpInFinite sum.split-sel*)+

have *7* : ($\sigma \models$ (*empty* $\vee$ (*f* $\wedge$ *inf*);(($\exists$ *k*. (*power f k*));(*empty* $\vee$ *f* $\wedge$ *inf*)) $\vee$
(*f* $\wedge$ *finite*);(($\exists$ *k*. (*power f k*));(*empty* $\vee$ *f* $\wedge$ *inf*)))) =
($\sigma \models$ (*empty* $\vee$ *f*;(($\exists$ *k*. (*power f k*));(*empty* $\vee$ *f* $\wedge$ *inf*))))
using *PowerstarEqvSemhelp3* **by** *fastforce*
have *8*: ($\sigma \models$ (*empty* $\vee$ *f*;(($\exists$ *k*. (*power f k*));(*empty* $\vee$ *f* $\wedge$ *inf*)))) =
($\sigma \models$ (*empty* $\vee$ *f*;(*powerstar f*)))
by (*simp add*: *powerstar-d-def*)
from *1 2 3 4 5 6 7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *FPowerstarEqvSem*:
($\sigma \models$ (*fpowerstar f*) = (*empty* $\vee$ (*f* $\wedge$ *finite*);(*fpowerstar f*)))
proof −
have *1*: ($\sigma \models$ (*fpowerstar f*)) =
($\sigma \models$ ($\exists$ *k*. *power f k*))
by (*simp add*: *fpowerstar-d-def*)
have *2*: ($\sigma \models$ ($\exists$ *k*. *power f k*)) =
($\sigma \models$ (*empty* $\vee$ (*f* $\wedge$ *finite*);($\exists$ *k*. (*power f k*))))
using *powersem* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** (*simp add*: *fpowerstar-d-def*)
qed

lemma *SPowerstarEqvSem*:
($\sigma \models$ (*spowerstar f*) = (*empty* $\vee$ *f*$\frown$(*spowerstar f*)))
proof −
have *1*: ($\sigma \models$ (*spowerstar f*)) =
($\sigma \models$ ($\exists$ *k*. *spower f k*))
by (*simp add*: *spowerstar-d-def*)
have *2*: ($\sigma \models$ ($\exists$ *k*. *spower f k*)) =
($\sigma \models$ (*empty* $\vee$ *f*$\frown$($\exists$ *k*. (*spower f k*))))
using *spowersem* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** (*simp add*: *spowerstar-d-def*)
qed

lemma *powerchopsem*:
$\vdash$ ($\exists$ *k*. *power* (*f* $\wedge$ *more*) *k*) =
(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);($\exists$ *k*. (*power* (*f* $\wedge$ *more*) *k*))
)
using *powersem* **by** *auto*

lemma *spowerchopsem*:
$\vdash$ ($\exists$ *k*. *spower* (*f* $\wedge$ *more*) *k*) =
(*empty* $\vee$ (*f* $\wedge$ *more*)$\frown$($\exists$ *k*. (*spower* (*f* $\wedge$ *more*) *k*))
)
using *spowersem* **by** *auto*

lemma *ChopstarEqvSem*:
($\sigma \models$ $f^\star$ = (*empty* $\vee$ (*f* $\wedge$ *more*); $f^\star$))
by (*metis PowerstarEqvSem chopstar-d-def*)

lemma *SChopstarEqvSem*:
  $(\sigma \models (schopstar\ f) = (empty \lor (f \land more) \frown (schopstar\ f)))$
by (*metis SPowerstarEqvSem schopstar-d-def*)

11.5.11 OmegaUnroll

lemma *omega-unroll-chain*:
$(\exists\, l.\ infinite\text{-}index\text{-}sequence\ 0\ l \land (\forall\, i.\ f\ (Inl\ (subinterval\ \sigma\ (l\ i)\ (l\ (Suc\ i))))))$
$=$
$(\exists\, n.$
  $f\ (Inl\ (iprefix\ n\ \sigma)) \land$
  $0 < n \land$
  $(\exists\, l.$
    $infinite\text{-}index\text{-}sequence\ 0\ l \land$
    $(\forall\, i.\ f\ (Inl\ (subinterval\ (isuffix\ n\ \sigma)\ (l\ i)\ (l\ (Suc\ i)))))$
  $)$
$)$

proof −
have $(\exists\, l.\ infinite\text{-}index\text{-}sequence\ 0\ l \land$
  $(\forall\, i.\ f\ (Inl\ (subinterval\ \sigma\ (l\ i)\ (l\ (Suc\ i)))))) =$
  $(\exists\, l.\ infinite\text{-}index\text{-}sequence\ 0\ l \land l = conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1))) \land$
  $(\forall\, i.\ f\ (Inl\ (subinterval\ \sigma\ (l\ i)\ (l\ (Suc\ i))))))$
  using *iidx-1* **by** *blast*
also have ... =
  $(\exists\, l.\ infinite\text{-}index\text{-}sequence\ 0\ (conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1)))) \land$
    $l = conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1))) \land$
    $(\forall\, i.\ f\ (Inl\ (subinterval\ \sigma\ (conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1)))\ i)$
      $(conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1)))\ (Suc\ i))))$
    $)$
  $)$
  by *force*
also have ... =
  $(\exists\, l.\ (l\ 0) = 0 \land (l\ 0) < (l\ 1) \land$
    $infinite\text{-}index\text{-}sequence\ (l\ 1)\ (\lambda x.\ (l\ (x{+}1))) \land$
    $l = conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1))) \land$
    $(\forall\, i.\ f\ (Inl\ (subinterval\ \sigma\ (conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1)))\ i)$
      $(conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1)))\ (Suc\ i))))$
    $)$
  $)$

  using *iidx-2* **by** *force*
also have ... =
  $(\exists\, l.\ (l\ 0) = 0 \land (l\ 0) < (l\ 1) \land$
    $infinite\text{-}index\text{-}sequence\ (l\ 1)\ (\lambda x.\ (l\ (x{+}1))) \land$
    $l = conc\ \langle(l\ 0)\rangle\ (\lambda x.\ (l\ (x{+}1))) \land$
    $f\ (Inl\ (subinterval\ \sigma\ (l\ 0)\ (l\ 1))) \land$
    $(\forall\, i.\ f\ (Inl\ (subinterval\ \sigma\ ((\lambda x.\ (l\ (x{+}1)))\ i)\ ((\lambda x.\ (l\ (x{+}1)))\ (Suc\ i)))))$
  $)$

(**is** *?L=?R*)
proof *rule*
show *?L* $\Longrightarrow$ *?R*
by (*metis One-nat-def Suc-eq-plus1*)
show *?R* $\Longrightarrow$ *?L*
by (*metis* (*lifting*) *One-nat-def conc-empty-suc not0-implies-Suc*)
qed
also have ... =
$(\exists\, l\ ls.\ \ ls = (\lambda x.\ (l\ (x{+}1))) \wedge (l\ 0) = 0 \wedge (l\ 0) < (l\ 1) \wedge$
$\textit{infinite-index-sequence}\ (l\ 1)\ ls\ \wedge$
$l = \textit{conc}\ \langle (l\ 0)\rangle\ ls\ \wedge$
$f\ (\textit{Inl}\ (\textit{subinterval}\ \sigma\ (l\ 0)\ (l\ 1)))\ \wedge$
$(\forall\, i.\ f\ (\textit{Inl}\ (\textit{subinterval}\ \sigma\ \ (ls\ i)\ (ls\ (\textit{Suc}\ i)))))$
)

by *force*
also have ... =
$(\exists\, l\ ls.\ \ ls = (\lambda x.\ (l\ (x{+}1))) \wedge (l\ 0) = 0 \wedge (l\ 0) < (l\ 1) \wedge$
$\textit{infinite-index-sequence}\ (l\ 1)\ ls\ \wedge$
$l = \textit{conc}\ \langle (l\ 0)\rangle\ ls\ \wedge$
$f\ (\textit{Inl}\ (\textit{iprefix}\ (l\ 1)\ \sigma))\ \wedge$
$(\forall\, i.\ f\ (\textit{Inl}\ (\textit{subinterval}\ \sigma\ \ (ls\ i)\ (ls\ (\textit{Suc}\ i)))))$
)

by (*metis iprefix-def*)
also have ... =
$(\exists\, l\ ls\ n.\ n = (ls\ 0) \wedge ls = (\lambda x.\ (l\ (x{+}1))) \wedge 0 < n\ \wedge$
$\textit{infinite-index-sequence}\ n\ ls \wedge l = \textit{conc}\ \langle 0\rangle\ ls\ \wedge$
$f\ (\textit{Inl}\ (\textit{iprefix}\ n\ \sigma))\ \wedge$
$(\forall\, i.\ f\ (\textit{Inl}\ (\textit{subinterval}\ \sigma\ \ (ls\ i)\ (ls\ (\textit{Suc}\ i)))))$
)

by (*metis* (*no-types*, *lifting*) *One-nat-def conc-empty-suc conc-empty-zero*)
also have ... =
$(\exists\ \ ls\ n.\ n = (ls\ 0) \wedge 0 < n\ \wedge$
$\textit{infinite-index-sequence}\ n\ ls\ \wedge$
$f\ (\textit{Inl}\ (\textit{iprefix}\ n\ \sigma))\ \wedge$
$(\forall\, i.\ f\ (\textit{Inl}\ (\textit{subinterval}\ \sigma\ \ (ls\ i)\ (ls\ (\textit{Suc}\ i)))))$
)

using *iidx-0* **by** *rule* (*metis* (*no-types*, *lifting*) *Suc-eq-plus1 conc-empty-suc*)
also have ... =
$(\exists\ \ ls\ n\ lsk.\ \ n = (ls\ 0) \wedge 0 < n \wedge lsk = (\textit{shiftm}\ n) \circ ls\ \wedge$
$\textit{infinite-index-sequence}\ n\ ls\ \wedge$
$f\ (\textit{Inl}\ (\textit{iprefix}\ n\ \sigma))\ \wedge$
$(\forall\, i.\ f\ (\textit{Inl}\ (\textit{subinterval}\ \sigma\ \ (ls\ i)\ (ls\ (\textit{Suc}\ i)))))$
)
by *blast*
also have ... =
$(\exists\ \ ls\ n\ lsk.\ \ n = (ls\ 0) \wedge 0 < n \wedge lsk = (\textit{shiftm}\ n) \circ ls\ \wedge$

$\quad\quad$ *infinite-index-sequence n ls* $\wedge$ *infinite-index-sequence 0 lsk* $\wedge$
$\quad\quad$ f (*Inl* (*iprefix n* σ)) $\wedge$
$\quad\quad$ ($\forall$ *i*. f (*Inl* (*subinterval* σ (*ls i*) (*ls* (*Suc i*)))))
)
using *iidx-5* **by** *auto*
also have ... =
($\exists$ *ls n lsk*. n = (*ls 0*) $\wedge$ $0 < n$ $\wedge$ *ls* = (*shift n*) $\circ$ *lsk* $\wedge$
$\quad\quad$ *infinite-index-sequence n ls* $\wedge$ *infinite-index-sequence 0 lsk* $\wedge$
$\quad\quad$ f (*Inl* (*iprefix n* σ)) $\wedge$
$\quad\quad$ ($\forall$ *i*. f (*Inl* (*subinterval* σ (*ls i*) (*ls* (*Suc i*)))))
)
using *iidx-6* **by** *blast*
also have ... =
($\exists$ *ls n lsk*. n = (((*shift n*) $\circ$ *lsk*) *0*) $\wedge$ $0 < n$ $\wedge$ *ls* = (*shift n*) $\circ$ *lsk* $\wedge$
$\quad\quad$ *infinite-index-sequence n ls* $\wedge$ *infinite-index-sequence 0 lsk* $\wedge$
$\quad\quad$ f (*Inl* (*iprefix n* σ)) $\wedge$
$\quad\quad$ ($\forall$ *i*. f (*Inl* (*subinterval* σ (((*shift n*) $\circ$ *lsk*) *i*)
$\quad\quad\quad\quad$ (((*shift n*) $\circ$ *lsk*) (*Suc i*))))
$\quad\quad$)
)
by *metis*
also have ... =
($\exists$ *ls n lsk*. *0* = (*lsk 0*) $\wedge$ $0 < n$ $\wedge$ *ls* = (*shift n*) $\circ$ *lsk* $\wedge$
$\quad\quad$ *infinite-index-sequence n ls* $\wedge$ *infinite-index-sequence 0 lsk* $\wedge$
$\quad\quad$ f (*Inl* (*iprefix n* σ)) $\wedge$
$\quad\quad$ ($\forall$ *i*. f (*Inl* (*subinterval* σ ((*lsk i*)+*n*) ((*lsk* (*Suc i*)+*n*)))))
)
using *shift-def* **by** *auto*
also have ... =
($\exists$ *ls n lsk*. *0* = (*lsk 0*) $\wedge$ $0 < n$ $\wedge$ *ls* = (*shift n*) $\circ$ *lsk* $\wedge$
$\quad\quad$ *infinite-index-sequence 0 lsk* $\wedge$
$\quad\quad$ f (*Inl* (*iprefix n* σ)) $\wedge$
$\quad\quad$ ($\forall$ *i*. f (*Inl* (*subinterval* σ ((*lsk i*)+*n*) ((*lsk* (*Suc i*)+*n*)))))
)
using *iidx-8* **by** *blast*
also have ... =
($\exists$ *n lsk*. *0* = (*lsk 0*) $\wedge$ $0 < n$ $\wedge$
$\quad\quad$ *infinite-index-sequence 0 lsk* $\wedge$
$\quad\quad$ f (*Inl* (*iprefix n* σ)) $\wedge$
$\quad\quad$ ($\forall$ *i*. f (*Inl* (*subinterval* σ ((*lsk i*)+*n*) ((*lsk* (*Suc i*)+*n*)))))
)
by *blast*

also have ... =
($\exists$ *n lsk*. *0* = (*lsk 0*) $\wedge$ $0 < n$ $\wedge$
$\quad\quad$ *infinite-index-sequence 0 lsk* $\wedge$
$\quad\quad$ f (*Inl* (*iprefix n* σ)) $\wedge$
$\quad\quad$ ($\forall$ *i*. f (*Inl* (*subinterval* (*isuffix n* σ) ((*lsk i*)) ((*lsk* (*Suc i*))))))
)
using *subinterval-sub-isuffix-iidx* **by** (*metis calculation calculation*)

also have ... =
(∃ *n*. *f* (*Inl* (*iprefix n σ*)) ∧ *0* < *n* ∧
(∃ *l*. *infinite-index-sequence 0 l* ∧
(∀ *i*. *f* (*Inl* (*subinterval* (*isuffix n σ*) (*l i*) (*l* (*Suc i*)))))
)
)
using *infinite-index-sequence-def* **by** *auto*
finally show (∃ *l*. *infinite-index-sequence 0 l* ∧
(∀ *i*. *f* (*Inl* (*subinterval σ* (*l i*) (*l* (*Suc i*)))))
)
=
(∃ *n*. *f* (*Inl* (*iprefix n σ*)) ∧ *0* < *n* ∧
(∃ *l*. *infinite-index-sequence 0 l* ∧
(∀ *i*. *f* (*Inl* (*subinterval* (*isuffix n σ*) (*l i*) (*l* (*Suc i*)))))
)
) **.**
qed

lemma *omega-unroll-sem*:
((*Inr σ*) ⊨ (*f* ∧ *fmore*);(*omega f*) = (*omega f*))
proof
(*simp add*: *fmore-defs chop-defs omega-d-def iprefix-length*)
show (∃ *n*. *f* (*Inl* (*iprefix n σ*)) ∧
0 < *n* ∧ (∃ *l*. *infinite-index-sequence 0 l* ∧
(∀ *i*. *f* (*Inl* (*subinterval* (*isuffix n σ*) (*l i*) (*l* (*Suc i*))))))) =
(∃ *l*. *infinite-index-sequence 0 l* ∧ (∀ *i*. *f* (*Inl* (*subinterval σ* (*l i*) (*l* (*Suc i*))))))
using *omega-unroll-chain* **by** *metis*
qed

lemma *OmegaUnrollSem*:
σ ⊨ (*omega f*) = (*f* ∧ *fmore*);(*omega f*)
proof (*cases σ*)
case (*Inl a*)
then show *?thesis* **by** (*simp add*: *omega-d-def chop-defs-finite*)
next
case (*Inr b*)
then show *?thesis* **using** *omega-unroll-sem* **by** *fastforce*
qed

11.5.12 OmegaInduct

lemma *OmegaInductSem-help*:
(*σ* ⊨ *inf* ∧ *g* ∧ □(*g* ⟶ (*f* ∧ *fmore*);*g*)) =
(*case σ of* (*Inl σ*) ⇒ *False*
| (*Inr σ*) ⇒
g (*Inr σ*) ∧
(∀ *n*::*nat*. *g* (*Inr* (*isuffix n σ*)) ⟶
(∃ *na*::*nat*. *f* (*Inl* (*subinterval σ n* (*na*+*n*))) ∧
(*0*::*nat*) < *na* ∧ *g* (*Inr* (*isuffix* (*n* + *na*) *σ*))))

)

by (*simp add*: *infinite-defs always-defs chop-defs fmore-defs isuffix-isuffix sum.case-eq-if*)
(*metis iprefix-isuffix iprefix-length*)

lemma *OmegaInductSem-help-infinite*:
$((Inr\ \sigma) \models\ inf \wedge g \wedge \Box(g \longrightarrow (f \wedge fmore);g)) =$
$(g\ (Inr\ \sigma) \wedge$
$(\forall\, n::nat.\ g\ (Inr\ (isuffix\ n\ \sigma)) \longrightarrow$
$(\exists\, na::nat.\ f\ (Inl\ (subinterval\ \sigma\ n\ (na+n))) \wedge$
$(0::nat) < na \wedge g\ (Inr\ (isuffix\ (na+n)\ \sigma)))))$

by (*simp add*: *infinite-defs always-defs chop-defs fmore-defs isuffix-isuffix sum.case-eq-if*)
(*metis iprefix-isuffix iprefix-length add.commute*)

primrec *cpoint* :: $('a::world)\ formula\ \Rightarrow 'a\ formula \Rightarrow nat \Rightarrow 'a\ infinterval \Rightarrow nat$
where $cpoint\ f\ g\ 0\ \sigma = 0$
$|\ cpoint\ f\ g\ (Suc\ n)\ \sigma =$
$(\epsilon\ x.\ (\exists\ m.\ (Inl\ (subinterval\ \sigma\ (cpoint\ f\ g\ n\ \sigma)\ (m+(cpoint\ f\ g\ n\ \sigma))\ ) \models f)$
$\wedge\ m>0\ \wedge(Inr(isuffix\ (m+(cpoint\ f\ g\ n\ \sigma))\ \sigma) \models g) \wedge$
$x=m+(cpoint\ f\ g\ n\ \sigma)$
)
)

lemma *cpoint-order-0a*:
$na\le x \Longrightarrow\ intlen\ (suffix\ na\ (iprefix\ x\ \sigma)) = x-na$
by (*simp add*: *iprefix-length*)

lemma *cpoint-expand-0*:
$(cpoint\ f\ g\ 0\ \sigma) = 0$
by *simp*

lemma *cpoint-expand-1*:
$(cpoint\ f\ g\ 1\ \sigma) =$
$(SOME\ x.\ (\exists\ m.\ f\ (Inl\ (subinterval\ \sigma\ 0\ (m)\ ))$
$\wedge\ m>0 \wedge g\ (Inr\ (isuffix\ (m)\ \sigma))$
$\wedge\ x=m))$
by (*simp add*: *fmore-defs subinterval-length*)

lemma *cpoint-expand-n*:
$(cpoint\ f\ g\ (Suc\ n)\ \sigma) =$
$(SOME\ x.\ (\exists\ m.\ f\ (Inl\ (subinterval\ \sigma\ (cpoint\ f\ g\ n\ \sigma)\ (m+(cpoint\ f\ g\ n\ \sigma))))$
$\wedge\ m>0 \wedge g\ (Inr(isuffix\ (m+(cpoint\ f\ g\ n\ \sigma))\ \sigma))$
$\wedge\ x=m+(cpoint\ f\ g\ n\ \sigma))$
)
by (*simp add*: *fmore-defs subinterval-length*)

lemma *cpoint-0*:

```
 assumes g (Inr σ) ∧
          (∀ k. g (Inr (isuffix k σ)) ⟶
               (∃ m. f (Inl (subinterval σ k (m+k))) ∧
                  0 < m ∧ g (Inr (isuffix (m+k) σ))))

 shows  g (Inr(isuffix (cpoint f g i σ) σ))
proof
 (induct i)
 case 0
 then show ?case by (simp add: assms isuffix-0)
 next
 case (Suc i)
 then show ?case
 proof −
  have 1: g (Inr(isuffix (cpoint f g i σ) σ))
  by (simp add: Suc.hyps)
  have 2: g (Inr(isuffix (cpoint f g i σ) σ)) ⟶
       (∃ m. f (Inl (subinterval σ (cpoint f g i σ) (m+(cpoint f g i σ)))) ∧
                  0 < m ∧ g (Inr (isuffix (m+(cpoint f g i σ)) σ)))
  using assms by blast
  have 3: (∃ m. f (Inl (subinterval σ (cpoint f g i σ) (m+(cpoint f g i σ)))) ∧
                  0 < m ∧ g (Inr (isuffix (m+(cpoint f g i σ)) σ)))
  using 1 2 by auto
  have 4: (cpoint f g (Suc i) σ) =
       (SOME x. (∃ m. f (Inl (subinterval σ (cpoint f g i σ) (m+(cpoint f g i σ))))
            ∧ m>0 ∧ g (Inr(isuffix (m+(cpoint f g i σ)) σ))
            ∧ x=m+(cpoint f g i σ)))
  by simp
  have 5: g (Inr(isuffix ((cpoint f g (Suc i) σ)) σ))
  using 3 4 someI-ex[of λx. (∃ m. f (Inl (subinterval σ (cpoint f g i σ) (m+(cpoint f g i σ))))
            ∧ m>0 ∧ g (Inr(isuffix (m+(cpoint f g i σ)) σ))
            ∧ x=m+(cpoint f g i σ))] by auto
  from 5 show ?thesis by auto
  qed
 qed


lemma cpoint-1:
 assumes g (Inr σ) ∧
          (∀ k. g (Inr (isuffix k σ)) ⟶
               (∃ m. f (Inl (subinterval σ k (m+k))) ∧
                  0 < m ∧ g (Inr (isuffix (m+k) σ))))

 shows  ( g (Inr(isuffix (cpoint f g i σ) σ))
         ⟹ g (Inr(isuffix (cpoint f g (Suc i) σ) σ)))

proof −
 have 1: g (Inr σ) ∧
          (∀ k. g (Inr (isuffix k σ)) ⟶
```

$(\exists\, m.\ f\ (Inl\ (subinterval\ \sigma\ k\ (m{+}k))) \wedge$
$0 < m \wedge g\ (Inr\ (isuffix\ (m{+}k)\ \sigma))))$
using *assms* **by** *blast*
have *2*: $(\ g\ (Inr(isuffix\ (cpoint\ f\ g\ i\ \sigma)\ \sigma))$
$\Longrightarrow g\ (Inr(isuffix\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ \sigma)))$
proof
(*induct i*)
case *0*
then show *?case*
proof −
have *01*: $g\ (Inr(isuffix\ (cpoint\ f\ g\ 0\ \sigma)\ \sigma)) =$
$g\ (Inr(isuffix\ 0\ \sigma))$
by *auto*
have *02*: $g\ (Inr(isuffix\ 0\ \sigma)) = g\ (Inr\ \sigma)$
by (*simp add*: *isuffix-0*)
have *03*: $(\exists\, m.\ f\ (Inl\ (subinterval\ \sigma\ 0\ (m))) \wedge$
$0 < m \wedge g\ (Inr\ (isuffix\ (m)\ \sigma)))$
using *02 1* **by** *fastforce*
have *04*: $(cpoint\ f\ g\ 1\ \sigma) =$
$(SOME\ x.\ (\exists\ m.\ f\ (Inl\ (subinterval\ \sigma\ 0\ (m)\ ))$
$\wedge\ m{>}0 \wedge g\ (Inr\ (isuffix\ (m)\ \sigma))$
$\wedge\ x{=}m)$
$)$
using *cpoint-expand-1* **by** *blast*
have *05*: $g\ (Inr\ (isuffix\ (cpoint\ f\ g\ 1\ \sigma)\ \sigma))$
using *03 04 someI-ex* **by** (*metis* (*mono-tags*, *lifting*))
from *01 05* **show** *?thesis* **by** *auto*
qed
next
case (*Suc i*)
then show *?case*
proof −
have *n1*: $g\ (Inr\ (isuffix\ (cpoint\ f\ g\ i\ \sigma)\ \sigma)) \Longrightarrow g\ (Inr\ (isuffix\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ \sigma))$
using *Suc.hyps* **by** *blast*
have *n2*: $g\ (Inr\ (isuffix\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ \sigma))$
using *Suc.prems* **by** *blast*
have *n3*: $(\exists\, m.\ f\ (Inl\ (subinterval\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ (m{+}(cpoint\ f\ g\ (Suc\ i)\ \sigma)))) \wedge$
$0 < m \wedge g\ (Inr\ (isuffix\ (m{+}(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ \sigma)))$
using *assms n2* **by** *auto*
have *n4*: $(cpoint\ f\ g\ (Suc\ (Suc\ i))\ \sigma) =$
$(SOME\ x.\ (\exists\ m.\ f\ (Inl\ (subinterval\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ (m{+}(cpoint\ f\ g\ (Suc\ i)\ \sigma))))$
$\wedge\ m{>}0 \wedge g\ (Inr(isuffix\ (m{+}(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ \sigma))$
$\wedge\ x{=}m{+}(cpoint\ f\ g\ (Suc\ i)\ \sigma))$
$)$
using *cpoint-expand-n* **by** *blast*
have *n5*: $g\ (Inr\ (isuffix\ (cpoint\ f\ g\ (Suc\ (Suc\ i))\ \sigma)\ \sigma))$
using *n3 n4 someI-ex*[*of* $\lambda x.\ (\exists\ m.\ f\ (Inl\ (subinterval\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)$
$(m{+}(cpoint\ f\ g\ (Suc\ i)\ \sigma))))$
$\wedge\ m{>}0 \wedge g\ (Inr(isuffix\ (m{+}(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ \sigma))$
$\wedge\ x{=}m{+}(cpoint\ f\ g\ (Suc\ i)\ \sigma))$] **by** *auto*

```
    from n5 show ?thesis by auto
    qed
   qed
  from 2 show ?thesis using assms cpoint-0 by blast
qed


lemma cpoint-2:
 assumes g (Inr σ) ∧
            (∀ k. g (Inr (isuffix k σ)) ⟶
                  (∃ m. f (Inl (subinterval σ k (m+k))) ∧
                      0 < m ∧ g (Inr (isuffix (m+k) σ))))

 shows   f (Inl (subinterval σ (cpoint f g i σ) (cpoint f g (Suc i) σ)))

proof
 (induct i)
 case 0
 then show ?case
 proof −
  have 1: g (Inr(isuffix 0 σ))
      using assms cpoint-0 cpoint-expand-0 by (simp add: isuffix-0)
  have 2: (∃ m. f (Inl (subinterval σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)))) ∧
                      0 < m ∧ g (Inr (isuffix (m+(cpoint f g 0 σ)) σ)))
      using assms 1 by auto
  have 3: (cpoint f g 1 σ) =
        (SOME x. (∃ m. f (Inl (subinterval σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)) ))
                  ∧ m>0 ∧ g (Inr (isuffix (m+(cpoint f g 0 σ)) σ))
                  ∧ x=m+(cpoint f g 0 σ))
             )

      by simp
  have 4: f (Inl (subinterval σ (cpoint f g 0 σ) ((cpoint f g 1 σ)) ))
      using 2 3 someI-ex[of λx. (∃ m. f (Inl (subinterval σ (cpoint f g 0 σ)
                                             (m+(cpoint f g 0 σ)) ))
                  ∧ m>0 ∧ g (Inr (isuffix (m+(cpoint f g 0 σ)) σ))
                  ∧ x=m+(cpoint f g 0 σ))] by auto
  from 4 show ?thesis by auto
  qed
 next
 case (Suc i)
 then show ?case
 proof −
  have n1: g (Inr (isuffix (cpoint f g (Suc i) σ) σ))
      using assms cpoint-0 by blast
  have n2: (∃ m. f (Inl (subinterval σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ)))) ∧
                      0 < m ∧ g (Inr (isuffix (m+(cpoint f g (Suc i) σ)) σ)))
      using assms n1 by auto
  have n3: (cpoint f g (Suc (Suc i)) σ) =
        (SOME x. (∃ m. f (Inl (subinterval σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
```

```
                ∧ m>0 ∧ g (Inr(isuffix (m+(cpoint f g (Suc i) σ)) σ))
                ∧ x=m+(cpoint f g (Suc i) σ))
              )
        using cpoint-expand-n by blast
  have n4: f (Inl (subinterval σ (cpoint f g (Suc i) σ) ((cpoint f g (Suc (Suc i)) σ))))
        using n2 n3 someI-ex[of λx. (∃ m. f (Inl (subinterval σ (cpoint f g (Suc i) σ)
                                                   (m+(cpoint f g (Suc i) σ))))
                ∧ m>0 ∧ g (Inr(isuffix (m+(cpoint f g (Suc i) σ)) σ))
                ∧ x=m+(cpoint f g (Suc i) σ)) ] by auto
  from n4 show ?thesis by auto
  qed
qed

lemma cpoint-3a:
 m>0 ∧ x=m+(cpoint f g (Suc i) σ) ⟹ (cpoint f g (Suc i) σ) < x
by auto


lemma cpoint-3:
 assumes g (Inr σ) ∧
              (∀ k. g (Inr (isuffix k σ)) ⟶
                     (∃ m. f (Inl (subinterval σ k (m+k))) ∧
                          0 < m ∧ g (Inr (isuffix (m+k) σ))))

 shows   (cpoint f g i σ) < (cpoint f g (Suc i) σ)

proof
 (induct i)
 case 0
 then show ?case
 proof −
  have 1: g (Inr(isuffix 0 σ))
        using assms cpoint-0 cpoint-expand-0 by (simp add: isuffix-0)
  have 2: (∃ m. f (Inl (subinterval σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)))) ∧
                          0 < m ∧ g (Inr (isuffix (m+(cpoint f g 0 σ)) σ)))
        using assms 1 by auto
  have 3: (cpoint f g 1 σ) =
           (SOME x. (∃ m. f (Inl (subinterval σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)) ))
                        ∧ m>0 ∧ g (Inr (isuffix (m+(cpoint f g 0 σ)) σ))
                        ∧ x=m+(cpoint f g 0 σ))
                )

        by simp
  have 4: (cpoint f g 0 σ) < (cpoint f g 1 σ)
        using 2 3 someI-ex[of λx. (∃ m. f (Inl (subinterval σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)) ))
                        ∧ m>0 ∧ g (Inr (isuffix (m+(cpoint f g 0 σ)) σ))
                        ∧ x=m+(cpoint f g 0 σ))] by auto
  from 4 show ?thesis by auto
  qed
 next
```

case (*Suc i*)
then show *?case*
proof −
have *n1*: *g* (*Inr* (*isuffix* (*cpoint f g* (*Suc i*) σ) σ))
using *assms cpoint-0* **by** *blast*
have *n2*: ($\exists$ *m*. *f* (*Inl* (*subinterval* σ (*cpoint f g* (*Suc i*) σ) (*m*+(*cpoint f g* (*Suc i*) σ)))) $\wedge$
0 < *m* $\wedge$ *g* (*Inr* (*isuffix* (*m*+(*cpoint f g* (*Suc i*) σ)) σ)))
using *assms n1* **by** *auto*
have *n3*: (*cpoint f g* (*Suc* (*Suc i*)) σ) =
(*SOME x*. ($\exists$ *m*. *f* (*Inl* (*subinterval* σ (*cpoint f g* (*Suc i*) σ) (*m*+(*cpoint f g* (*Suc i*) σ))))
$\wedge$ *m*>*0* $\wedge$ *g* (*Inr*(*isuffix* (*m*+(*cpoint f g* (*Suc i*) σ)) σ))
$\wedge$ *x*=*m*+(*cpoint f g* (*Suc i*) σ))
)
using *cpoint-expand-n* **by** *blast*
have *n4*: ($\exists$ *m*. *f* (*Inl* (*subinterval* σ (*cpoint f g* (*Suc i*) σ) (*m*+(*cpoint f g* (*Suc i*) σ))))
$\wedge$ *m*>*0* $\wedge$ *g* (*Inr*(*isuffix* (*m*+(*cpoint f g* (*Suc i*) σ)) σ))
$\wedge$ (*cpoint f g* (*Suc* (*Suc i*)) σ)=*m*+(*cpoint f g* (*Suc i*) σ))
using *n2 n3 someI-ex*[*of* λ*x*. ($\exists$ *m*. *f* (*Inl* (*subinterval* σ (*cpoint f g* (*Suc i*) σ) (*m*+(*cpoint f g* (*Suc i*) σ))))
$\wedge$ *m*>*0* $\wedge$ *g* (*Inr*(*isuffix* (*m*+(*cpoint f g* (*Suc i*) σ)) σ))
$\wedge$ *x*=*m*+(*cpoint f g* (*Suc i*) σ))] **by** *auto*
have *n5*: (*cpoint f g* (*Suc i*) σ) < (*cpoint f g* (*Suc* (*Suc i*)) σ)
using *n4* **using** *cpoint-3a* **by** *blast*
from *n5* **show** *?thesis* **by** *auto*
qed
qed

lemma *OmegaInductSem*:
(*w* $\models$ (*inf* $\wedge$ *g* $\wedge$ $\square$(*g* $\longrightarrow$ (*f* $\wedge$ *fmore*);*g*)) $\longrightarrow$ *omega f*)
proof (*cases w*)
case (*Inl a*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *finite-d-def finite-defs-1 int-simps*(*21*) *inteq-reflection unl-con unl-lift2*)
next
case (*Inr b*)
then show *?thesis*
proof −
have *1*: ((*Inr b*) $\models$ (*inf* $\wedge$ *g* $\wedge$ $\square$(*g* $\longrightarrow$ (*f* $\wedge$ *fmore*);*g*))) =
(*g* (*Inr b*) $\wedge$
($\forall$ *k*. *g* (*Inr* (*isuffix k b*)) $\longrightarrow$
($\exists$ *m*. *f* (*Inl* (*subinterval b k* (*m*+*k*))) $\wedge$
0 < *m* $\wedge$ *g* (*Inr* (*isuffix* (*m*+*k*) *b*)))))
by (*simp add*: *infinite-defs always-defs chop-defs fmore-defs isuffix-isuffix sum.case-eq-if*)
(*metis iprefix-isuffix iprefix-length add.commute*)
have *2*: (*g* (*Inr b*) $\wedge$
($\forall$ *k*. *g* (*Inr* (*isuffix k b*)) $\longrightarrow$
($\exists$ *m*. *f* (*Inl* (*subinterval b k* (*m*+*k*))) $\wedge$
0 < *m* $\wedge$ *g* (*Inr* (*isuffix* (*m*+*k*) *b*))))) $\longrightarrow$
infinite-index-sequence 0 (λ*i*. (*cpoint f g i b*))

```
  using 1 infinite-index-sequence-def cpoint-3 by (metis cpoint-expand-0)
  have 3: (g (Inr b) ∧
           (∀ k. g (Inr (isuffix k b)) ⟶
                 (∃ m. f (Inl (subinterval b k (m+k))) ∧
                      0 < m ∧ g (Inr (isuffix (m+k) b))))) ⟶
                        (∀ i. f (Inl (subinterval b ((λi. (cpoint f g  i b  )) i)
                                                   ((λi. (cpoint f g  i b  )) (Suc i)))))
  using 1 cpoint-2 by (metis )
  have 4: (g (Inr b) ∧
           (∀ k. g (Inr (isuffix k b)) ⟶
                 (∃ m. f (Inl (subinterval b k (m+k))) ∧
                      0 < m ∧ g (Inr (isuffix (m+k) b))))) ⟶
                       infinite-index-sequence 0 (λi. (cpoint f g  i b  )) ∧
                       (∀ i. f (Inl (subinterval b ((λi. (cpoint f g  i b  )) i)
                                                  ((λi. (cpoint f g  i b  )) (Suc i)))))
   using 2 3 by auto
  have 5: (g (Inr b) ∧
           (∀ k. g (Inr (isuffix k b)) ⟶
                 (∃ m. f (Inl (subinterval b k (m+k))) ∧
                      0 < m ∧ g (Inr (isuffix (m+k) b))))) ⟶
                       (∃ (ls::infiniteindex). infinite-index-sequence 0 ls ∧
                                          ls = (λi. (cpoint f g  i b  )))
   using 4 by auto
  have 6: (g (Inr b) ∧
           (∀ k. g (Inr (isuffix k b)) ⟶
                 (∃ m. f (Inl (subinterval b k (m+k))) ∧
                      0 < m ∧ g (Inr (isuffix (m+k) b))))) ⟶ ((Inr b) ⊨ omega f)
   using 3 5 unfolding omega-d-def by auto
  have 7: ((Inr b) ⊨ (inf ∧ g ∧ □(g ⟶ (f ∧ fmore);g)) ⟶ (omega f)  )
  using 6 1 by auto
  from 7 show ?thesis using Inr by blast
 qed
qed
```

11.6 Quantification over State (Flexible) Variables

Quantification in Infinite ITL is done similarly as in Finite ITL.

```
typedecl state

instance state :: world ..

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

11.7 Temporal Quantifiers

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

syntax
-Eex :: [*idts*, *lift*] ⇒ *lift* ((*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*

end

12 Infinite ITL: Axioms and Rules

theory *InfiniteITL*
imports
InfiniteSemantics
begin

The Infinite ITL axiom and proof rules are introduced (taken from [9]). The soundness of the rules and axioms are checked using the lemmas of InfiniteSemantics.thy.

12.1 Rules

lemma *MP* :
assumes ⊢ *f* ⟶ *g*
⊢ *f*
shows ⊢ *g*
using *assms*(*1*) *assms*(*2*) **by** *fastforce*

lemma *BoxGen* :
assumes ⊢ *f*
shows ⊢ □ *f*
using *assms*
by (*auto simp add*: *always-defs Valid-def sum*.*case-eq-if*)

lemma *BiGen*:
assumes ⊢ *f*
shows ⊢ *bi f*
using *assms*
by (*auto simp add*: *bi-defs Valid-def sum*.*case-eq-if*)

12.2 Axioms

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

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

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

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

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

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

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

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

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

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

lemma *OmegaUnroll*:
$\vdash f^\omega = (f \land$ *fmore*);f^ω
using *OmegaUnrollSem Valid-def* **by** *blast*

lemma *OmegaInduct*:
$\vdash$ (*inf* $\land g \land \Box(g \longrightarrow (f \land$ *fmore*);g)) $\longrightarrow$ *omega* f
using *OmegaInductSem Valid-def* **by** *blast*

12.3 Additional Lemmas

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

lemma *int-eq-true*:
assumes $\vdash P$
shows $\vdash P = \#True$
using *assms* **by** *auto*

lemma *int-eq*:
assumes $\vdash X = Y$
shows $X = Y$
using *assms* **by** (*auto simp*: *inteq-reflection*)

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

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

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

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

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

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

12.4 Quantification

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

lemma *EExE*:
assumes $\bigwedge x.\ \vdash F\, x \longrightarrow G$
shows $\vdash (\exists\exists\ x.\ F\, x) \longrightarrow G$
using *assms* **by** (*metis* (*mono-tags*, *lifting*) *Valid-def exist-state-d-def unl-lift2*)

lemma *EExValFinite*:
$((Inl\ w) \models (\exists\exists\ x.\ F\, x)) =$
$(\exists\ x\ (val :: 'a\ interval).\ (\ (\ val = (map\ x\ w) \wedge\ ((Inl\ w) \models F\, x))))$

by (*simp add*: *exist-state-d-def*)

lemma *EExValInfinite*:
$((\mathit{Inr}\ w) \models (\exists\exists\ x.\ F\ x)) =$
$(\exists\ x\ (\mathit{val} :: 'a\ \mathit{infinterval}).\ (\ (\ \mathit{val} = x \circ w \wedge\ ((\mathit{Inr}\ w) \models F\ x))))$
by (*simp add*: *exist-state-d-def*)

lemma *AAxDef*:
$\vdash (\forall\forall\ x.\ F\ x) = (\neg(\exists\exists\ x.\ \neg(F\ x)))$
by (*simp add*: *Valid-def forall-state-d-def exist-state-d-def*)

lemma *ExEqvRule*:
assumes $\bigwedge x.\ \vdash (f\ x) = (g\ x)$
shows $\vdash (\exists\ x.\ f\ x) = (\exists\ x.\ g\ x)$
using *assms* **by** *fastforce*

12.5 Lemmas about *current-val*

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

lemma *current-fun1*: $\vdash \$(f{<}x{>}) = f\ {<}\$x{>}$
by (*simp add*: *current-val-d-def intI sum.case-eq-if*)

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

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

lemma *current-forall*: $\vdash \$(\forall\ x.\ P\ x) = (\forall\ x.\ \$(P\ x))$
by (*auto simp*: *current-val-d-def intI sum.case-eq-if*)

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

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

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

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

12.6 Lemmas about *next-val*

lemma *next-const*: $\vdash \mathit{more} \longrightarrow (\#c)\$ = \#c$
by (*auto simp*: *next-val-d-def more-defs intI sum.case-eq-if*)

lemma *next-fun1*: ⊢ *more* ⟶ *f*<*x*>$ = *f*<*x*$>
by (*auto simp*: *next-val-d-def more-defs intI sum.case-eq-if*)

lemma *next-fun2*: ⊢ *more* ⟶ *f*<*x*,*y*>$ = *f* <*x*$,*y*$>
by (*auto simp*: *next-val-d-def more-defs intI sum.case-eq-if*)

lemma *next-fun3*: ⊢ *more* ⟶ *f*<*x*,*y*,*z*>$ = *f* <*x*$,*y*$,*z*$>
by (*auto simp*: *next-val-d-def more-defs intI sum.case-eq-if*)

lemma *next-forall*: ⊢ *more* ⟶ (∀ *x*. *P x*)$ = (∀ *x*. (*P x*)$)
by (*auto simp*: *next-val-d-def intI sum.case-eq-if*)

lemma *next-exists*: ⊢ *more* ⟶ (∃ *x*. *P x*)$ = (∃ *x*. (*P x*)$)
by (*auto simp*: *next-val-d-def intI sum.case-eq-if*)

lemma *next-exists1*: ⊢ *more* ⟶ (∃! *x*. *P x*)$ = (∃! *x*. (*P x*)$)
by (*auto simp*: *next-val-d-def more-defs intI sum.case-eq-if*)

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

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

12.7 Lemmas about *fin-val*

lemma *fin-const*: ⊢ *finite* ⟶ !(#*c*) = #*c*
by (*auto simp*: *fin-val-d-def finite-defs intI sum.case-eq-if*)

lemma *fin-fun1*: ⊢ *finite* ⟶!(*f*<*x*>) = *f* <!*x*>
by (*auto simp*: *fin-val-d-def finite-defs intI sum.case-eq-if*)

lemma *fin-fun2*: ⊢ *finite* ⟶ !(*f*<*x*,*y*>) = *f* <!*x*, !*y*>
by (*auto simp*: *fin-val-d-def finite-defs intI sum.case-eq-if*)

lemma *fin-fun3*: ⊢ *finite* ⟶ !(*f*<*x*,*y*,*z*>) = *f* <!*x*,!*y*,!*z*>
by (*auto simp*: *fin-val-d-def finite-defs intI sum.case-eq-if*)

lemma *fin-forall*: ⊢ *finite* ⟶ !(∀ *x*. *P x*) = (∀ *x*. !(*P x*))
by (*auto simp*: *fin-val-d-def finite-defs intI sum.case-eq-if*)

lemma *fin-exists*: ⊢ *finite* ⟶ !(∃ *x*. *P x*) = (∃ *x*. !(*P x*))
by (*auto simp*: *fin-val-d-def finite-defs intI sum.case-eq-if*)

lemma *fin-exists1*: ⊢ *finite* ⟶ !(∃! *x*. *P x*) = (∃! *x*. !(*P x*))
by (*auto simp*: *fin-val-d-def finite-defs intI sum.case-eq-if*)

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

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

12.8 Lemmas about *penult-val*

lemma *penult-const*: ⊢ *more* ∧ *finite* ⟶ (#*c*)! = #*c*
by (*auto simp*: *penult-val-d-def more-defs finite-defs intI sum.case-eq-if*)

lemma *penult-fun1*: ⊢ *more* ∧ *finite* ⟶ *f*<*x*>! = *f*<*x*!>
by (*auto simp*: *penult-val-d-def more-defs finite-defs intI sum.case-eq-if*)

lemma *penult-fun2*: ⊢ *more* ∧ *finite* ⟶ *f*<*x*,*y*>! = *f* <*x*!,*y*!>
by (*auto simp*: *penult-val-d-def more-defs finite-defs intI sum.case-eq-if*)

lemma *penult-fun3*: ⊢ *more* ∧ *finite* ⟶ *f*<*x*,*y*,*z*>! = *f* <*x*!,*y*!,*z*!>
by (*auto simp*: *penult-val-d-def more-defs finite-defs intI sum.case-eq-if*)

lemma *penult-forall*: ⊢ *more* ∧ *finite* ⟶ (∀ *x*. *P x*)! = (∀ *x*. (*P x*)!)
by (*auto simp*: *penult-val-d-def finite-defs intI sum.case-eq-if*)

lemma *penult-exists*: ⊢ *more* ∧ *finite* ⟶ (∃ *x*. *P x*)! = (∃ *x*. (*P x*)!)
by (*auto simp*: *penult-val-d-def finite-defs intI sum.case-eq-if*)

lemma *penult-exists1*: ⊢ *more* ∧ *finite* ⟶ (∃! *x*. *P x*)! = (∃! *x*. (*P x*)!)
by (*auto simp*: *penult-val-d-def more-defs finite-defs intI sum.case-eq-if*)

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

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

12.9 Basic temporal variables properties

lemma *empty-imp-fin-eqv-curr*:
⊢ *empty* ⟶ !*v* = $*v*
by (*simp add*: *Valid-def current-val-d-def empty-defs finval-defs sum.case-eq-if*)

lemma *skip-imp-fin-eqv-next*:
⊢ *skip* ⟶ !*v* = *v*$
by (*simp add*: *Valid-def skip-defs next-val-d-def finval-defs sum.case-eq-if*)

lemma *skip-imp-penult-eqv-curr*:
⊢ *skip* ⟶ *v*! = $*v*
by (*simp add*: *Valid-def skip-defs penultval-defs current-val-d-def sum.case-eq-if*)

end

13 Infinite ITL theorems using Weak Chop

theory *InfiniteTheorems*
imports
InfiniteITL
begin

We give the proofs of a list of Infinite ITL theorems. These proofs and theorems are from [8] but adapted for infinite and finite intervals.

13.1 Propositional reasoning

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

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

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

lemma *Prop02*:
assumes $\vdash f \longrightarrow g$
$\vdash f1 \longrightarrow g$
shows $\vdash f \vee f1 \longrightarrow g$
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* **using** *int-eq* **by** *auto*

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

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

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

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

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

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

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

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

lemma *Prop13*:
 assumes $\vdash f \longrightarrow g \vee h$
 shows $\vdash f \wedge \neg h \longrightarrow g$
using *assms* **by** (*auto simp*: *Valid-def*)

13.2 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 sum.case-eq-if*)

lemma *Finprop* :
 $\vdash ((\#True;(f \wedge empty)) \wedge (\#True;(g \wedge empty))) = (\#True;((f \wedge g) \wedge empty))$
 $\vdash ((\#True;(f \wedge empty)) \vee (\#True;(g \wedge empty))) = (\#True;((f \vee g) \wedge empty))$
 $\vdash (\#True;((\#True) \wedge empty))$
 $\vdash finite \longrightarrow (\neg\, (\#True;(f \wedge empty))) = (\#True;(\neg f \wedge empty))$
 $\vdash (\neg\, (\#True;(f \wedge empty))) = ((\#True;(\neg f \wedge empty)) \wedge finite)$

by (*auto simp*: *finalt-defs finite-defs sum.case-eq-if*)
(*auto simp add*: *chop-defs finite-defs empty-defs sum.case-eq-if*)

13.3 finite and inf properties

lemma *EmptyImpFinite*:
$\vdash$ *empty* $\longrightarrow$ *finite*
by (*simp add*: *empty-defs finite-defs intI sum.case-eq-if*)

lemma *SkipChopFiniteImpFinite*:
$\vdash$ *skip*;*finite* $\longrightarrow$ *finite*
by (*simp add*: *Valid-def finite-defs chop-defs skip-defs sum.case-eq-if*)

lemma *FiniteChopSkipImpFinite*:
$\vdash$ *finite*;*skip* $\longrightarrow$ *finite*
by (*simp add*: *Valid-def finite-defs chop-defs skip-defs sum.case-eq-if*)

lemma *FiniteChopSkipEqvFiniteAndMore*:
$\vdash$ *finite*;*skip* = (*finite* $\wedge$ *more*)
by (*simp add*: *Valid-def more-defs finite-defs chop-defs skip-defs sum.case-eq-if*)
(*metis Suc-leI add-diff-cancel-left′ diff-diff-cancel diff-is-0-eq′*
diff-le-self less-Suc0 nat-neq-iff plus-1-eq-Suc)

lemma *FiniteChopSkipEqvSkipChopFinite*:
$\vdash$ *finite*;*skip* = *skip*;*finite*
by (*simp add*: *Valid-def finite-defs chop-defs skip-defs sum.case-eq-if*)
(*metis diff-diff-cancel diff-le-self min.absorb1*)

lemma *FiniteAndEmptyEqvEmpty*:
$\vdash$ (*finite* $\wedge$ *empty*) = *empty*
by (*simp add*: *Valid-def empty-defs finite-defs chop-defs skip-defs sum.case-eq-if*)

lemma *FiniteChopFiniteEqvFinite*:
$\vdash$ *finite*;*finite* = *finite*
by (*simp add*: *Valid-def finite-defs chop-defs sum.case-eq-if*) *blast*

lemma *InfChopInfEqvInf*:
$\vdash$ *inf*;*inf* = *inf*
by (*simp add*: *Valid-def infinite-defs chop-defs sum.case-eq-if*)

lemma *InfChopFiniteEqvInf*:
$\vdash$ *inf*;*finite* = *inf*
by (*simp add*: *Valid-def infinite-defs chop-defs sum.case-eq-if*)

lemma *FiniteChopInfEqvInf*:
$\vdash$ *finite*;*inf* = *inf*
by (*simp add*: *Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if*)

lemma *InfEqvNotFinite*:

$\vdash$ *inf* = ($\neg$ *finite*)
by (*simp add*: *Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if*)

lemma *FiniteEqvNotInf*:
$\vdash$ *finite* = ($\neg$ *inf*)
by (*simp add*: *Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if*)

lemma *ChopTrueAndFiniteEqvAndFiniteChopFinite*:
$\vdash$ ((*f*;#*True*) $\wedge$ *finite*) = (*f* $\wedge$ *finite*);*finite*
by (*simp add*: *Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if*)

lemma *TrueChopAndFiniteEqvAndFiniteChopFinite*:
$\vdash$ ((#*True*;*f*) $\wedge$ *finite*) = *finite*;(*f* $\wedge$ *finite*)
by (*simp add*: *Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if*)

lemma *FiniteChopMoreEqvMore*:
$\vdash$ *finite*;*more* = *more*
by (*auto simp add*: *Valid-def more-defs infinite-defs finite-defs chop-defs sum.case-eq-if*)

lemma *ChopAndFiniteDist*:
$\vdash$ ((*f*;*g*) $\wedge$ *finite*) = (*f* $\wedge$ *finite*);(*g* $\wedge$ *finite*)
by (*simp add*: *Valid-def finite-defs chop-defs sum.case-eq-if*)

lemma *FiniteOrInfinite*:
$\vdash$ *finite* $\vee$ *inf*
by (*simp add*: *Valid-def finite-defs infinite-defs sum.case-eq-if*)

lemma *FiniteImpAnd*:
assumes $\vdash$ *finite* $\longrightarrow$ *f* = *g*
shows $\vdash$ (*f* $\wedge$ *finite*) = (*g* $\wedge$ *finite*)
using *assms* **by** (*auto simp add*: *Valid-def finite-defs*)

lemma *FmoreEqvSkipChopFinite*:
$\vdash$ *fmore* = *skip*;*finite*
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*
fmore-d-def inteq-reflection lift-and-com)

lemma *FiniteImp*:
$\vdash$ (*f* $\wedge$ *finite* $\longrightarrow$ *g*) = (*f* $\wedge$ *finite* $\longrightarrow$ *g* $\wedge$ *finite*)
by (*simp add*: *finite-defs Valid-def*)

lemma *ChopAndInf*:
$\vdash$ ((*f*;*g*) $\wedge$ *inf*) = (*f*;(*g* $\wedge$ *inf*))
by (*simp add*: *Valid-def chop-defs finite-defs infinite-defs sum.case-eq-if*)

lemma *ChopAndInfB*:
$\vdash$ ((*f*;*g*) $\wedge$ *inf*) = ((*f* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *finite*);(*g* $\wedge$ *inf*))
by (*auto simp add*: *Valid-def chop-defs finite-defs infinite-defs sum.case-eq-if*)

lemma *MoreAndInfEqvInf*:
$\vdash$ (*more* $\wedge$ *inf*) $=$ *inf*
by (*metis ChopAndInf EmptyImpFinite FiniteChopMoreEqvMore InfEqvNotFinite Prop11 Prop12 empty-d-def finite-d-def int-simps*(*32*) *inteq-reflection*)

lemma *AndInfChopAndInfEqvAndInf*:
$\vdash$ (*f* $\wedge$ *inf*);(*f* $\wedge$ *inf*) $=$ (*f* $\wedge$ *inf*)
by (*simp add*: *Valid-def infinite-defs chop-defs sum.case-eq-if*)

lemma *AndInfChopEqvAndInf*:
$\vdash$ (*f* $\wedge$ *inf*);*g* $=$ (*f* $\wedge$ *inf*)
by (*simp add*: *Valid-def chop-defs infinite-defs sum.case-eq-if*)

lemma *AndMoreAndInfEqvAndInf*:
$\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *inf*) $=$ (*f* $\wedge$ *inf*)
by (*simp add*: *Valid-def more-defs infinite-defs sum.case-eq-if*)

lemma *AndMoreAndFiniteEqvAndFmore*:
$\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) $=$ (*f* $\wedge$ *fmore*)
by (*simp add*: *Valid-def more-defs fmore-defs finite-defs sum.case-eq-if*)

lemma *NotFmoreAndEmpty*:
$\vdash$ $\neg$ (*empty* $\wedge$ *fmore*)
by (*auto simp add*: *fmore-d-def empty-d-def*)

lemma *NotFmoreAndInf*:
$\vdash$ $\neg$ ((*f* $\wedge$ *inf*) $\wedge$ *fmore*)
by (*auto simp add*: *fmore-d-def finite-d-def infinite-d-def*)

lemma *FmoreChopAnd*:
$\vdash$ (((*f* $\wedge$ *more*);*g*) $\wedge$ *fmore*) $=$ ((*f* $\wedge$ *fmore*);(*g* $\wedge$ *finite*))
by (*auto simp add*: *Valid-def more-defs fmore-defs chop-defs finite-defs sum.case-eq-if*)

lemma *NotEmptyAndInf*:
$\vdash$ $\neg$(*empty* $\wedge$ *inf*)
by (*simp add*: *Valid-def empty-defs infinite-defs sum.case-eq-if*)

lemma *OrFiniteInf*:
$\vdash$ *f* $=$ ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*))
by (*simp add*: *finite-defs Valid-def infinite-defs sum.case-eq-if*)

lemma *AndInfEqvChopFalse*:
$\vdash$ (*f* $\wedge$ *inf*) $=$ *f*;#*False*
by (*simp add*: *finite-defs Valid-def infinite-defs chop-defs sum.case-eq-if*)

13.4 Basic Theorems

lemma *BiChopImpChop* :

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

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

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

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

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

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

qed

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

lemma *RightChopEqvChop*:
assumes $\vdash g = g1$
shows $\vdash (f;g) = (f;g1)$
using *assms RightChopImpChop*[*of g g1 f*] *RightChopImpChop*[*of g1 g f*]
by *fastforce*

lemma *ChopOrEqv*:
$\vdash f;(g \vee g1) = (f;g \vee f;g1)$
proof −
have *1*: $\vdash g \longrightarrow g \vee g1$ **by** *auto*
hence *2*: $\vdash f;g \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
have *3*: $\vdash g1 \longrightarrow g \vee g1$ **by** *auto*
hence *4*: $\vdash f;g1 \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
from *2 4* **show** *?thesis* **by** (*meson ChopOrImp Prop02 Prop11*)
qed

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

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

lemma *LeftChopEqvChop*:

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

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

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

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

lemma *NextImpDist*:
$\vdash \bigcirc (f \longrightarrow g) \longrightarrow \bigcirc f \longrightarrow \bigcirc g$
proof –
have *1*: $\vdash (\neg (f \longrightarrow g)) = (f \wedge \neg g)$ **by** *auto*
hence *2*: $\vdash skip;(\neg (f \longrightarrow g)) = skip;(f \wedge \neg g)$ **by** (*rule RightChopEqvChop*)
have *3*: $\vdash f \longrightarrow g \vee (f \wedge \neg g)$ **by** *auto*
hence *4*: $\vdash skip;f \longrightarrow (skip;g) \vee (skip;(f \wedge \neg g))$ **by** (*rule ChopOrImpRule*)
hence *5*: $\vdash \neg (skip;(f \wedge \neg g)) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** *auto*

have *6*: ⊢ ¬ (*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 *FiniteChopImpDiamond*:
⊢ (*f* ∧ *finite*);*g* ⟶ ◇ *g*
proof −
have *1*: ⊢ *f* ∧ *finite* ⟶ *finite* **by** *auto*
hence *2*: ⊢ (*f* ∧ *finite*);*g* ⟶ *finite*;*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* ⟶ *finite* **by** (*rule EmptyImpFinite*)
hence *3*: ⊢ *empty*;*f* ⟶ *finite*;*f* **by** (*rule LeftChopImpChop*)
have *4*: ⊢ *f* ⟶ *finite*;*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*;(*finite*;*f*) = ((*skip*;*finite*);*f*) **by** (*rule ChopAssoc*)
hence *2*: ⊢ (*skip*;*finite*);*f* = *skip*;(*finite*;*f*) **by** *auto*
hence *3*: ⊢ (*skip*;*finite*);*f* = ○(◇*f*) **by** (*simp add*: *next-d-def sometimes-d-def*)
have *4*: ⊢ (*skip*;*finite*);*f* ⟶ ◇ *f*
by (*simp add*: *SkipChopFiniteImpFinite LeftChopImpChop sometimes-d-def*)
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*: $\vdash \Box\ f \longrightarrow f$ **by** (*metis always-d-def*)
have *4*: $\vdash \bigcirc\ (\ \Diamond\ (\neg\ f)) \longrightarrow \Diamond\ (\ \neg\ f\ )$ **by** (*rule NextDiamondImpDiamond*)
have *5*: $\vdash \neg\ \neg\ (\Diamond\ (\neg\ f)) \longrightarrow \Diamond(\ \neg\ f\ )$ **by** *auto*
hence *6*: $\vdash \bigcirc\ (\ \neg\ \neg\ (\Diamond\ (\neg\ f))\ ) \longrightarrow \bigcirc\ (\Diamond(\ \neg\ f\ ))$ **by** (*rule NextImpNext*)
have *7*: $\vdash \bigcirc\ (\ \neg\ \neg\ (\Diamond\ (\neg\ f))\ ) \longrightarrow \Diamond\ (\ \neg\ f\ )$ **using** *4 6* **by** *auto*
hence *8*: $\vdash \bigcirc\ (\ \neg(\ \Box\ f)) \longrightarrow \Diamond\ (\ \neg\ f\ )$ **by** (*simp add*: *always-d-def*)
hence *9*: $\vdash \neg\ (\Diamond\ (\ \neg\ f\ )) \longrightarrow \neg\ (\ \bigcirc\ (\ \neg(\ \Box\ f)))$ **by** *auto*
hence *10*:$\vdash \Box f \longrightarrow$ *wnext* $(\ \Box\ f\ )$ **by** (*simp add*: *always-d-def wnext-d-def*)
from *3 10* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpBoxRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \Box\ f \longrightarrow \Box\ g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ g \longrightarrow \neg\ f$ **by** *auto*
hence *3*: $\vdash \Box(\neg\ g \longrightarrow \neg\ f)$ **by** (*rule BoxGen*)
have *4*: $\vdash \Box(\neg\ g \longrightarrow \neg\ f) \longrightarrow (\mathit{finite};(\neg g)) \longrightarrow (\mathit{finite};(\neg f))$ **by** (*rule BoxChopImpChop*)
have *5*: $\vdash (\mathit{finite};(\neg g)) \longrightarrow (\mathit{finite};(\neg f))$ **using** *3 4 MP* **by** *blast*
hence *6*: $\vdash \Diamond\ (\neg g) \longrightarrow \Diamond(\neg f)$ **by** (*simp add*: *sometimes-d-def*)
hence *7*: $\vdash \neg\ (\Diamond(\neg f)\ ) \longrightarrow \neg(\ \Diamond\ (\neg g))$ **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *always-d-def*)
qed

lemma *BoxImpDist*:
$\vdash \Box(f \longrightarrow g) \longrightarrow \Box\ f \longrightarrow \Box\ g$
proof −
have *1*: $\vdash (f \longrightarrow g) \longrightarrow (\neg\ g \longrightarrow \neg\ f)$ **by** *auto*
hence *2*: $\vdash \Box(f \longrightarrow g) \longrightarrow \Box(\neg\ g \longrightarrow \neg\ f)$ **by** (*rule BoxImpBoxRule*)
have *3*: $\vdash \Box((\neg\ g) \longrightarrow \neg\ f) \longrightarrow (\mathit{finite};\ (\neg\ g)) \longrightarrow (\mathit{finite};\ (\neg\ f))$
by (*rule BoxChopImpChop*)
have *4*: $\vdash \Box(f \longrightarrow g) \longrightarrow (\mathit{finite};\ (\neg\ g)) \longrightarrow (\mathit{finite};\ (\neg\ f))$
using *2 3 lift-imp-trans* **by** *blast*
hence *5*: $\vdash \Box(f \longrightarrow g) \longrightarrow \Diamond(\neg\ g) \longrightarrow \Diamond(\neg\ f)$ **by** (*simp add*: *sometimes-d-def*)
hence *6*: $\vdash \Box(f \longrightarrow g) \longrightarrow \neg(\ \Diamond(\neg\ f)) \longrightarrow \neg(\ \Diamond(\neg\ g))$ **by** *auto*
from *6* **show** *?thesis* **by** (*simp add*: *always-d-def*)
qed

lemma *DiamondEmptyEqvFinite*:
$\vdash \Diamond\ \mathit{empty} = \mathit{finite}$
proof −
have *1*: $\vdash \mathit{finite};\ \mathit{empty} = \mathit{finite}$ **by** (*rule ChopEmpty*)
from *1* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

lemma *NextEqvNext*:
assumes $\vdash f = g$
shows $\vdash \bigcirc\ f = \bigcirc\ g$
proof −

have *1*: ⊢ *f*= *g* **using** *assms* **by** *auto*
hence *2*: ⊢ *skip*;*f* = *skip*;*g* **by** (*rule RightChopEqvChop*)
from *1* **show** *?thesis* **by** (*metis 2 next-d-def*)
qed

lemma *NextAndNextImpNextRule*:
assumes ⊢ (*f* ∧ *g*) ⟶ *h*
shows ⊢ (○ *f* ∧ ○ *g*) ⟶ ○ *h*
using *assms*
by (*simp add*: *Valid-def next-defs sum.case-eq-if*)

lemma *NextAndNextEqvNextRule*:
assumes ⊢ (*f* ∧ *g*) = *h*
shows ⊢ (○ *f* ∧ ○ *g*) = ○ *h*
using *assms*
by (*simp add*: *NextAndNextImpNextRule NextImpNext Prop11 Prop12*)

lemma *WeakNextEqvWeakNext*:
assumes ⊢ *f* = *g*
shows ⊢ *wnext* *f* = *wnext* *g*
using *assms* **using** *inteq-reflection* **by** *force*

lemma *DiamondImpDiamond*:
assumes ⊢ *f* ⟶ *g*
shows ⊢ ◇ *f* ⟶ ◇ *g*
using *assms* **by** (*simp add*: *RightChopImpChop sometimes-d-def*)

lemma *DiamondEqvDiamond*:
assumes ⊢ *f* = *g*
shows ⊢ ◇ *f* = ◇ *g*
using *assms* **using** *int-eq* **by** *force*

lemma *BoxEqvBox*:
assumes ⊢ *f* = *g*
shows ⊢ □ *f* = □ *g*
using *assms* **using** *inteq-reflection* **by** *force*

lemma *BoxAndBoxImpBoxRule*:
assumes ⊢ *f* ∧ *g* ⟶ *h*
shows ⊢ □ *f* ∧ □ *g* ⟶ □ *h*
using *assms* **by** (*auto simp*: *always-defs Valid-def sum.case-eq-if*)

lemma *BoxAndBoxEqvBoxRule*:
assumes ⊢ (*f* ∧ *g*) = *h*
shows ⊢ (□ *f* ∧ □ *g*) = □ *h*
using *assms BoxAndBoxImpBoxRule BoxImpBoxRule* **by** (*metis int-iffD1 int-iffD2 int-iffI Prop12*)

lemma *ImpBoxRule*:
assumes ⊢ *f* ⟶ *g*
shows ⊢ □ *f* ⟶ □ *g*

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

lemma *BoxIntro*:
 assumes $\vdash$ *f* $\longrightarrow$ *g*
 $\vdash$ *more* $\wedge$ *f* $\longrightarrow$ $\bigcirc$ *f*
 shows $\vdash$ *f* $\longrightarrow$ $\square$ *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*: *Valid-def next-defs empty-defs more-defs sum.case-eq-if*)
 hence *3*: $\vdash$ *f* $\longrightarrow$ *wnext f*
 by (*auto simp*: *Valid-def wnext-defs empty-defs next-defs sum.case-eq-if*)
 hence *4*: $\vdash$ $\square$(*f* $\longrightarrow$ *wnext f*)
 by (*rule BoxGen*)
 have *5*: $\vdash$ ($\square$ (*f* $\longrightarrow$ *wnext f*)) $\wedge$ *f* $\longrightarrow$ $\square$ *f*
 by (*rule BoxInduct*)
 hence *6*: $\vdash$ ($\square$ (*f* $\longrightarrow$ *wnext f*)) $\longrightarrow$ (*f* $\longrightarrow$ $\square$*f*)
 by *fastforce*
 have *7*: $\vdash$ *f* $\longrightarrow$ $\square$*f*
 using *4 6 MP* **by** *blast*
 have *8*: $\vdash$ $\square$*f* $\longrightarrow$ *f*
 by (*rule BoxElim*)
 have *9*: $\vdash$ *f* = $\square$ *f*
 using *7 8* **by** *fastforce*
 have *10*: $\vdash$ *f* $\longrightarrow$ *g*
 using *assms* **by** *auto*
 hence *11*: $\vdash$ $\square$*f* $\longrightarrow$ $\square$ *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 $\vdash$ *finite* $\longrightarrow$ $\neg$ *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*: *Valid-def more-defs wnext-defs next-defs sum.case-eq-if*)
 hence *3*: $\vdash$ *f* $\longrightarrow$ *wnext f*
 by *auto*
 hence *4*: $\vdash$ $\square$(*f* $\longrightarrow$ *wnext f*)
 by (*rule BoxGen*)
 have *5*: $\vdash$ $\square$ (*f* $\longrightarrow$ *wnext f*) $\wedge$ *f* $\longrightarrow$ $\square$ *f*
 by (*rule BoxInduct*)
 hence *6*: $\vdash$ $\square$ (*f* $\longrightarrow$ *wnext f*) $\longrightarrow$ (*f* $\longrightarrow$ $\square$*f*)
 by *fastforce*
 have *7*: $\vdash$ *f* $\longrightarrow$ $\square$*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*: $\vdash$ *finite* = ($\neg$($\Box$ *more*))
by (*auto simp*: *Valid-def finite-defs always-defs more-defs sum.case-eq-if*)
from *7 9 11 12* **show** *?thesis*
by *fastforce*
qed

lemma *WnextEqvEmptyOrNext*:
$\vdash$ *wnext f* = (*empty* $\vee \bigcirc f$)
by (*auto simp*: *Valid-def empty-defs wnext-defs next-defs sum.case-eq-if*)

lemma *NotEmptyAndNext*:
$\vdash \neg$(*empty* $\wedge \bigcirc f$)
by (*auto simp*: *Valid-def empty-defs next-defs sum.case-eq-if*)

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*: *Valid-def isuffix-isuffix always-defs sum.case-eq-if*)

lemma *DiamondIntroC*:
assumes $\vdash f \longrightarrow \bigcirc\ g$
shows $\vdash f \longrightarrow \Diamond\ g$
using *assms*
by (*metis* (*no-types*, *lifting*) *ChopAssoc FiniteChopSkipEqvSkipChopFinite NextChop NextDiamondImpDiamond NowImpDiamond inteq-reflection lift-imp-trans next-d-def sometimes-d-def*)

lemma *DiamondIntro*:
assumes $\vdash (f \wedge \neg\ g) \longrightarrow \bigcirc\ f$
shows $\vdash f \wedge \mathit{finite} \longrightarrow \Diamond\ g$
proof −
have *1*: $\vdash f \wedge \neg\ g \longrightarrow \bigcirc\ f$
  using *assms* **by** *auto*
hence *2*: $\vdash f \wedge \neg\ g \wedge (\Box\ (\neg\ g)) \longrightarrow (\bigcirc\ f) \wedge (\Box\ (\neg\ g))$
  by *auto*
have *3*: $\vdash (\Box\ (\neg\ g)) \longrightarrow \neg\ g$
  by (*rule BoxElim*)
hence *4*: $\vdash \Box\ (\neg\ g) = ((\Box\ (\neg\ g)) \wedge \neg\ g)$
  using *BoxImpBoxBox BoxBoxImpBox* **by** *fastforce*
have *5*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow \bigcirc\ f \wedge \Box\ (\neg\ g)$
  using *2 4* **by** *fastforce*
have *6*: $\vdash \Box\ (\neg\ g) = ((\neg\ g) \wedge \mathit{wnext}(\Box\ (\neg\ g)))$
  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*: *Valid-def always-defs more-defs next-defs wnext-defs sum.case-eq-if*)
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*: *Valid-def wnext-defs always-defs next-defs sum.case-eq-if*)
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*: $\vdash (f \wedge (\Box\ (\neg\ g))) \longrightarrow$ *more*
    using *9* **by** *auto*
hence *19*: $\vdash \Box(f \wedge (\Box\ (\neg\ g))) \longrightarrow \Box$ *more*
    by (*rule ImpBoxRule*)
have *20*: $\vdash$ *finite* $= (\neg(\Box$ *more*$))$
    by (*auto simp*: *Valid-def finite-defs always-defs more-defs sum.case-eq-if*)
have *21*: $\vdash$ *finite* $\longrightarrow \neg(f \wedge (\Box\ (\neg\ g)))$
    using *17 19 20* **by** *fastforce*
hence *22*: $\vdash$ *finite* $\longrightarrow \neg\ f \vee \neg\ (\Box\ (\neg\ g))$
    by *auto*
have *23*: $\vdash (\neg\ (\Box\ (\neg\ g))) = \Diamond\ g$
    by (*auto simp*: *always-d-def*)
from *22 23* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondIntroB*:
assumes $\vdash (f \wedge \neg\ \ g) \longrightarrow \bigcirc\ (f \wedge \neg\ g)$
shows $\vdash f \wedge$ *finite* $\longrightarrow \Diamond\ g$
proof –
have *1*: $\vdash (f \wedge \neg\ \ g) \longrightarrow \bigcirc\ (f \wedge \neg g)$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ *finite* $\longrightarrow \neg(f \wedge \neg\ g)$ **by** (*rule NextLoop*)
hence *3*: $\vdash f \wedge$ *finite* $\longrightarrow g$ **by** *auto*
have *4*: $\vdash g \longrightarrow \Diamond\ g$ **by** (*rule NowImpDiamond*)
from *3 4* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NextContra* :
assumes $\vdash (f \wedge \neg\ \ g) \longrightarrow (\bigcirc\ f \wedge \neg(\ \bigcirc\ g))$
shows $\vdash f \wedge$ *finite* $\longrightarrow g$
proof –
have *1*: $\vdash (f\wedge \neg\ \ g)\longrightarrow (\bigcirc\ f \wedge \neg(\ \bigcirc\ g))$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg(\ f \longrightarrow g) \longrightarrow \bigcirc\ (\ \neg(f \longrightarrow g))$ **by** (*auto simp*: *next-defs Valid-def sum.case-eq-if*)
hence *3*: $\vdash$ *finite* $\longrightarrow \neg\ \neg(\ f \longrightarrow g)$ **by** (*rule NextLoop*)
from *3* **show** *?thesis* **by** *auto*
qed

lemma *DiamondDiamondEqvDiamond*:
$\vdash \Diamond(\Diamond\ f) = \Diamond\ f$
proof –
have *1*: $\vdash$ *finite;finite* $=$ *finite* **by** (*simp add*: *FiniteChopFiniteEqvFinite*)
hence *2*: $\vdash$ (*finite;finite*)*;f* $=$ *finite;f* **using** *LeftChopEqvChop* **by** *blast*
have *3*: $\vdash$ (*finite;finite*)*;f* $=$ *finite;*(*finite;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 $\vdash$ *finite* $\longrightarrow$ *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*: $\vdash$ *finite* $\longrightarrow \neg\neg$ *f* **by** (*rule NextLoop*)
from *7* **show** *?thesis* **by** *auto*
qed

lemma *EmptyNextInducta*:
assumes $\vdash$ *empty* $\longrightarrow$ *f*
$\vdash \bigcirc$ *f* $\longrightarrow$ *f*
shows $\vdash$ *finite* $\longrightarrow$ *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*: $\vdash$ *finite* $\longrightarrow \neg\ \neg$ *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* $\wedge$ *finite* $\longrightarrow$ *g*
proof –
have *1*: $\vdash$ *empty* $\wedge$ *f* $\longrightarrow$ *g* **using** *assms* **by** *auto*
have *2*: $\vdash \bigcirc$(*f* $\longrightarrow$ *g*) $\wedge$ *f* $\longrightarrow$ *g* **using** *assms* **by** *blast*
have *3*: $\vdash$ (*empty* $\vee \bigcirc$(*f* $\longrightarrow$ *g*)) $\wedge$ *f* $\longrightarrow$ *g* **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *wnext* (*f* $\longrightarrow$ *g*) $\wedge$ *f* $\longrightarrow$ *g* **using** *WnextEqvEmptyOrNext* **by** *fastforce*
hence *5*: $\vdash$ *wnext* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*f* $\longrightarrow$ *g*) **by** *fastforce*
hence *6*: $\vdash \neg$ (*f* $\longrightarrow$ *g*) $\longrightarrow \neg$ (*wnext* (*f* $\longrightarrow$ *g*)) **by** *fastforce*
hence *7*: $\vdash \neg$ (*f* $\longrightarrow$ *g*) $\longrightarrow \bigcirc$ ($\neg$(*f* $\longrightarrow$ *g*)) **by** (*simp add*: *wnext-d-def*)
hence *8*: $\vdash$ *finite* $\longrightarrow \neg\ \neg$ (*f* $\longrightarrow$ *g*) **by** (*rule NextLoop*)
from *8* **show** *?thesis* **by** *auto*
qed

lemma *FinImpFin*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ *fin f* $\longrightarrow$ *fin g*
using *ImpBoxRule*[*of LIFT* (*empty* $\longrightarrow$ *f*) *LIFT* (*empty* $\longrightarrow$ *g*)] *assms*
fin-d-def [*of f*] *fin-d-def* [*of g*] **by** *fastforce*

lemma *FinEqvFin*:
assumes $\vdash f = g$
shows $\vdash$ *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 sum.case-eq-if*)
qed

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

lemma *HaltEqvHalt*:
assumes $\vdash f = g$
shows $\vdash$ *halt* $f =$ *halt* g
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*empty* $= f$) $=$ (*empty* $= g$) **by** *auto*
hence *3*: $\vdash \Box$(*empty* $= f$) $= \Box$ (*empty* $= g$) **by** (*rule BoxEqvBox*)
from *3* **show** *?thesis* **by** (*simp add*: *halt-d-def*)
qed

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

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

lemma *BiImpBiRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash$ *bi* $f \longrightarrow$ *bi* g
proof −

have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ g \longrightarrow \neg\ f$ **by** *auto*
hence *3*: $\vdash\ di\ (\neg\ g) \longrightarrow\ di\ (\neg\ f)$ **by** (*rule DiImpDi*)
hence *4*: $\vdash \neg\ (\ di\ (\neg\ f)) \longrightarrow \neg\ (\ di\ (\neg\ g))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

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

lemma *BiEqvBi*:
assumes $\vdash f = g$
shows $\vdash\ bi\ f =\ bi\ g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ f) = (\neg\ g)$ **by** *auto*
hence *3*: $\vdash\ di\ (\neg\ f) =\ di\ (\neg\ g)$ **by** (*rule DiEqvDi*)
hence *4*: $\vdash (\neg\ (di\ (\neg\ f))) = (\neg\ (\ di\ (\neg\ g)))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

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

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

lemma *BiAndChopImport*:
$\vdash\ bi\ f \wedge (f1;\ g) \longrightarrow (f \wedge f1);g$
proof −
have *1*: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence *2*: $\vdash bi\ f \longrightarrow bi\ (f1 \longrightarrow f \wedge f1)$ **by** (*rule BiImpBiRule*)
have *3*: $\vdash bi\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow\ f1;\ g \longrightarrow (f \wedge f1);\ g$ **by** (*rule BiChopImpChop*)

from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *StateAndChopImport*:
$\vdash$ (*init w*) $\wedge$ (*f*;*g*) $\longrightarrow$ ((*init w*) $\wedge$ *f*);*g*
proof −
have *1*: $\vdash$ (*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

13.5 Further Properties Di and Bi

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

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

lemma *StateChop*:
$\vdash$ (*init w*); *f* $\longrightarrow$ (*init w*)
by (*auto simp*: *DiState di-defs init-defs chop-defs sum.case-eq-if*
iprefix-0 iprefix-nth)

lemma *StateChopExportA*:
$\vdash$ ((*init w*) $\wedge$ *f*); *g* $\longrightarrow$ (*init w*)
using *DiState*
by (*auto simp*: *init-defs chop-defs sum.case-eq-if DiState*
iprefix-0 iprefix-nth)

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

lemma *ChopOrEqvRule*:

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

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

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

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

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

lemma *EmptyOrNextChopImpRule*:
assumes $\vdash f \longrightarrow empty \lor \bigcirc f1$
shows $\vdash f;\ g \longrightarrow g \lor \bigcirc(f1;\ g)$
proof −
have *1*: $\vdash f \longrightarrow empty \lor \bigcirc f1$ **using** *assms* **by** *auto*

hence *2*: ⊢ *f*; *g* ⟶ (*empty* ∨ ○ *f1*); *g* **by** (*rule LeftChopImpChop*)
have *3*: ⊢ (*empty* ∨ ○ *f1*); *g* = (*g* ∨ ○(*f1*; *g*)) **by** (*rule EmptyOrNextChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopEqvRule*:
assumes ⊢ *f* = (*empty* ∨ ○ *f1*)
shows ⊢ *f*; *g* = (*g* ∨ ○(*f1*; *g*))
proof −
have *1*: ⊢ *f* = (*empty* ∨ ○ *f1*) **using** *assms* **by** *auto*
hence *2*: ⊢ *f*; *g* = ((*empty* ∨ ○ *f1*); *g*) **by** (*rule LeftChopEqvChop*)
have *3*: ⊢ (*empty* ∨ ○ *f1*); *g* = (*g* ∨ ○(*f1*; *g*)) **by** (*rule EmptyOrNextChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

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

lemma *StateAndEmptyImpBoxState*:
⊢ (*init w*) ∧ *empty* ⟶ □ (*init w*)
by (*simp add*: *init-defs empty-defs always-defs Valid-def sum.case-eq-if*)

lemma *BoxEqvAndBox*:
⊢ □ *f* = (*f* ∧ □ *f*)
by (*simp add*: *always-defs Valid-def sum.case-eq-if*)
(*metis* (*no-types*, *lifting*) *interval-intlen-gr-zero interval-suffix-zero*
isl-def isuffix-0 projl-def sum.case(*1*) *sum.case-eq-if surjective-sum*)

lemma *NotBoxImpNotOrNotNextBox*:
⊢ ¬(□ *f*) ⟶ ¬*f* ∨ ¬(○ (□ *f*))
proof −
have *1*: ⊢ *f* ∧ (○ (□ *f*)) ⟶ □ *f*
using *BoxEqvAndEmptyOrNextBox* **by** *fastforce*
hence *2*: ⊢ ¬(□ *f*) ⟶ ¬(*f* ∧ (○ (□ *f*))) **by** *fastforce*
have *3*: ⊢ (¬(*f* ∧ (○ (□ *f*)))) = (¬*f* ∨ ¬(○ (□ *f*))) **by** *auto*
from *2 3* **show** *?thesis* **by** *auto*
qed

lemma *BoxStateChopBoxAndInfImpBox*:
⊢ □ (*init w*); □ (*init w*) ∧ *inf* ⟶ □ (*init w*)
by (*simp add*: *Valid-def always-defs chop-defs init-defs sum.case-eq-if infinite-defs*
iprefix-length iprefix-0)

(*metis add.right-neutral iprefix-length iprefix-nth isuffix-def le-cases le-iff-add*)

lemma *BoxStateChopBoxEqvBox*:
$\vdash$ $\Box$ (*init w*); $\Box$ (*init w*) = $\Box$ (*init w*)
proof –
have *1*: $\vdash$ ($\Box$ (*init w*)) = ((*init w*) $\wedge$ (*empty* $\vee$ $\bigcirc$($\Box$ (*init w*))))
by (*rule BoxEqvAndEmptyOrNextBox*)
hence *2*: $\vdash$ ($\Box$ (*init w*); $\Box$ (*init w*)) =
((*init w*) $\wedge$ ((*empty* $\vee$ $\bigcirc$($\Box$ (*init w*))); $\Box$ (*init w*)))
by (*metis StateAndChop inteq-reflection*)
have *3*: $\vdash$ ((*empty* $\vee$ $\bigcirc$($\Box$ (*init w*))); $\Box$ (*init w*)) =
($\Box$ (*init w*) $\vee$ $\bigcirc$($\Box$ (*init w*); $\Box$ (*init w*)))
by (*rule EmptyOrNextChopEqv*)
have *4*: $\vdash$ ($\Box$ (*init w*); $\Box$ (*init w*)) =
((*init w*) $\wedge$ ($\Box$ (*init w*) $\vee$ $\bigcirc$($\Box$ (*init w*); $\Box$ (*init w*))))
using *2 3* **by** *fastforce*
have *5*: $\vdash$ $\neg$ ($\Box$ (*init w*)) $\longrightarrow$ $\neg$ (*init w*) $\vee$ $\neg$($\bigcirc$($\Box$ (*init w*)))
by (*rule NotBoxImpNotOrNotNextBox*)
have *6*: $\vdash$ ($\Box$ (*init w*); $\Box$ (*init w*)) $\wedge$ $\neg$($\Box$ (*init w*)) $\longrightarrow$
$\bigcirc$($\Box$ (*init w*); $\Box$ (*init w*)) $\wedge$ $\neg$($\bigcirc$($\Box$ (*init w*)))
using *4 5* **by** *fastforce*
hence *7*: $\vdash$ $\Box$ (*init w*); $\Box$ (*init w*) $\wedge$ *finite* $\longrightarrow$ $\Box$ (*init w*)
by (*rule NextContra*)
have *8*: $\vdash$ $\Box$ (*init w*); $\Box$ (*init w*) $\wedge$ *inf* $\longrightarrow$ $\Box$ (*init w*)
by (*rule BoxStateChopBoxAndInfImpBox*)
have *9*: $\vdash$ $\Box$ (*init w*); $\Box$ (*init w*) $\wedge$ (*finite* $\vee$ *inf*) $\longrightarrow$ $\Box$ (*init w*)
using *7 8* **by** *fastforce*
hence *10*: $\vdash$ $\Box$ (*init w*); $\Box$ (*init w*) $\longrightarrow$ $\Box$ (*init w*)
using *FiniteOrInfinite* **by** *fastforce*
have *11*: $\vdash$ $\Box$ (*init w*) = ((*init w*) $\wedge$ $\Box$ (*init w*))
by (*rule BoxEqvAndBox*)
have *12*: $\vdash$ *empty* ; $\Box$ (*init w*) = $\Box$ (*init w*)
by (*rule EmptyChop*)
have *13*: $\vdash$ ((*init w*) $\wedge$ *empty*); $\Box$ (*init w*) = ((*init w*) $\wedge$ (*empty* ; $\Box$ (*init w*)))
by (*rule StateAndChop*)
have *14*: $\vdash$ $\Box$ (*init w*) = ((*init w*) $\wedge$ *empty*); $\Box$ (*init w*)
using *11 12 13* **by** *fastforce*
have *15*: $\vdash$ (*init w*) $\wedge$ *empty* $\longrightarrow$ $\Box$ (*init w*)
by (*rule StateAndEmptyImpBoxState*)
hence *16*: $\vdash$ ((*init w*) $\wedge$ *empty*); $\Box$ (*init w*) $\longrightarrow$ $\Box$ (*init w*); $\Box$ (*init w*)
by (*rule LeftChopImpChop*)
have *17*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ $\Box$ (*init w*); $\Box$ (*init w*)
using *14 16* **by** *fastforce*
from *10 17* **show** *?thesis* **by** *fastforce*
qed

lemma *NotBoxStateImpBoxYieldsNotBox*:
$\vdash$ $\neg$($\Box$ (*init w*)) $\longrightarrow$ ($\Box$ (*init w*)) *yields* ($\neg$($\Box$ (*init w*)))
proof –

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

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

lemma *FiniteChopEqvDiamond*:
$\vdash$ *finite*; $f = \Diamond f$
by (*simp add*: *sometimes-d-def*)

13.6 Properties of Da and Ba

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

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

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

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

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

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

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

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

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

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

lemma *ChopAndCommute*:

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

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

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

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

lemma *DiEqvNotBiNot*:
$\vdash$ $di\ f = (\neg(\ bi\ (\neg\ f)))$
proof −
have *1*: $\vdash bi\ (\neg\ f) = (\neg\ (\ di\ (\neg\neg\ f)))$ **by** (*simp add*: *bi-d-def*)

hence *2*: $\vdash$ *di* $(\neg\neg\ f) = (\neg($ *bi* $(\neg\ f)))$ **by** *auto*
have *3*: $\vdash f = (\neg\neg\ f)$ **by** *auto*
hence *4*: $\vdash$ *di* $f =$ *di* $(\neg\neg\ f)$ **by** (*rule DiEqvDi*)
from *2 4* **show** *?thesis* **by** *auto*
qed

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

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

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

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

lemma *BoxImpBoxImpBox*:
$\vdash \Box\ h \longrightarrow \Box(g \longrightarrow \Box\ h \wedge g\ )$
proof −
have *1*: $\vdash \Box\ h \longrightarrow (g \longrightarrow \Box\ h \wedge g\ )$ **by** *auto*

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

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

lemma *NotChopEqvYieldsNot*:
$\vdash\ \ (\neg\ (f;\ g)) = f\ \mathit{yields}\ \ (\neg\ \ g)$
proof –
have *1*: $\vdash g = (\neg\ \neg\ \ g)$ **by** *auto*
hence *2*: $\vdash f;\ g = f;\ (\neg\ \neg\ \ g)$ **by** (*rule RightChopEqvChop*)
hence *3*: $\vdash (\neg\ (f;\ g)) = (\neg\ (f;\ (\neg\ \neg\ \ g)))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *NotDiFalse*:
$\vdash\ \ \neg\ (\ \mathit{di}\ \ \#\mathit{False})$
proof –
have *1*: $\vdash (\mathit{init}\ \#\mathit{True}) \longrightarrow \mathit{bi}\ (\mathit{init}\ \#\mathit{True})$ **by** (*rule StateImpBi*)
hence *2*: $\vdash \#\mathit{True} \longrightarrow \mathit{bi}\ \#\mathit{True}$ **by** (*auto simp*: *bi-defs sum.case-eq-if*)
have *3*: $\vdash \#\mathit{True}$ **by** *auto*
have *4*: $\vdash \mathit{bi}\ \#\mathit{True}$ **using** *2 3 MP* **by** *auto*
hence *5*: $\vdash \neg\ (\ \mathit{di}\ (\neg\ \#\mathit{True}))$ **by** (*simp add*: *bi-d-def*)
have *6*: $\vdash (\neg\ \#\mathit{True}) =\ \ \#\mathit{False}$ **by** *auto*
hence *7*: $\vdash\ \ \mathit{di}\ (\neg\ \#\mathit{True}) =\ \ \mathit{di}\ \ \#\mathit{False}$ **by** (*rule DiEqvDi*)
from *5 7* **show** *?thesis* **by** *auto*
qed

lemma *StateAndEmptyChop*:
$\vdash\ \ ((\mathit{init}\ w) \wedge\ \ \mathit{empty}\ );\ f = ((\mathit{init}\ w) \wedge f)$
proof –
have *1*: $\vdash ((\mathit{init}\ w) \wedge\ \ \mathit{empty}\ );\ f = ((\mathit{init}\ w) \wedge\ \ \mathit{empty}\ ;\ f)$ **by** (*rule StateAndChop*)
have *2*: $\vdash\ \ \mathit{empty}\ ;\ f = f$ **by** (*rule EmptyChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndNextChop*:
$\vdash\ \ ((\mathit{init}\ w) \wedge \bigcirc\ f);\ g = ((\mathit{init}\ w) \wedge \bigcirc(f;\ g))$
proof –
have *1*: $\vdash ((\mathit{init}\ w) \wedge \bigcirc\ f);\ g = ((\mathit{init}\ w) \wedge (\bigcirc\ f);\ g)$ **by** (*rule StateAndChop*)
have *2*: $\vdash (\bigcirc\ f);\ g = \bigcirc(f;\ g)$ **by** (*rule NextChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NextAndEqvNextAndNext*:
$\vdash \bigcirc (f \wedge g) = (\bigcirc\, f \wedge \bigcirc\, g)$
by (*auto simp*: *next-defs sum.case-eq-if*)

lemma *NextStateAndChop*:
$\vdash \quad \bigcirc(((init\ w) \wedge f); g) = (\bigcirc\ (init\ w) \wedge \bigcirc(f; g))$
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 $\vdash (init\ w) \longrightarrow f$
shows $\vdash (init\ w) \longrightarrow bi\ \ f$
proof −
have *1*: $\vdash (init\ w) \longrightarrow f$ **using** *assms* **by** *auto*

hence *2*: ⊢ ¬ *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 ⊢ *f* ⟶ *f1*
shows ⊢ (*f1 yields g*) ⟶ (*f yields g*)

proof −
 have *1*: ⊢ *f* ⟶ *f1* **using** *assms* **by** *auto*
 hence *2*: ⊢ *f*; (¬ *g*) ⟶ *f1*; (¬ *g*) **by** (*rule LeftChopImpChop*)
 hence *3*: ⊢ ¬ (*f1*; (¬ *g*)) ⟶ ¬ (*f*; (¬ *g*)) **by** *auto*
 from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *LeftYieldsEqvYields*:
 assumes ⊢ *f* = *f1*
 shows ⊢ (*f yields g*) = (*f1 yields g*)
proof −
 have *1*: ⊢ *f* = *f1* **using** *assms* **by** *auto*
 hence *2*: ⊢ *f*; (¬ *g*) = *f1*; (¬ *g*) **by** (*rule LeftChopEqvChop*)
 hence *3*: ⊢ (¬ (*f*; (¬ *g*))) = (¬ (*f1*; (¬ *g*))) **by** *auto*
 from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

13.7 Properties of Fin

lemma *FinEqvTrueChopAndEmpty*:
 ⊢ *fin f* = #*True*;(*f* ∧ *empty*)
proof −
 have *1*: ⊢ *fin f* = □(*empty* ⟶ *f*)
 by (*simp add*: *fin-d-def*)
 have *2*: ⊢ □(*empty* ⟶ *f*) = (¬(◇(¬(*empty* ⟶ *f*))))
 by (*simp add*: *always-d-def*)
 have *3*: ⊢ (¬(*empty* ⟶ *f*)) = (¬ *f* ∧ *empty*)
 by *auto*
 hence *4*: ⊢ ◇(¬(*empty* ⟶ *f*)) = ◇(¬ *f* ∧ *empty*)
 using *DiamondEqvDiamond* **by** *blast*
 hence *5*: ⊢ ¬(◇(¬(*empty* ⟶ *f*))) = (¬(◇(¬ *f* ∧ *empty*)))
 by *auto*
 have *6*: ⊢ (¬(◇(¬ *f* ∧ *empty*))) = #*True*;(*f* ∧ *empty*)
 using *interval-suffix-intlast* **by** (*auto simp add*: *Valid-def sometimes-defs empty-defs chop-defs sum.case-eq-if*)
from *1 2 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondFin*:
⊢ ◇(*fin w*) = *fin w*
by (*metis* (*no-types*, *lifting*) *ChopAssoc ChopOrEqv FinEqvTrueChopAndEmpty FiniteChopFiniteEqvFinite FiniteChopInfEqvInf FiniteOrInfinite int-eq-true inteq-reflection sometimes-d-def*)

lemma *FiniteChopFinExportA*:
⊢ (*f* ∧ *finite*);(*g* ∧ *fin w*) ⟶ *fin w*
using *DiamondFin*
by (*metis ChopAndB FiniteChopImpDiamond 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* $\wedge$ *finite*);(*g* $\wedge$ *fin w*)) = (*fin w* $\wedge$ (*f* $\wedge$ *finite*);*g*)
using *FinAndChopImport FiniteChopFinExportA ChopAndA ChopAndCommute*
by *fastforce*

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

lemma *FinAndEmpty*:
$\vdash$ ((*fin w*) $\wedge$ *empty*) = (*w* $\wedge$ *empty*)
proof −
 have *1*: $\vdash$ ((*fin w*) $\wedge$ *empty*) = (#*True*;(*w* $\wedge$ *empty*) $\wedge$ *empty*)
  using *FinEqvTrueChopAndEmpty* **by** *fastforce*
 have *2*: $\vdash$ (#*True*;(*w* $\wedge$ *empty*) $\wedge$ *empty*) = ((#*True* $\wedge$ *empty*);(*w* $\wedge$ *empty*))
  using *ChopAndEmptyEqvEmptyChopEmpty*[*of LIFT*(#*True*) *LIFT*(*w* $\wedge$ *empty*)]
  by (*metis AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty Prop11 Prop12 int-eq*)
 have *3*: $\vdash$ (#*True* $\wedge$ *empty*);(*w* $\wedge$ *empty*) = (*empty*;(*w* $\wedge$ *empty*))
  using *LeftChopEqvChop* **by** *fastforce*
 have *4*: $\vdash$ (*empty*;(*w* $\wedge$ *empty*)) = (*w* $\wedge$ *empty*)
  using *EmptyChop* **by** *blast*
 from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *AndFinEqvChopAndEmpty*:
$\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin g*) = (*f* $\wedge$ *finite*); (*g* $\wedge$ *empty*)
proof −
 have *1*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin g*) = ((*f* $\wedge$ *finite*) ;*empty* $\wedge$ *fin g*)
  using *ChopEmpty* **by** (*metis inteq-reflection*)
 have *2*: $\vdash$ (*fin g* $\wedge$ (*f* $\wedge$ *finite*);*empty*) = ((*f* $\wedge$ *finite*);(*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* $\wedge$ *finite*);(*empty* $\wedge$ *fin g*) = (*f* $\wedge$ *finite*);(*g* $\wedge$ *empty*)

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

lemma *AndFinEqvChopStateAndEmpty*:
$\vdash$ $((f \wedge finite) \wedge$ fin $(init\ w)) = (f \wedge finite); ((init\ w) \wedge$ $empty$ $)$
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* (*no-types*, *lifting*) *5 DiamondFin NextDiamondImpDiamond Prop10 Prop12 int-eq lift-and-com*)
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*) $\wedge$ *finite*); *f* = $\diamond$ ((*init w*) $\wedge$ *f*)
proof −
have *1*: $\vdash$ (*fin* (*init w*) $\wedge$ *finite*) = (*finite*;((*init w*) $\wedge$ *empty*))
by (*metis AndFinEqvChopAndEmpty int-simps*(*17*) *inteq-reflection lift-and-com*)
hence *2*: $\vdash$ (*fin* (*init w*) $\wedge$ *finite*);*f* = (*finite*;((*init w*) $\wedge$ *empty*));*f*
by (*rule LeftChopEqvChop*)
have *3*: $\vdash$ *finite*;(((*init w*) $\wedge$ *empty*);*f*) = (*finite*;((*init w*) $\wedge$ *empty*));*f*
by (*rule ChopAssoc*)
have *4*: $\vdash$ *finite*;(((*init w*) $\wedge$ *empty*);*f*)= $\diamond$ (((*init w*) $\wedge$ *empty*);*f*)
by (*simp add*: *sometimes-d-def*)
have *5*: $\vdash$ ((*init w*) $\wedge$ *empty*);*f* = ((*init w*) $\wedge$ *f*)
using *StateAndEmptyChop* **by** *blast*
hence *6*: $\vdash$ $\diamond$ (((*init w*) $\wedge$ *empty*);*f*) = $\diamond$ ((*init w*) $\wedge$ *f*)
by (*rule DiamondEqvDiamond*)
from *2 3 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *NotDiamondAndNot*:
$\vdash \neg(\diamond (f \wedge \neg f))$
proof −
have *1*: $\vdash (\neg(\diamond (f \wedge \neg f))) = \Box(\neg(f \wedge \neg f))$ **using** *NotDiamondNotEqvBox* **by** *fastforce*
have *2*: $\vdash \neg(f \wedge \neg f)$ **by** *simp*
have *3*: $\vdash \Box(\neg(f \wedge \neg f))$ **using** *2* **by** (*simp add*: *BoxGen*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FinYields*:
$\vdash$ (*fin* (*init w*) $\wedge$ *finite*) *yields* (*init w*)
proof −
have *1*: $\vdash$ (*fin* (*init w*) $\wedge$ *finite*); ($\neg$(*init w*)) = $\diamond$((*init w*) $\wedge$ $\neg$(*init w*))
by (*rule FinChopEqvDiamond*)
have *2*: $\vdash$ $\neg$($\diamond$((*init w*) $\wedge$ $\neg$ (*init w*))) **by** (*rule NotDiamondAndNot*)
have *3*: $\vdash$ $\neg$ ((*fin* (*init w*) $\wedge$ *finite*); ($\neg$ (*init w*))) **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ImpAndFinStateOrFinNotState*:
$\vdash$ *f* $\longrightarrow$ (*f* $\wedge$ *fin* (*init w*)) $\vee$ (*f* $\wedge$ *fin* ($\neg$ (*init w*)))
by (*simp add*: *fin-defs Valid-def sum.case-eq-if*)

lemma *AndFinChopEqvStateAndChop*:
$\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); *g* = (*f* $\wedge$ *finite*); ((*init w*) $\wedge$ *g*)
proof −
have *1*: $\vdash$ (*fin* (*init w*) $\wedge$ *finite*) *yields* (*init w*)
by (*rule FinYields*)
have *2*: $\vdash$ (*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*) $\longrightarrow$ *fin* (*init w*)

by *auto*
hence *3*: $\vdash$ (*fin* (*init w*) $\wedge$ *finite*) *yields* (*init w*) $\longrightarrow$
((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)) *yields* (*init w*)
using *LeftYieldsImpYields*
by (*metis AndFinEqvChopAndEmpty Prop11 Prop12 inteq-reflection*)
have *4*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)) *yields* (*init w*)
using *1 3 MP* **by** *fastforce*
have *5*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); *g* $\wedge$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)) *yields* (*init w*)
$\longrightarrow$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); (*g* $\wedge$ (*init w*))
by (*rule ChopAndYieldsImp*)
have *6*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); *g* $\longrightarrow$
((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); (*g* $\wedge$ (*init w*))
using *4 5* **by** *fastforce*
have *7*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); (*g* $\wedge$ (*init w*)) $\longrightarrow$ (*f* $\wedge$ *finite*); (*g* $\wedge$ (*init w*))
by (*rule AndChopA*)
have *8*: $\vdash$ *g* $\wedge$ (*init w*) $\longrightarrow$ (*init w*) $\wedge$ *g*
by *auto*
hence *9*: $\vdash$ (*f* $\wedge$ *finite*); (*g* $\wedge$ (*init w*)) $\longrightarrow$ (*f* $\wedge$ *finite*); ((*init w*) $\wedge$ *g*)
by (*rule RightChopImpChop*)
have *10*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); *g* $\longrightarrow$ (*f* $\wedge$ *finite*); ((*init w*) $\wedge$ *g*)
using *6 7 9* **by** *fastforce*
have *11*: $\vdash$ (*f* $\wedge$ *finite*) $\longrightarrow$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)) $\vee$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* ($\neg$ (*init w*)))
using *ImpAndFinStateOrFinNotState* **by** *blast*
hence *12*: $\vdash$ (*f* $\wedge$ *finite*); ((*init w*) $\wedge$ *g*) $\longrightarrow$
(((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)) $\vee$
((*finite* $\wedge$ *f*) $\wedge$ *fin* ($\neg$ (*init w*)))); ((*init w*) $\wedge$ *g*)
using *LeftChopImpChop*
by (*metis inteq-reflection lift-and-com*)
have *13*: $\vdash$ (((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)) $\vee$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* ($\neg$ (*init w*)))); ((*init w*) $\wedge$ *g*)
=
(((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*));
((*init w*) $\wedge$ *g*) $\vee$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* ($\neg$ (*init w*))); ((*init w*) $\wedge$ *g*))
by (*rule OrChopEqv*)
have *14*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init* ($\neg$ *w*))); ((*init w*) $\wedge$ *g*) $\longrightarrow$
$\Diamond$((*init* ($\neg$ *w*)) $\wedge$ ((*init w*) $\wedge$ *g*))
using *FinChopEqvDiamond*
by (*metis AndFinEqvChopAndEmpty ChopEmpty FiniteChopImpDiamond LeftChopImpChop int-eq*)
have *141*: $\vdash$ $\neg$($\Diamond$((*init* ($\neg$ *w*)) $\wedge$ ((*init w*) $\wedge$ *g*))) $\longrightarrow$
$\neg$ (((*f* $\wedge$ *finite*) $\wedge$ *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 sum.case-eq-if*)
have *151*: $\vdash$ $\neg$ (((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init* ($\neg$ *w*))); ((*init w*) $\wedge$ *g*))
using *15 141* **by** *fastforce*
have *1511*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *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* $\wedge$ *finite*) $\wedge$ *fin* (*init w*));
((*init w*) $\wedge$ *g*) $\vee$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* ($\neg$ (*init w*))); ((*init w*) $\wedge$ *g*) $\longrightarrow$
((*f* $\wedge$ *finite*)$\wedge$ *fin* (*init w*)); ((*init w*) $\wedge$ *g*)
using *1511* **by** *fastforce*

have *16*: $\vdash$ $(f \wedge finite)$; $((init\ w) \wedge g) \longrightarrow ((f \wedge finite) \wedge\ \ fin\ \ (init\ w))$; $((init\ w) \wedge g)$
using *12 13 152*
proof $-$
have $\vdash (f \wedge finite);(init\ w \wedge g) \longrightarrow$
$((f \wedge finite) \wedge fin\ (init\ w) \vee (f \wedge finite) \wedge fin\ (\neg\ init\ w));(init\ w \wedge g)$
by (*metis 12 inteq-reflection lift-and-com*)
then show *?thesis*
using *13 152* **by** *fastforce*
qed
have *17*: $\vdash ((f \wedge finite) \wedge\ \ fin\ \ (init\ w))$; $((init\ w) \wedge g) \longrightarrow ((f \wedge finite) \wedge fin\ (init\ w))$; g
by (*rule ChopAndB*)
have *18*: $\vdash (f \wedge finite)$; $((init\ w) \wedge g) \longrightarrow ((f \wedge finite) \wedge\ \ fin\ \ (init\ w))$; g
using *16 17* **by** *fastforce*
from *10 18* **show** *?thesis* **by** *fastforce*
qed

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

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

lemma *BiImpFinEqvYieldsState*:
$\vdash\ \ bi\ (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w)) = (f\ \wedge finite) yields\ \ (init\ w)$
proof $-$
have *1*: $\vdash\ \ di\ ((f \wedge finite) \wedge fin\ (init\ (\neg\ w))) = (f \wedge finite)$; $\ \ (init\ (\neg\ w))$
by (*rule DiAndFinEqvChopState*)
have *2*: $\vdash ((f \wedge finite) \wedge fin(init\ (\neg\ w)))\ \ = ((f \wedge finite) \wedge \neg(fin(init\ w)))$
using *FinNotStateEqvNotFinState* **by** *fastforce*
have *3*: $\vdash ((f \wedge finite) \wedge \neg\ (fin(init\ w))) = (\neg\ (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w)))$
by *auto*
have *4*: $\vdash ((f \wedge finite) \wedge fin(init\ \ (\neg\ w))) = (\neg\ (f \wedge finite \longrightarrow\ \ fin(init\ w)))$
using *2 3* **by** *fastforce*
hence *5*: $\vdash\ \ di\ ((f \wedge finite) \wedge\ \ fin\ \ \ (init\ (\neg\ w))) =\ \ di\ (\neg\ (f \wedge finite \longrightarrow fin(init\ w)))$
by (*rule DiEqvDi*)
have *6*: $\vdash\ \ di\ (\neg\ (f \wedge finite \longrightarrow\ \ fin\ \ (init\ w))) = (\neg(\ bi\ (f \wedge finite \longrightarrow fin(init\ w))))$

by (*rule DiNotEqvNotBi*)
have *7*: $\vdash \neg$ (*bi* (*f* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w*))) = (*f* $\wedge$ *finite*);(*init* ($\neg$ *w*))
using *1 5 6 Initprop* **by** *fastforce*
hence *8*: $\vdash$ *bi* (*f* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w*)) = ($\neg$ ((*f* $\wedge$ *finite*); ($\neg$ (*init w*))))
by (*metis Initprop*(*2*) *int-eq int-simps*(*7*))
from *8* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

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

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

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

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

lemma *RightYieldsImpYields*:
assumes $\vdash$ *g* $\longrightarrow$ *g1*
shows $\vdash$ (*f yields g*) $\longrightarrow$ (*f yields g1*)
proof −
have *1*: $\vdash$ *g* $\longrightarrow$ *g1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ $\neg$ *g1* $\longrightarrow$ $\neg$ *g* **by** *auto*

hence *3*: $\vdash f; (\neg\ g1) \longrightarrow f; (\neg\ g)$ **by** (*rule RightChopImpChop*)
hence *4*: $\vdash \neg\ (f; (\neg\ g)) \longrightarrow \neg\ (f; (\neg\ g1))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *RightYieldsEqvYields*:
assumes $\vdash\ g = g1$
shows $\vdash (f\ \mathit{yields}\ g) = (f\ \mathit{yields}\ g1)$
proof −
have *1*: $\vdash g = g1$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ g) = (\neg\ g1)$ **by** *auto*
hence *3*: $\vdash f; (\neg\ g) = f; (\neg\ g1)$ **by** (*rule RightChopEqvChop*)
hence *4*: $\vdash (\neg\ (f; (\neg\ g))) = (\neg\ (f; (\neg\ g1)))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *BoxImpYields*:
$\vdash\ \Box\, g \longrightarrow (f \wedge \mathit{finite})\ \mathit{yields}\ g$
proof −
have *1*: $\vdash (f \wedge \mathit{finite}); (\neg\ g) \longrightarrow \Diamond(\neg\ g)$ **by** (*rule FiniteChopImpDiamond*)
hence *2*: $\vdash \neg\ (\Diamond(\neg\ g)) \longrightarrow \neg\ ((f \wedge \mathit{finite}); (\neg\ g))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *yields-d-def always-d-def*)
qed

lemma *BoxEqvFiniteYields*:
$\vdash\ \Box\, f = \mathit{finite\ yields}\ f$
proof −
have *1*: $\vdash \mathit{finite}; (\neg\ f) = \Diamond\ (\neg\ f)$ **by** (*rule FiniteChopEqvDiamond*)
hence *2*: $\vdash (\neg\ (\mathit{finite}; (\neg\ f))) = (\neg(\ \Diamond\ (\neg\ f)))$ **by** *auto*
have *3*: $\vdash \Box\ f = (\neg\ (\ \Diamond\ (\neg\ f)))$ **by** (*simp add*: *always-d-def*)
have *4*: $\vdash \Box\ f = (\neg\ (\mathit{finite}; (\neg\ f)))$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *YieldsGen*:
assumes $\vdash\ g$
shows $\vdash (f \wedge \mathit{finite})\ \mathit{yields}\ g$
proof −
have *1*: $\vdash g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ g$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box\ g \longrightarrow (f \wedge \mathit{finite})\ \mathit{yields}\ g$ **by** (*rule BoxImpYields*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *YieldsAndYieldsEqvYieldsAnd*:
$\vdash\ ((f\ \mathit{yields}\ g) \wedge (f\ \mathit{yields}\ g1)) = f\ \mathit{yields}\ (g \wedge g1)$
proof −
have *1*: $\vdash f; (\neg\ g \vee\ \neg\ g1) = ((f; (\neg\ g)) \vee\ (f; (\neg\ g1)))$ **by** (*rule ChopOrEqv*)
hence *2*: $\vdash ((f; (\neg\ g)) \vee\ (f; (\neg\ g1))) = f; (\neg\ g \vee\ \neg\ g1)$ **by** *auto*
have *3*: $\vdash (\neg\ g \vee\ \neg\ g1)\ =\ (\neg\ (g \wedge g1))$ **by** *auto*

hence *4*: $\vdash f; (\neg\ g \vee\ \neg\ g1) = f; (\neg (g \wedge g1))$ **by** (*rule RightChopEqvChop*)
have *5*: $\vdash (f; (\neg\ g)) \vee (f; (\neg\ g1)) = f; (\neg (g \wedge g1))$ **using** *2 4* **by** *fastforce*
hence *6*: $\vdash (\neg (f; (\neg\ g)) \wedge \neg (f; (\neg\ g1))) = (\neg (f; (\neg (g \wedge g1))))$
by (*auto simp*: *chop-defs sum.case-eq-if*)
from *6* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

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

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

lemma *FmoreChopImpFmore*:
$\vdash$ *fmore* ; (*f* $\wedge$ *finite*) $\longrightarrow$ *fmore*
proof −
have *1*: $\vdash$ *fmore*; (*f* $\wedge$ *finite*)= $\bigcirc$(*finite*; (*f* $\wedge$ *finite*))
using *FmoreEqvSkipChopFinite* **by** (*metis NextChop inteq-reflection next-d-def*)
have *2*: $\vdash$ $\bigcirc$(*finite*; (*f* $\wedge$ *finite*)) $\longrightarrow$ *fmore*
by (*auto simp*: *fmore-defs chop-defs finite-defs more-defs next-defs sum.case-eq-if*)
have *3*: $\vdash$ ($\bigcirc$*finite*; (*f* $\wedge$ *finite*)) $\longrightarrow$ *fmore* **using** *1 2*
by (*metis FmoreEqvSkipChopFinite inteq-reflection next-d-def*)

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

lemma *ChopMoreImpMore*:
$\vdash$ f; *more* $\longrightarrow$ *more*
proof −
have *1*: $\vdash$ (f $\wedge$ *finite*) ; *more* $\longrightarrow$ $\Diamond$ *more* **by** (*rule FiniteChopImpDiamond*)
have *11*: $\vdash$ (f $\wedge$ *inf*) ; *more* $\longrightarrow$ *more*
by (*simp add*: *more-defs infinite-defs chop-defs Valid-def sum.case-eq-if*)
have *2*: $\vdash$ $\Diamond$ *more* $\longrightarrow$ *more* **by** (*auto simp*: *more-defs sometimes-defs sum.case-eq-if*)
have *3*: $\vdash$ (f $\wedge$ *finite*) ; *more* $\longrightarrow$ *more* **using** *1 2* **by** *fastforce*
have *4*: $\vdash$ f = ((f $\wedge$ *finite*) $\vee$ (f $\wedge$ *inf*))
by (*simp add*: *Valid-def finite-defs infinite-defs sum.case-eq-if*)
hence *5*: $\vdash$ f;*more* = ((f $\wedge$ *finite*);*more* $\vee$ (f $\wedge$ *inf*);*more*) **by** (*simp add*: *OrChopEqvRule*)
from *11 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreChopEqvNextDiamond*:
$\vdash$ *fmore* ; f = $\bigcirc$($\Diamond$ f)
proof −
have *1*: $\vdash$ *fmore* ; f = ($\bigcirc$ *finite*); f
by (*simp add*: *Valid-def chop-defs fmore-defs next-defs finite-defs sum.case-eq-if*)
have *2*: $\vdash$ ($\bigcirc$ *finite*); f = $\bigcirc$(*finite*; f) **by** (*rule NextChop*)
have *3*: $\vdash$ *fmore* ; f = $\bigcirc$(*finite*; f) **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

lemma *WeakNextBoxImpMoreYields*:
$\vdash$ *fmore* *yields* f = *wnext*($\Box$ f)
proof −
have *1*: $\vdash$ *fmore* ; ($\neg$ f) = $\bigcirc$($\Diamond$ ($\neg f$)) **by** (*rule MoreChopEqvNextDiamond*)
have *2*: $\vdash$ $\bigcirc$($\Diamond$ ($\neg f$)) = $\bigcirc$($\neg$($\Box f$)) **by** (*auto simp*: *always-d-def*)
have *3*: $\vdash$ $\bigcirc$($\neg$($\Box f$)) = ($\neg$ (*wnext*($\Box$ f))) **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash$ *fmore* ; ($\neg$ f) = ($\neg$(*fmore* *yields* f)) **by** (*simp add*: *yields-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

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

lemma *LeftChopImpMoreRule*:

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

lemma *LeftChopImpFMoreRule*:
assumes $\vdash$ *f* $\longrightarrow$ *fmore*
shows $\vdash$ *f*; (*g* $\wedge$ *finite*) $\longrightarrow$ *fmore*
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *fmore* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; (*g* $\wedge$ *finite*) $\longrightarrow$ *fmore* ; (*g* $\wedge$ *finite*) **by** (*rule LeftChopImpChop*)
have *3*: $\vdash$ *fmore* ; (*g* $\wedge$ *finite*) $\longrightarrow$ *fmore* **using** *FmoreChopImpFmore* **by** *fastforce*
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

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

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

lemma *ChopImpDi*:
$\vdash$ *f*; *g* $\longrightarrow$ *di* *f*
proof −
have *1*: $\vdash$ *g* $\longrightarrow$ #*True* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ *f*; #*True* **by** (*rule RightChopImpChop*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *TrueEqvTrueChopTrue*:
$\vdash$ #*True* = #*True*; #*True*
proof −
have *1*: $\vdash$ #*True*; #*True* $\longrightarrow$ #*True* **by** *auto*

have *2*: $\vdash$ #*True* $\longrightarrow$ *di* #*True* **by** (*rule DiIntro*)
hence *3*: $\vdash$ #*True* $\longrightarrow$ #*True*; #*True* **by** (*simp add*: *di-d-def*)
from *1 3* **show** *?thesis* **by** *auto*
qed

lemma *DiEqvDiDi*:
$\vdash$ *di f* = *di* (*di f*)
proof −
have *1*: $\vdash$ #*True* = #*True*; #*True* **by** (*rule TrueEqvTrueChopTrue*)
hence *2*: $\vdash$ *f*; #*True* = *f*; (#*True*; #*True*) **by** (*rule RightChopEqvChop*)
have *3*: $\vdash$ *f*; (#*True*; #*True*)= (*f*; #*True*); #*True* **by** (*rule ChopAssoc*)
have *4*: $\vdash$ *f*; #*True* = (*f*; #*True*); #*True* **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis di-d-def*)
qed

lemma *BiEqvBiBi*:
$\vdash$ *bi f* = *bi*(*bi f*)
proof −
have *1*: $\vdash$ *di* ($\neg$ *f*) = *di*(*di* ($\neg$ *f*)) **by** (*rule DiEqvDiDi*)
have *2*: $\vdash$ *di* ($\neg$ *f*) = ($\neg$ (*bi f*)) **by** (*rule DiNotEqvNotBi*)
hence *3*: $\vdash$ *di* (*di* ($\neg$ *f*)) = *di* ($\neg$ (*bi f*)) **by** (*rule DiEqvDi*)
have *4*: $\vdash$ *di* ($\neg$ *f*) = *di* ($\neg$(*bi f*)) **using** *1 3* **by** *fastforce*
hence *5*: $\vdash$ ($\neg$ (*di* ($\neg$ *f*))) = ($\neg$ (*di* ($\neg$ (*bi f*)))) **by** *fastforce*
from *5* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *DiOrEqv*:
$\vdash$ *di* (*f* $\vee$ *g*) = (*di f* $\vee$ *di g*)
proof −
have *1*: $\vdash$ (*f*$\vee$ *g*); #*True* = (*f*; #*True* $\vee$ *g*; #*True*) **by** (*rule OrChopEqv*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

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

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

lemma *DiAndImpAnd*:
$\vdash$ *di* (*f* $\wedge$ *g*) $\longrightarrow$ *di f* $\wedge$ *di g*
proof −

have *1*: ⊢ *di* (*f* ∧ *g*) ⟶ *di f* **by** (*rule DiAndA*)
have *2*: ⊢ *di* (*f* ∧ *g*) ⟶ *di g* **by** (*rule DiAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DiSkipEqvMore*:
⊢ *di skip* = *more*
proof −
have *1*: ⊢ *skip* ; #*True* = ○#*True* **by** (*rule SkipChopEqvNext*)
have *2*: ⊢ ○#*True* = *more* **by** (*auto simp*: *more-d-def*)
have *3*: ⊢ *skip* ; #*True* = *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiMoreEqvMore*:
⊢ *di more* = *more*
proof −
have *1*: ⊢ *di* (○ #*True*) = ○(*di* #*True*) **by** (*rule DiNext*)
have *2*: ⊢ ○(*di* #*True*) ⟶ *more* **by** (*auto simp*: *next-defs di-defs more-defs sum.case-eq-if*)
have *3*: ⊢ *di*(○ #*True*) ⟶ *more* **using** *1 2* **by** *fastforce*
hence *4*: ⊢ *di more* ⟶ *more* **by** (*simp add*: *more-d-def*)
have *5*: ⊢ *more* ⟶ *di more* **by** (*rule ImpDi*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *DiIfEqvRule*:
assumes ⊢ *f* = *if*$_i$ (*init w*) *then g else h*
shows ⊢ *di f* = *if*$_i$ (*init w*) *then* (*di g*) *else* (*di h*)
proof −
have *1*: ⊢ *f* = *if*$_i$ (*init w*) *then g else h* **using** *assms* **by** *auto*
hence *2*: ⊢ *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*:
⊢ *di empty*
proof −
have *1*: ⊢ #*True* **by** *auto*
have *2*: ⊢ *empty* ; #*True* = #*True* **by** (*rule EmptyChop*)
have *3*: ⊢ *empty* ; #*True* **using** *1 2* **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

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

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

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

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

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

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

lemma *BaImpDist*:

⊢ *ba* (*f* ⟶ *g*) ⟶ *ba f* ⟶ *ba g*
proof −
have *1*: ⊢ *bi* (*f* ⟶ *g*) ⟶ (*bi f* ⟶ *bi g*) **by** (*rule BiImpDist*)
hence *2*: ⊢ □(*bi* (*f* ⟶ *g*) ⟶ (*bi f* ⟶ *bi g*)) **by** (*rule BoxGen*)
have *3*: ⊢ □(*bi* (*f* ⟶ *g*) ⟶ (*bi f* ⟶ *bi g*))
⟶
(□ (*bi* (*f* ⟶ *g*)) ⟶ (□(*bi f*) ⟶ □(*bi g*)))
by (*meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09*)
have *4*: ⊢ □(*bi* (*f* ⟶ *g*)) ⟶ (□(*bi f*) ⟶ □(*bi g*)) **using** *2 3 MP* **by** *fastforce*
have *5*: ⊢ *ba* (*f*⟶ *g*) = □(*bi* (*f* ⟶ *g*)) **by** (*rule BaEqvBtBi*)
have *6*: ⊢ *ba f* = □(*bi f*) **by** (*rule BaEqvBtBi*)
have *7*: ⊢ *ba g* = □(*bi g*) **by** (*rule BaEqvBtBi*)
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndEqv*:
⊢ *ba* (*f* ∧ *g*) = (*ba f* ∧ *ba g*)
proof −
have *1*: ⊢ *ba* (*f* ∧ *g*) = □(*bi* (*f* ∧ *g*))
by (*rule BaEqvBtBi*)
have *2*: ⊢ *bi* (*f* ∧ *g*) = (*bi f* ∧ *bi g*)
by (*auto simp*: *bi-defs sum.case-eq-if*)
hence *3*: ⊢ □(*bi* (*f* ∧ *g*)) = □(*bi f* ∧ *bi g*)
using *BoxEqvBox* **by** *blast*
have *4*: ⊢ □(*bi f* ∧ *bi g*)= (□(*bi f*) ∧ □(*bi g*))
by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
have *5*: ⊢*ba f* = □(*bi f*)
by (*rule BaEqvBtBi*)
have *6*: ⊢ *ba g* = □(*bi g*)
by (*rule BaEqvBtBi*)
from *1 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBaEqvBa*:
⊢ *ba* (*f* = *g*) ⟶ (*ba f* = *ba g*)
proof −
have *1*: ⊢ *ba* (*f* ⟶ *g*) ⟶ *ba f* ⟶ *ba g* **by** (*rule BaImpDist*)
have *2*: ⊢ *ba* (*g* ⟶ *f*) ⟶ *ba g* ⟶ *ba f* **by** (*rule BaImpDist*)
have *3*: ⊢ *ba* (*f* = *g*) = *ba* ((*f* ⟶ *g*) ∧ (*g* ⟶ *f*)) **by** (*auto simp*: *ba-defs sum.case-eq-if*)
have *4*: ⊢ *ba* ((*f* ⟶ *g*) ∧ (*g* ⟶ *f*)) = (*ba*((*f* ⟶ *g*)) ∧ *ba*((*g* ⟶ *f*))) **by** (*rule BaAndEqv*)
have *5*: ⊢ ((*ba f* ⟶ *ba g*) ∧ (*ba g* ⟶ *ba f*)) = (*ba f* = *ba g*) **by** *auto*
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBa*:
assumes ⊢ *f* ⟶ *g*
shows ⊢*ba f* ⟶ *ba g*
using *BaGen BaImpDist MP assms* **by** *metis*

lemma *BaEqvBa*:

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

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

lemma *DiamondEqvDiamondDiamond*:
$\vdash \Diamond\ f = \Diamond\ (\Diamond\ f)$
proof −
have *1*: $\vdash \Diamond\ (\Diamond\ f) = finite;(finite;f)$
by (*simp add*: *sometimes-d-def*)
have *2*: $\vdash finite;(finite;f)\ \ = (finite;finite);f$
by (*rule ChopAssoc*)
have *3*: $\vdash (finite;finite);f\ = finite;f$
by (*simp add*: *LeftChopEqvChop FiniteChopFiniteEqvFinite*)
have *4*: $\vdash finite;f\ \ = \Diamond f$
by (*simp add*: *sometimes-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DaEqvDaDa*:
$\vdash\ \ \ da\ \ f = \ da\ (\ \ da\ \ f)$
proof −
have *1*: $\vdash\ \ da\ \ f = \Diamond(\ di\ \ f)$
by (*rule DaEqvDtDi*)
have *2*: $\vdash\ \ di\ \ f = \ (di\ (\ di\ \ f))$
by (*rule DiEqvDiDi*)
hence *3*: $\vdash \Diamond\ (\ di\ \ f) = \Diamond\ (di\ \ (di\ \ f))$
by (*rule DiamondEqvDiamond*)
have *4*: $\vdash \Diamond\ \ (di\ \ f) = \Diamond(\Diamond\ (di\ \ (di\ \ f)))$
using *DiamondEqvDiamondDiamond DiEqvDiDi* **using** *3* **by** *fastforce*
have *5*: $\vdash \Diamond\ \ (di\ \ (di\ \ f)) = \ di\ (\Diamond\ \ (di\ \ f))$
by (*rule DtDiEqvDiDt*)
hence *6*: $\vdash \Diamond(\Diamond\ \ (di\ \ (di\ \ f))) = \Diamond\ (di\ (\Diamond\ \ (di\ \ f)))$
by (*rule DiamondEqvDiamond*)
have *7*: $\vdash\ \ da\ \ f = \Diamond\ (di(\ \Diamond\ (\ di\ \ f)))$
using *1 3 4 6* **by** *fastforce*
have *8*: $\vdash\ \ da\ (\Diamond\ (\ di\ \ f)) = \Diamond(\ di\ (\Diamond\ \ (di\ \ f)))$
by (*rule DaEqvDtDi*)
have *9*: $\vdash\ \ da\ (\ da\ \ f) = \ da\ (\Diamond\ \ (di\ \ f))$
using *1* **by** (*rule DaEqvDa*)
from *7 8 9* **show** *?thesis* **by** *fastforce*
qed

lemma *BaEqvBaBa*:
$\vdash\ \ \ ba\ \ f = \ ba\ \ (ba\ \ f)$

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

lemma *BaLeftChopImpChop*:
⊢ *ba* (*f* ⟶ *f1*) ⟶ *f*; *g* ⟶ *f1*; *g*
proof −
have *1*: ⊢ *ba* (*f* ⟶ *f1*) ⟶ *bi* (*f* ⟶ *f1*) **by** (*rule BaImpBi*)
have *2*: ⊢ *bi* (*f* ⟶ *f1*) ⟶ *f*; *g* ⟶ *f1*; *g* **by** (*rule BiChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaRightChopImpChop*:
⊢ *ba* (*g* ⟶ *g1*) ⟶ *f*; *g* ⟶ *f*; *g1*
proof −
have *1*: ⊢ *ba* (*g* ⟶ *g1*) ⟶ □(*g* ⟶ *g1*) **by** (*rule BaImpBt*)
have *2*: ⊢ □(*g* ⟶ *g1*) ⟶ *f*; *g* ⟶ *f*; *g1* **by** (*rule BoxChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopAndBaImport*:
⊢ (*f*; *f1*) ∧ *ba* *g* ⟶ (*f* ∧ *g*); (*f1* ∧ *g*)
proof −
have *1*: ⊢ *ba* *g* ∧ (*f*; *f1*) ⟶ (*g* ∧ *f*); (*g* ∧ *f1*) **by** (*rule BaAndChopImport*)
have *2*: ⊢ (*g* ∧ *f*); (*g* ∧ *f1*) = (*f* ∧ *g*); (*f1* ∧ *g*) **by** (*rule AndChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndChopImportA*:
⊢ *ba f* ∧ *g*;*g1* ⟶ (*f* ∧ *g*);*g1*
by (*meson BaAndChopImport ChopAndB lift-imp-trans*)

lemma *BaAndChopImportB*:
⊢ *ba f* ∧ *g*;*g1* ⟶ (*f* ∧ *g*);(*ba f* ∧ *g1*)
proof −
have *1*: ⊢ *ba f* = *ba* (*ba f*)
by (*simp add*: *BaEqvBaBa*)
have *2*: ⊢ *ba* (*ba f*) ∧ *g*;*g1* ⟶ *g*;(*ba f* ∧ *g1*)
by (*metis AndChopB BaAndChopImport lift-imp-trans*)
have *3*: ⊢ *ba f* ∧ *g*;(*ba f* ∧ *g1*) ⟶ (*f* ∧ *g*);(*ba f* ∧ *g1*)
by (*simp add*: *BaAndChopImportA*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBaImpBaAnd*:

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

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

lemma *DiNotBaImpNotBa*:
$\vdash$ *di* ($\neg$ (*ba f*)) $\longrightarrow$ $\neg$ (*ba f*)
proof −
have *1*: $\vdash$ *ba f* = *ba*(*ba f*) **by** (*rule BaEqvBaBa*)
have *2*: $\vdash$ *ba* (*ba f*) $\longrightarrow$ *bi* (*ba f*) **by** (*rule BaImpBi*)
have *3*: $\vdash$ *ba f* $\longrightarrow$ *bi* (*ba f*) **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *ba f* $\longrightarrow$ $\neg$ (*di* ($\neg$ (*ba f*))) **by** (*simp add*: *bi-d-def*)
from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotBaChopImpNotBa*:
$\vdash$ ($\neg$ (*ba f*)); *g* $\longrightarrow$ $\neg$ (*ba f*)
proof −
have *1*: $\vdash$ ($\neg$ (*ba f*)); *g* $\longrightarrow$ *di* ($\neg$ (*ba f*)) **by** (*rule ChopImpDi*)
have *2*: $\vdash$ *di* ($\neg$ (*ba f*)) $\longrightarrow$ $\neg$ (*ba f*) **by** (*rule DiNotBaImpNotBa*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *DiamondFinImpFin*:
$\vdash$ $\diamond$ (*fin f*) $\longrightarrow$ *fin f*
proof −
have *1*: $\vdash$ *fin f* = #*True*;(*f* $\wedge$ *empty*)
by (*rule FinEqvTrueChopAndEmpty*)
hence *2*: $\vdash$ $\diamond$ (*fin f*) = *finite*;(#*True*;(*f* $\wedge$ *empty*))
by (*metis FiniteChopFiniteEqvFinite LeftChopEqvChop inteq-reflection sometimes-d-def*)
have *3*: $\vdash$ *finite*;(#*True*;(*f* $\wedge$ *empty*)) = (*finite*;#*True*);(*f* $\wedge$ *empty*)
by (*rule ChopAssoc*)
have *4*: $\vdash$ (*finite*;#*True*);(*f* $\wedge$ *empty*) $\longrightarrow$ #*True*;(*f* $\wedge$ *empty*)
using *1 2 3 DiamondFin* **by** *fastforce*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopFinImpFin*:

$\vdash$ $(f \wedge finite);$ fin $(init\ w) \longrightarrow$ fin $(init\ w)$
proof −
have *1*: $\vdash (f \wedge finite);$ fin $(init\ w) \longrightarrow \Diamond$ (fin $(init\ w)$) **by** (*rule FiniteChopImpDiamond*)
have *2*: $\vdash \Diamond$ $(fin$ $(init\ w)) \longrightarrow$ fin $(init\ w)$ **by** (*rule DiamondFinImpFin*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FiniteRightChopEqvChop*:
assumes $\vdash finite \longrightarrow g = g1$
shows $\vdash finite \longrightarrow f;g = f;g1$
using *assms* **by** (*auto simp add*: *Valid-def finite-defs chop-defs sum.case-eq-if*)

lemma *FinImpYieldsFin*:
$\vdash$ fin $(init\ w) \wedge finite \longrightarrow (f \wedge finite)$ *yields* $(fin$ $(init\ w) \wedge finite)$
proof −
have *1*: $\vdash (f \wedge finite);$ $(fin$ $(init\ (\neg\ w)) \wedge finite) \longrightarrow$ $(fin$ $(init\ (\neg\ w)) \wedge finite)$
by (*metis* (*no-types*, *lifting*) *ChopAndB FiniteChopEqvDiamond FiniteChopFinExportA FiniteChopFiniteEqvFinite FiniteChopImpDiamond Prop12 inteq-reflection lift-and-com lift-imp-trans*)
have *2*: $\vdash finite \longrightarrow (\neg$ (fin $(init\ w) \wedge finite)) = (fin$ $(init\ (\neg\ w)) \wedge finite)$
using *FinNotStateEqvNotFinState* **by** *fastforce*
hence *3*: $\vdash$ $finite \longrightarrow (f \wedge finite); (\neg$ (fin $(init\ w) \wedge finite)) =$
$(f \wedge finite);$ (fin $(init\ (\neg\ w)) \wedge finite)$
using *FiniteRightChopEqvChop*[*of LIFT*$(\neg$ (fin $(init\ w) \wedge finite))$
LIFT$(fin$ $(init\ (\neg\ w)) \wedge finite)$ *LIFT*$(f \wedge finite)$]
by *blast*
have *4*: $\vdash (f \wedge finite); (\neg$ (fin $(init\ w) \wedge finite)) \longrightarrow (\neg$ (fin $(init\ w) \wedge finite))$
using *1 2 3* **by** *fastforce*
hence *5*: $\vdash$ fin $(init\ w) \wedge finite \longrightarrow \neg$ $((f \wedge finite); (\neg$ (fin $(init\ w) \wedge finite)))$
by *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ChopAndFin*:
$\vdash$ $((f; g) \wedge$ $(fin$ $(init\ w) \wedge finite)) = (f \wedge finite); (g \wedge$ $(fin$ $(init\ w) \wedge finite))$
proof −
have *1*: $\vdash$ fin $(init\ w) \wedge finite \longrightarrow (f \wedge finite)$ *yields* (fin $(init\ w) \wedge finite)$
by (*rule FinImpYieldsFin*)
have *10*: $\vdash ((f; g) \wedge$ $(fin$ $(init\ w) \wedge finite)) =$
$(((f \wedge finite); (g \wedge finite)) \wedge$ fin $(init\ w) \wedge finite)$
using *ChopAndFiniteDist*[*of f g*] **by** *auto*
have *2*: $\vdash (f; g) \wedge$ $(fin$ $(init\ w) \wedge finite) \longrightarrow$
$((f \wedge finite); (g \wedge finite)) \wedge (f \wedge finite)$ *yields* (fin $(init\ w) \wedge finite)$
using *1 10* **by** *fastforce*
have *3*: $\vdash ((f \wedge finite); (g \wedge finite)) \wedge (f \wedge finite)$ *yields* (fin $(init\ w) \wedge finite) \longrightarrow$
$(f \wedge finite); ((g \wedge finite) \wedge$ $(fin$ $(init\ w) \wedge finite))$
using *ChopAndYieldsImp* **by** *blast*
have *30*: $\vdash ((g \wedge finite) \wedge$ $(fin$ $(init\ w) \wedge finite)) = (g \wedge fin$ $(init\ w) \wedge finite)$

by *auto*
have *4*: ⊢ (*f*; *g*) ∧ (*fin* (*init w*) ∧ *finite*)⟶ (*f* ∧ *finite*); (*g* ∧ *fin* (*init w*) ∧ *finite*)
using *2 3 30*
by (*metis* (*mono-tags*, *lifting*) *inteq-reflection lift-imp-trans*)
have *11*: ⊢ (*f* ∧ *finite*); (*g* ∧ *fin* (*init w*) ∧ *finite*) ⟶ (*f* ∧ *finite*); (*g* ∧ *finite*)
using *ChopAndA* **by** (*metis 30 inteq-reflection*)
have *12*: ⊢ (*f* ∧ *finite*); (*g* ∧ *fin* (*init w*) ∧ *finite*) ⟶
(*f* ∧ *finite*); (*fin* (*init w*) ∧ *finite*)
by (*rule ChopAndB*)
have *13*: ⊢ (*f* ∧ *finite*); (*fin* (*init w*) ∧ *finite*)⟶ ◇ (*fin* (*init w*) ∧ *finite*)
using *FiniteChopImpDiamond* **by** *blast*
have *14*: ⊢ ◇(*fin* (*init w*) ∧ *finite*) ⟶ *fin* (*init w*)
by (*metis ChopAndA DiamondFin inteq-reflection sometimes-d-def*)
have *15*: ⊢ (*f* ∧ *finite*); (*g* ∧ *fin* (*init w*) ∧ *finite*) ⟶
((*f* ∧ *finite*); (*g* ∧ *finite*)) ∧ *fin* (*init w*)
using *11 12 13 14* **by** *fastforce*
from *4 15* **show** *?thesis* **by** (*metis ChopAndFiniteDist Prop12 int-iffI inteq-reflection*)
qed

lemma *ChopAndNotFin*:
⊢ (*f*; *g* ∧ ¬ (*fin* (*init w*)) ∧ *finite*) = (*f* ∧ *finite*); (*g* ∧ ¬ (*fin* (*init w*)) ∧ *finite*)
proof −
have *1*: ⊢ (*f*; *g* ∧ *fin* (*init* (¬ *w*)) ∧ *finite*) =
(*f* ∧ *finite*); (*g* ∧ *fin* (*init* (¬ *w*)) ∧ *finite*)
by (*rule ChopAndFin*)
have *2*: ⊢ (*fin* (*init* (¬ *w*)) ∧ *finite*)= ((¬ (*fin* (*init w*))) ∧ *finite*)
using *FinNotStateEqvNotFinState* **by** *fastforce*
hence *3*: ⊢ (*g* ∧ *fin* (*init* (¬ *w*)) ∧ *finite*) = (*g* ∧ ¬(*fin* (*init w*)) ∧ *finite*)
by *auto*
hence *4*: ⊢ (*f* ∧ *finite*); (*g* ∧ *fin* (*init* (¬ *w*)) ∧ *finite*) =
(*f* ∧ *finite*); (*g* ∧ ¬ (*fin* (*init w*)) ∧ *finite*)
by (*rule RightChopEqvChop*)
from *1 2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *FinChopChain*:
⊢ (((*init w*) ∧ *finite* ⟶ *fin* (*init w1*)));
(((*init w1*) ∧ *finite* ⟶ *fin* (*init w2*)))
∧ *finite*
⟶ (((*init w*) ∧ *finite* ⟶ *fin* (*init w2*)))
proof −
have *1*: ⊢ (*init w*) ∧ *finite* ∧
((*init w*) ∧ *finite* ⟶ *fin* (*init w1*)); ((*init w1*) ∧ *finite* ⟶ *fin* (*init w2*))
⟶
((*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *fin* (*init w1*)));
(((*init w1*) ∧ *finite*⟶ *fin* (*init w2*)) ∧ *finite*)
by (*metis* (*no-types*, *lifting*) *ChopAndFiniteDist StateAndChop int-iffD2*
inteq-reflection lift-and-com)
have *2*: ⊢ (*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *fin*(*init w1*)) ⟶ *fin* (*init w1*) ∧ *finite*

by *auto*
have *3*: ⊢ ((*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *fin* (*init w1*)));
(((*init w1*) ∧ *finite* ⟶ *fin* (*init w2*)) ∧ *finite*)
⟶
(*fin* (*init w1*) ∧ *finite*); (((*init w1*) ∧ *finite* ⟶ *fin* (*init w2*)) ∧ *finite*)
using *2 LeftChopImpChop* **by** *blast*
have *4*: ⊢ (*fin* (*init w1*) ∧ *finite*); (((*init w1*) ∧ *finite* ⟶ *fin* (*init w2*)) ∧ *finite*) =
◇((*init w1*) ∧ ((*init w1*) ∧ *finite* ⟶ *fin* (*init w2*)) ∧ *finite*)
using *FinChopEqvDiamond* **by** *blast*
have *41*: ⊢ ((*init w1*) ∧ *finite* ∧ ((*init w1*) ∧ *finite* ⟶ *fin* (*init w2*))) ⟶ *fin* (*init w2*)
by *auto*
have *42*: ⊢ ◇((*init w1*) ∧ *finite* ∧ ((*init w1*) ∧ *finite* ⟶ *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*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *fin* (*init w1*));
((*init w1*) ∧ *finite* ⟶ *fin* (*init w2*))
⟶ *fin* (*init w2*)
using *1 3 4 5 42*
by (*metis* (*no-types*, *hide-lams*) *inteq-reflection lift-and-com lift-imp-trans*)
from *6* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopRule*:
assumes ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ *fin* (*init w1*)
⊢ (*init w1*)∧ *f1* ∧ *finite* ⟶ *fin* (*init w2*)
shows ⊢ (*init w*) ∧ (*f*; *f1*) ∧ *finite* ⟶ *fin* (*init w2*)
proof −
have *1*: ⊢ (*init w*) ∧ (*f*; *f1*) ∧ *finite* ⟶ ((*init w*) ∧ *f* ∧ *finite*); (*f1* ∧ *finite*)
using *StateAndChopImport*
by (*metis ChopAndFiniteDist inteq-reflection*)
have *2*: ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ *fin* (*init w1*) ∧ *finite* **using** *assms* **by** *auto*
hence *3*: ⊢ ((*init w*) ∧ *f* ∧ *finite*); (*f1* ∧ *finite*) ⟶ (*fin* (*init w1*) ∧ *finite*); (*f1* ∧ *finite*)
by (*rule LeftChopImpChop*)
have *4*: ⊢ (*fin* (*init w1*) ∧ *finite*); (*f1* ∧ *finite*) = ◇((*init w1*) ∧ *f1* ∧ *finite*)
by (*rule FinChopEqvDiamond*)
have *5*: ⊢ (*init w1*) ∧ *f1* ∧ *finite* ⟶ *fin* (*init w2*) **using** *assms* **by** *auto*
hence *6*: ⊢ ◇((*init w1*) ∧ *f1* ∧ *finite*) ⟶ ◇ (*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* ∧ *finite* ⟶ *f1* ∧ *fin* (*init w1*)
⊢ (*init w1*) ∧ *g* ∧ *finite* ⟶ *g1*
shows ⊢ (*init w*) ∧ (*f*; *g*) ∧ *finite* ⟶ ((*f1* ∧ *finite*); *g1*)
proof −
have *1*: ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ ((*f1* ∧ *finite*) ∧ *fin* (*init w1*)) **using** *assms* **by** *auto*
hence *2*: ⊢ (*init w*) ∧ ((*f* ∧ *finite*); (*g* ∧ *finite*)) ⟶

((*f1* ∧ *finite*)∧ *fin* (*init w1*)); (*g* ∧ *finite*)
using *StateAndChopImpChopRule* **by** *blast*
have *3*: ⊢ ((*f1* ∧ *finite*) ∧ *fin* (*init w1*)); (*g* ∧ *finite*) =
(*f1* ∧ *finite*); ((*init w1*) ∧ (*g* ∧ *finite*))
using *AndFinChopEqvStateAndChop* **by** *blast*
have *4*: ⊢ (*init w1*)∧ *g* ∧ *finite* ⟶ *g1* **using** *assms* **by** *auto*
hence *5*: ⊢ (*f1* ∧ *finite*); ((*init w1*) ∧ *g* ∧ *finite*) ⟶ (*f1* ∧ *finite*); *g1*
using *RightChopImpChop* **by** *blast*
from *2 3 5* **show** *?thesis* **using** *ChopAndFiniteDist* **by** *fastforce*
qed

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

lemma *TrueChopMoreEqvMore*:
⊢ #*True* ; *more* = *more*
by (*metis ChopMoreImpMore EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteChopMoreEqvMore LeftChopImpChop Prop09 int-eq-true int-iffI inteq-reflection*)

lemma *FiniteChopFmoreEqvFmore*:
⊢ *finite*;*fmore* = *fmore*
by (*metis TrueChopAndFiniteEqvAndFiniteChopFinite TrueChopMoreEqvMore fmore-d-def inteq-reflection*)

lemma *MoreChopLoop*:
assumes ⊢ *f* ⟶ *fmore* ; *f*
shows ⊢ *finite* ⟶ ¬ *f*
proof −
have *1*: ⊢ *f* ⟶ *fmore* ; *f*
using *assms* **by** *auto*
hence *11*: ⊢ ◇ (*f*) ⟶ ◇ (*fmore*;*f*)
using *DiamondImpDiamond* **by** *blast*
have *12*: ⊢ ◇ (*fmore*;*f*) = *finite*;(*fmore*;*f*)
by (*simp add*: *sometimes-d-def*)
have *13*: ⊢ *finite*;(*fmore*;*f*) = (*finite*;*fmore*);*f*
by (*rule ChopAssoc*)
have *14*: ⊢ ◇ (*fmore*;*f*) = *fmore*;*f*

using *FiniteChopFmoreEqvFmore 12 13* **by** (*metis int-eq*)
have *2*: $\vdash$ *fmore* ; *f* = $\bigcirc$($\Diamond$ *f*)
using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash$ $\Diamond$ (*f*) $\longrightarrow$ $\bigcirc$($\Diamond$ *f*)
using *11 14 2* **by** *fastforce*
hence *4*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ($\Diamond$ *f*)
using *NextLoop* **by** *blast*
have *5*: $\vdash$ $\neg$ ($\Diamond$ *f*) $\longrightarrow$ $\neg$ *f*
using *NowImpDiamond* **by** *fastforce*
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *MoreChopContra*:
assumes $\vdash$ *f* $\wedge$ $\neg$ *g* $\longrightarrow$ (*fmore* ; (*f* $\wedge$ $\neg$ *g*))
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *g*
proof −
have *1*: $\vdash$ *f* $\wedge$ $\neg$ *g* $\longrightarrow$ (*fmore* ; (*f* $\wedge$ $\neg$ *g*)) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *finite* $\longrightarrow$ $\neg$ (*f* $\wedge$ $\neg$ *g*) **by** (*rule MoreChopLoop*)
from *2* **show** *?thesis*
by (*simp add*: *Valid-def finite-defs infinite-defs sum.case-eq-if*)
qed

lemma *MoreChopLoopFinite*:
assumes $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *fmore* ; *f*
shows $\vdash$ *finite* $\longrightarrow$ $\neg$ *f*
proof −
have *1*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *fmore* ; *f*
using *assms* **by** *auto*
hence *11*: $\vdash$ $\Diamond$ (*f* $\wedge$ *finite*) $\longrightarrow$ $\Diamond$ (*fmore*;*f*)
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash$ $\Diamond$ (*fmore*;*f*) = *finite*;(*fmore*;*f*)
by (*simp add*: *sometimes-d-def*)
have *13*: $\vdash$ *finite*;(*fmore*;*f*) = (*finite*;*fmore*);*f*
by (*rule ChopAssoc*)
have *14*: $\vdash$ $\Diamond$ (*fmore*;*f*) = *fmore*;*f*
using *FiniteChopFmoreEqvFmore 12 13* **by** (*metis int-eq*)
have *2*: $\vdash$ *fmore* ; *f* = $\bigcirc$($\Diamond$ *f*)
using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash$ $\Diamond$ (*f* $\wedge$ *finite*) $\longrightarrow$ $\bigcirc$($\Diamond$ *f*)
using *11 14 2* **by** *fastforce*
have *31*: $\vdash$ $\Diamond$ (*f* $\wedge$ *finite*) = (($\Diamond$ *f*) $\wedge$ *finite*)
by (*metis* (*no-types*, *lifting*) *3 ChopAndB ChopAndNotChopImp DiamondDiamondEqvDiamond DiamondIntroC FiniteChopFiniteEqvFinite FiniteChopInfEqvInf Prop11 Prop12 finite-d-def inteq-reflection sometimes-d-def*)
have *32*: $\vdash$ ($\Diamond$ *f*) $\wedge$ *finite* $\longrightarrow$ $\bigcirc$($\Diamond$ *f*)
using *3 31* **by** *fastforce*
hence *4*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ($\Diamond$ *f*)
by (*metis* (*no-types*, *lifting*) *DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09 finite-d-def int-simps*(*15*) *int-simps*(*32*) *inteq-reflection sometimes-d-def*)
have *5*: $\vdash$ $\neg$ ($\Diamond$ *f*) $\longrightarrow$ $\neg$ *f*

by (*simp add*: *NowImpDiamond*)
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *MoreChopEqvFmoreOrInf*:
$\vdash$ *more* ; *f* = ((*fmore*;*f*) $\vee$ *inf*)
by (*simp add*: *Valid-def more-defs fmore-defs chop-defs infinite-defs sum.case-eq-if*)

lemma *MoreChopLoopFiniteB*:
assumes $\vdash$ *f* $\longrightarrow$ *more* ; *f*
shows $\vdash$ *finite* $\longrightarrow$ $\neg$ *f*
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *more* ; *f*
using *assms* **by** *auto*
have *10*: $\vdash$ *f* $\longrightarrow$ (*fmore*;*f*) $\vee$ *inf*
using *MoreChopEqvFmoreOrInf assms* **by** *fastforce*
hence *100*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ (*fmore*;*f*)
by (*simp add*: *Prop13 finite-d-def*)
hence *11*: $\vdash$ $\diamond$ (*f* $\wedge$ *finite*) $\longrightarrow$ $\diamond$ (*fmore*;*f*)
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash$ $\diamond$ (*fmore*;*f*) = *finite*;(*fmore*;*f*)
by (*simp add*: *sometimes-d-def*)
have *13*: $\vdash$ *finite*;(*fmore*;*f*) = (*finite*;*fmore*);*f*
by (*rule ChopAssoc*)
have *14*: $\vdash$ $\diamond$ (*fmore*;*f*) = *fmore*;*f*
using *FiniteChopFmoreEqvFmore 12 13* **by** (*metis int-eq*)
have *2*: $\vdash$ *fmore* ; *f* = $\bigcirc$($\diamond$ *f*)
using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash$ $\diamond$ (*f* $\wedge$ *finite*) $\longrightarrow$ $\bigcirc$($\diamond$ *f*)
using *11 14 2* **by** *fastforce*
have *31*: $\vdash$ $\diamond$ (*f* $\wedge$ *finite*) = (($\diamond$ *f*) $\wedge$ *finite*)
by (*metis* (*no-types*, *hide-lams*) *ChopAndA ChopAndB ChopAndNotChopImp FiniteChopFiniteEqvFinite FiniteChopInfEqvInf Prop11 Prop12 finite-d-def inteq-reflection sometimes-d-def*)
have *32*: $\vdash$ ($\diamond$ *f*) $\wedge$ *finite* $\longrightarrow$ $\bigcirc$($\diamond$ *f*)
using *3 31* **by** *fastforce*
hence *4*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ($\diamond$ *f*)
by (*metis* (*no-types*, *lifting*) *DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09 finite-d-def int-simps*(*15*) *int-simps*(*32*) *inteq-reflection sometimes-d-def*)
have *5*: $\vdash$ $\neg$ ($\diamond$ *f*) $\longrightarrow$ $\neg$ *f*
by (*simp add*: *NowImpDiamond*)
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

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

hence *2*: ⊢ *finite* ⟶ ¬ (*f* ∧ ¬ *g*) **using** *MoreChopLoopFinite* **by** (*simp add*: *MoreChopLoopFinite*)
from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *MoreChopContraFiniteB*:
assumes ⊢ (*f* ∧ ¬ *g*) ⟶ (*more* ; (*f* ∧ ¬ *g*))
shows ⊢ *f* ∧ *finite* ⟶ *g*
proof −
have *1*: ⊢ (*f* ∧ ¬ *g*) ⟶ (*more* ; (*f* ∧ ¬ *g*)) **using** *assms* **by** *auto*
hence *2*: ⊢ *finite* ⟶ ¬ (*f* ∧ ¬ *g*) **using** *MoreChopLoopFinite* **by** (*simp add*: *MoreChopLoopFiniteB*)
from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

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

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

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

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

13.8 Properties of Chopstar and Chopplus

lemma *FPowerstardef*:
$\vdash fpowerstar\ f = (\exists\ n.\ power\ f\ n)$
by (*simp add*: *fpowerstar-d-def*)

lemma *Powerstardef*:
$\vdash powerstar\ f = (fpowerstar\ f);(empty \vee (f \wedge inf))$
by (*simp add*: *fpowerstar-d-def powerstar-d-def*)

lemma *Chopstardef*:
$\vdash chopstar\ f = powerstar\ (f \wedge more)$
by (*simp add*: *chopstar-d-def*)

lemma *AndEmptyChopAndEmptyEqvAndEmpty*:
$\vdash (f \wedge empty);(f \wedge empty) = (f \wedge empty)$
by (*simp add*: *Valid-def empty-defs chop-defs sum.case-eq-if min.absorb1*)
(*metis interval-prefix-intlen interval-suffix-zero le-antisym le-numeral-extra*(*3*) *min.absorb1*
sum.collapse(*1*))

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

lemma *ChopInductL*:
assumes $\vdash g \vee f;h \longrightarrow h$
shows $\vdash (power\ f\ n);g \longrightarrow h$
proof

(*induct n*)
case *0*
then show *?case* **using** *EmptyChop assms*
by (*metis MP Prop12 int-eq int-iffD1 int-simps*(*33*) *pow-0*)
next
case (*Suc n*)
then show *?case* **using** *assms*
by (*metis AndChopA ChopAssoc Prop05 Prop11 RightChopImpChop lift-imp-trans pow-Suc*)
qed

lemma *ChopInductFiniteL*:
assumes $\vdash$ *g* $\vee$ (*f* $\wedge$ *finite*);*h* $\longrightarrow$ *h*
shows $\vdash$ (*power f n*);*g* $\longrightarrow$ *h*
proof
(*induct n*)
case *0*
then show *?case* **using** *EmptyChop assms*
by (*metis MP Prop12 int-eq int-iffD1 int-simps*(*33*) *pow-0*)
next
case (*Suc n*)
then show *?case* **using** *assms*
by (*metis ChopAndB ChopAssoc Prop05 Prop10 inteq-reflection lift-imp-trans pow-Suc*)
qed

lemma *ChopInductFiniteMoreL*:
assumes $\vdash$ *g* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);*h* $\longrightarrow$ *h*
shows $\vdash$ (*power f n*);*g* $\longrightarrow$ *h*
proof
(*induct n*)
case *0*
then show *?case* **using** *assms* **by** (*metis ChopInductFiniteL pow-0*)
next
case (*Suc n*)
then show *?case*
proof $-$
have *1*: $\vdash$ *power f* (*Suc n*);*g* = ((*f* $\wedge$ *finite*);*power f n*);*g*
by *simp*
have *2*: $\vdash$ ((*f* $\wedge$ *finite*);*power f n*);*g* = (*f* $\wedge$ *finite*);((*power f n*);*g*)
by (*meson ChopAssoc Prop11*)
have *3*: $\vdash$ (*f* $\wedge$ *finite*);((*power f n*);*g*) $\longrightarrow$ (*f* $\wedge$ *finite*);*h*
by (*simp add*: *RightChopImpChop Suc*.*hyps*)
have *4*: $\vdash$ (*f* $\wedge$ *finite*);*h* = (((*f* $\wedge$ *more*) $\wedge$ *finite*); *h* $\vee$ ((*f* $\wedge$ *empty*) $\wedge$ *finite*);*h*)
using *neq0-conv*
by (*simp add*: *Valid-def finite-defs chop-defs more-defs empty-defs sum*.*case-eq-if*)
fastforce
have *5*: $\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); *h* $\longrightarrow$ *h* **using** *assms* **by** *auto*
have *6*: $\vdash$ ((*f* $\wedge$ *empty*) $\wedge$ *finite*); *h* $\longrightarrow$ *h*
by (*metis AndChopA AndChopB EmptyChop inteq-reflection lift-imp-trans*)
from *5 6 4 3 2 1* **show** *?thesis* **by** *fastforce*
qed

qed

lemma *ChopInductInfL*:
assumes $\vdash g \vee f;h \longrightarrow h$
shows $\vdash ((power\ f\ n);(f \wedge inf));g \longrightarrow h$
proof
(*induct n*)
case *0*
then show *?case* **using** *assms*
by (*metis* (*no-types*, *lifting*) *AndInfChopEqvAndInf ChopAssoc ChopInductFiniteL PowerstarEqvSemhelp3 Prop03 Prop10 Prop12 inteq-reflection*)
next
case (*Suc n*)
then show *?case* **using** *assms*
by (*metis AndChopA ChopAndB ChopAssoc Prop03 Prop10 int-eq lift-imp-trans pow-Suc*)
qed

lemma *ChopInductInfMoreL*:
assumes $\vdash g \vee f;h \longrightarrow h$
shows $\vdash ((power\ f\ n);((f \wedge more) \wedge inf));g \longrightarrow h$
using *ChopInductInfL*
by (*metis AndMoreAndInfEqvAndInf assms inteq-reflection*)

lemma *ChopInductR*:
assumes $\vdash g \vee h;f \longrightarrow h$
shows $\vdash g;(power\ f\ n) \longrightarrow h$
proof
(*induct n*)
case *0*
then show *?case* **using** *ChopEmpty assms*
by (*metis MP Prop12 int-iffD2 int-simps*(*33*) *inteq-reflection pow-0*)
next
case (*Suc n*)
then show *?case* **using** *assms*
by (*metis* (*no-types*, *hide-lams*) *ChopAndA ChopAssoc LeftChopImpChop PowerCommute Prop05 int-eq lift-imp-trans pow-Suc*)
qed

lemma *ChopInductInfR*:
assumes $\vdash g \vee h;f \longrightarrow h$
shows $\vdash g;((power\ f\ n);(f \wedge inf)) \longrightarrow h$
using *assms*
by (*metis* (*no-types*, *hide-lams*) *AndChopA ChopAndB ChopAssoc ChopInductR Prop05 Prop10 inteq-reflection lift-and-com lift-imp-trans*)

lemma *ChopExistPower*:
$\vdash (g;(\exists\ n.\ power\ f\ n)) = (\exists\ n.\ g;power\ f\ n)$

using *ChopExist* **by** *fastforce*

lemma *ExistChopPower*:
$\vdash$ ($\exists$ *n*. (*power f n*);*g*) = ($\exists$ *n*. *power f n*);*g*
using *ExistChop* **by** *fastforce*

lemma *PowerStarCommute*:
$\vdash$ (*f* $\wedge$ *finite*);($\exists$ *n*. *power f n*) = ($\exists$ *n*. *power f n*);(*f* $\wedge$ *finite*)
proof −
 have *1*: $\vdash$ (*f* $\wedge$ *finite*) ;($\exists$ *n*. *power f n*) =
 ($\exists$ *n*. (*f* $\wedge$ *finite*) ;*power f n*)
 using *ChopExistPower* **by** *blast*
 have *2*: $\vdash$ ($\exists$ *n*. (*f* $\wedge$ *finite*) ;*power f n*) =
 ($\exists$ *n*. (*power f n*);(*f* $\wedge$ *finite*))
 using *PowerCommute* **by** *fastforce*
 have *3*: $\vdash$ ($\exists$ *n*. (*power f n*);(*f* $\wedge$ *finite*)) =
 ($\exists$ *n*. (*power f n*));(*f* $\wedge$ *finite*)
 using *ExistChopPower* **by** *blast*
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *PowerSucAndEmptyEqvAndEmpty*:
$\vdash$ (*power* (*f* $\wedge$ *empty*) (*Suc n*)) = (*f* $\wedge$ *empty*)
proof
 (*induct n*)
 case *0*
 then show *?case* **using** *ChopEmpty*
 by (*metis* (*no-types*, *lifting*) *FiniteAndEmptyEqvEmpty Prop10 Prop12 int-iffD1 inteq-reflection pow-0 pow-Suc*)
 next
 case (*Suc n*)
 then show *?case*
 by (*metis AndEmptyChopAndEmptyEqvAndEmpty EmptyImpFinite Prop10 Prop12 int-iffD1 inteq-reflection pow-Suc*)
qed

lemma *FiniteOr*:
$\vdash$ ((*f* $\vee$ *g*) $\wedge$ *finite*) = ((*f* $\wedge$ *finite*) $\vee$ (*g* $\wedge$ *finite*))
by *auto*

lemma *PowerOr*:
$\vdash$ (*power* (*f* $\vee$ *g*) (*Suc n*)) = (((*f* $\wedge$ *finite*);*power* (*f* $\vee$ *g*) *n*) $\vee$
 ((*g* $\wedge$ *finite*);*power* (*f* $\vee$ *g*) *n*))
 by (*simp add*: *FiniteOr OrChopEqvRule*)

lemma *PowerEmptyOrMore*:
$\vdash$ (*power* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) (*Suc n*)) =
 ((*f* $\wedge$ *empty*);(*power* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) *n*) $\vee$
 (*f* $\wedge$ *fmore*);(*power* ((*f* $\wedge$ *empty*) $\vee$ (*f* $\wedge$ *more*)) *n*))

proof −
have *f2*: $\vdash(f \wedge empty) = (empty \wedge f)$
  by (*meson lift-and-com*)
have *f3*: $\vdash((f \wedge empty) \wedge finite) = (finite \wedge f \wedge empty)$
  by (*meson lift-and-com*)
have *f4*: $\vdash(empty \wedge f) = (finite \wedge f \wedge empty)$
  using *FiniteAndEmptyEqvEmpty* **by** *auto*
have $\vdash((f \wedge more) \wedge finite) = (f \wedge fmore)$
  by (*meson AndMoreAndFiniteEqvAndFmore*)
then show *?thesis*
  using *f4 f3 f2* **by** (*metis PowerOr inteq-reflection*)
qed

lemma *PSEqvEmptyOrChopPS*:
$\vdash$ *powerstar f* = (*empty* $\vee$ *f*;*powerstar f*)
using *PowerstarEqvSem Valid-def* **by** *blast*

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

lemma *CSEqvOrChopCS*:
$\vdash\ f^\star = (empty \vee (f; f^\star))$
proof −
have *1*: $\vdash f^\star = (empty \vee (f \wedge more);f^\star)$ **by** (*rule ChopstarEqv*)
have *2*: $\vdash (f \wedge more);f^\star \longrightarrow f;f^\star$ **by** (*rule AndChopA*)
have *3*: $\vdash f^\star \longrightarrow empty \vee f; f^\star$ **using** *1 2* **by** (*metis int-iffD1 Prop08*)
have *4*: $\vdash empty \longrightarrow f^\star$ **by** (*rule EmptyImpCS*)
have *5*: $\vdash f \longrightarrow empty \vee (f \wedge more)$ **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*: $\vdash f^\star \longrightarrow empty \vee (f \wedge more);f^\star$ **using** *1* **by** *fastforce*
have *8*: $\vdash f; f^\star \longrightarrow empty \vee (f \wedge more); f^\star$ **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 *PowerChopCommute*:
$\vdash ((f \wedge more) \wedge finite);power\ (f \wedge more)\ n = power\ (f \wedge more)\ n;((f \wedge more) \wedge finite)$
using *PowerCommute* **by** *auto*

lemma *ChopExist*:

$\vdash (g;(\exists\, n.\ power\ (f \wedge more)\ n)) = (\exists\ n.\ g;power\ (f \wedge more)\ n)$
using *ChopExistPower* **by** *auto*

lemma *ExistChop*:
$\vdash (\exists\, n.\ (power\ (f \wedge more)\ n);g) = (\exists\, n.\ power\ (f \wedge more)\ n);g$
using *ExistChopPower* **by** *auto*

lemma *FPowerstarInductL*:
assumes $\vdash g \vee (f \wedge finite);h \longrightarrow h$
shows $\vdash (fpowerstar\ f);g \longrightarrow h$
proof −
have *1*: $\vdash (fpowerstar\ f);g = (\exists\ n.\ power\ f\ n);g$
by (*simp add*: *fpowerstar-d-def LeftChopEqvChop*)
have *2*: $\vdash (\exists\ n.\ power\ f\ n);g =$
$(\exists\ n.\ (power\ f\ n);g)$
using *ExistChopPower* **by** *fastforce*
have *3*: $\bigwedge n. \vdash (power\ f\ n);g \longrightarrow h$
using *ChopInductFiniteL assms* **by** *blast*
have *4*: $\vdash (\exists\ n.\ ((power\ f\ n));g) \longrightarrow h$
using *3* **by** (*simp add*: *Valid-def*) *blast*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *FPowerstarInductMoreL*:
assumes $\vdash g \vee ((f \wedge more) \wedge finite);h \longrightarrow h$
shows $\vdash (fpowerstar\ f);g \longrightarrow h$
proof −
have *1*: $\vdash (fpowerstar\ f);g = (\exists\ n.\ power\ f\ n);g$
by (*simp add*: *fpowerstar-d-def LeftChopEqvChop*)
have *2*: $\vdash (\exists\ n.\ power\ f\ n);g =$
$(\exists\ n.\ (power\ f\ n);g)$
using *ExistChopPower* **by** *fastforce*
have *3*: $\bigwedge n. \vdash (power\ f\ n);g \longrightarrow h$
using *ChopInductFiniteMoreL assms* **by** *blast*
have *4*: $\vdash (\exists\ n.\ ((power\ f\ n));g) \longrightarrow h$
using *3* **by** (*simp add*: *Valid-def*) *blast*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *PowerstarInductL*:
assumes $\vdash g \vee f;h \longrightarrow h$
shows $\vdash (powerstar\ f);g \longrightarrow h$
proof −
have *1*: $\vdash (powerstar\ f);g = ((\exists\ n.\ power\ f\ n);(empty \vee (f \wedge inf)));g$
by (*simp add*: *powerstar-d-def LeftChopEqvChop*)
have *11*: $\vdash ((\exists\ n.\ power\ f\ n);(empty \vee (f \wedge inf)));g =$
$(\exists\ n.\ power\ f\ n);((empty \vee (f \wedge inf));g)$
by (*meson ChopAssoc Prop11*)
have *2*: $\vdash (\exists\ n.\ power\ f\ n);((empty \vee (f \wedge inf));g) =$
$(\exists\ n.\ ((power\ f\ n);((empty \vee (f \wedge inf));g)))$

using *ExistChopPower* **by** *fastforce*
have *3*: $\bigwedge$ *n*. ⊢ (*power f n*);*g* ⟶ *h*
using *ChopInductL assms* **by** *blast*
have *31*:$\bigwedge$ *n*. ⊢ ((*power f n*);(*f* ∧ *inf*));*g* ⟶ *h*
using *ChopInductInfL assms* **by** *blast*
have *4*: ⊢ (∃ *n*. ((*power f n*);(*empty* ∨ (*f* ∧ *inf*)));*g*) ⟶ *h*
using *3 31*
by (*simp add*: *Valid-def*)
(*metis* (*mono-tags*, *lifting*) *ChopAssoc ChopOrEqv EmptyOrChopEqv int-eq intensional-rews*(*3*))
from *1 11 2 4* **show** *?thesis*
by (*metis InfiniteSemantics*.*ExistChop inteq-reflection*)
qed

lemma *ChopstarInductL*:
assumes ⊢ *g* ∨ *f*;*h* ⟶ *h*
shows ⊢ (*chopstar f*);*g* ⟶ *h*
proof −
have *1*: ⊢ (*chopstar f*);*g* = ((∃ *n*. *power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)));*g*
by (*simp add*: *chopstar-d-def powerstar-d-def LeftChopEqvChop*)
have *11*: ⊢ ((∃ *n*. *power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)));*g* =
(∃ *n*. *power* (*f* ∧ *more*) *n*);((*empty* ∨ ((*f* ∧ *more*) ∧ *inf*));*g*)
by (*meson ChopAssoc Prop11*)
have *2*: ⊢ (∃ *n*. *power* (*f* ∧ *more*) *n*);((*empty* ∨ ((*f* ∧ *more*) ∧ *inf*));*g*) =
(∃ *n*. (*power* (*f* ∧ *more*) *n*);(((*empty* ∨ ((*f* ∧ *more*) ∧ *inf*));*g*)))
using *ExistChopPower* **by** *fastforce*
have *21*: ⊢ *g* ∨ (*f* ∧ *more*);*h* ⟶ *h*
using *AndChopA Prop03 Prop10 assms int-simps*(*33*) *inteq-reflection* **by** *fastforce*
have *3*: $\bigwedge$ *n*. ⊢ (*power* (*f* ∧ *more*) *n*);*g* ⟶ *h*
using *21 ChopInductL*[*of g LIFT*(*f* ∧ *more*) *h*] *assms* **by** *auto*
have *31*: $\bigwedge$ *n*. ⊢ ((*power* (*f* ∧ *more*) *n*);((*f* ∧ *more*) ∧ *inf*));*g* ⟶ *h*
using *assms*
by (*metis* (*no-types*, *lifting*) *AndChopA ChopInductInfL Prop03 Prop10 int-eq-true int-simps*(*33*)
inteq-reflection lift-imp-trans)
have *41*: $\bigwedge$*n*. ⊢ ((*power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)));*g* ⟶ *h*
using *3 31*
by (*metis* (*mono-tags*, *lifting*) *ChopAssoc ChopOrEqv EmptyOrChopEqv Prop02 int-eq*)
have *4*: ⊢ (∃ *n*. ((*power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)));*g*) ⟶ *h*
using *41* **by** *fastforce*
from *1 11 2 4* **show** *?thesis*
by (*metis InfiniteSemantics*.*ExistChop inteq-reflection*)
qed

lemma *ChopstarInductMoreL*:
assumes ⊢ *g* ∨ (*f* ∧ *more*);*h* ⟶ *h*
shows ⊢ (*chopstar f*);*g* ⟶ *h*
proof −
have *1*: ⊢ (*chopstar f*);*g* = ((∃ *n*. *power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)));*g*
by (*simp add*: *chopstar-d-def powerstar-d-def LeftChopEqvChop*)
have *11*: ⊢ ((∃ *n*. *power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)));*g* =
(∃ *n*. *power* (*f* ∧ *more*) *n*);((*empty* ∨ ((*f* ∧ *more*) ∧ *inf*));*g*)

by (*meson ChopAssoc Prop11*)
have *2*: ⊢ (∃ *n*. *power* (*f* ∧ *more*) *n*);((*empty* ∨ ((*f* ∧ *more*) ∧ *inf*));*g*) =
(∃ *n*. (*power* (*f* ∧ *more*) *n*);((*empty* ∨ ((*f* ∧ *more*) ∧ *inf*));*g*))
using *ExistChopPower* **by** *fastforce*
have *3*: ⋀ *n*. ⊢ (*power* (*f* ∧ *more*) *n*);*g* ⟶ *h*
using *ChopInductL assms* **by** (*metis*)
have *31*: ⊢ (∃ *n*. (*power* (*f* ∧ *more*) *n*);*g*) ⟶ *h*
using *3* **by** *fastforce*
have *32*: ⋀ *n*. ⊢ ((*power* (*f* ∧ *more*) *n*);((*f* ∧ *more*) ∧ *inf*));*g* ⟶ *h*
using *assms*
by (*metis* (*no-types*, *lifting*) *AndChopA ChopInductInfL int-eq-true inteq-reflection*)
have *33*: ⊢ (∃ *n*. ((*power* (*f* ∧ *more*) *n*);((*f* ∧ *more*) ∧ *inf*));*g*) ⟶ *h*
using *32* **by** *fastforce*
have *34*: ⊢ (∃ *n*. (*power* (*f* ∧ *more*) *n*);*g*) ∨
(∃ *n*. ((*power* (*f* ∧ *more*) *n*);((*f* ∧ *more*) ∧ *inf*));*g*) ⟶ *h*
using *31 33* **by** *fastforce*
have *35*: ⊢ (∃ *n*. (*power* (*f* ∧ *more*) *n*);*g* ∨ ((*power* (*f* ∧ *more*) *n*);((*f* ∧ *more*) ∧ *inf*));*g*)
⟶ *h*
using *34* **by** *fastforce*
have *36*: ⋀ *n* . ⊢ ((*power* (*f* ∧ *more*) *n*);*g* ∨ ((*power* (*f* ∧ *more*) *n*);((*f* ∧ *more*) ∧ *inf*));*g*) =
((*power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)));*g*
by (*metis* (*no-types*, *lifting*) *ChopAssoc ChopOrEqv EmptyOrChopEqv inteq-reflection*)
have *4*: ⊢ (∃ *n*. ((*power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)));*g*) ⟶ *h*
using *36 35* **by** *fastforce*
from *1 11 2 4* **show** *?thesis*
by (*metis InfiniteSemantics.ExistChop inteq-reflection*)
qed

lemma *FPowerstarInductR*:
assumes ⊢ *g* ∨ *h*;*f* ⟶ *h*
shows ⊢ *g*;(*fpowerstar f*) ⟶ *h*
proof −
have *1*: ⊢ *g*;(*fpowerstar f*) = *g*;(∃ *n*. *power f n*)
by (*simp add*: *fpowerstar-d-def*)
have *2*: ⊢ (*g*;(∃ *n*. *power f n*)) = (∃ *n*. *g*;(*power f n*))
using *ChopExistPower* **by** *blast*
have *3*: ⋀ *n*. ⊢ *g*;(*power f n*) ⟶ *h*
using *ChopInductR assms* **by** *blast*
have *4*: ⊢ (∃ *n*. *g*;(*power f n*)) ⟶ *h*
using *3* **by** *fastforce*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *PowerstarInductR*:
assumes ⊢ *g* ∨ *h*;*f* ⟶ *h*
shows ⊢ *g*;(*powerstar f*) ⟶ *h*
proof −
have *1*: ⊢ *g*;(*powerstar f*) = *g*;((∃ *n*. *power f n*);(*empty* ∨ (*f* ∧ *inf*)))
by (*simp add*: *chopstar-d-def powerstar-d-def*)
have *11*: ⊢ *g*;((∃ *n*. *power f n*);(*empty* ∨ (*f* ∧ *inf*))) =

$(g;(\exists\ n.\ power\ f\ n));(empty \vee (f \wedge inf))$
using *ChopAssoc* **by** *blast*
have *2*: $\vdash (g;(\exists\ n.\ power\ f\ n)) = (\exists\ n.\ g;(power\ f\ n))$
using *ChopExistPower* **by** *blast*
hence *21*: $\vdash\ (g;(\exists\ n.\ power\ f\ n));(empty \vee (f \wedge inf)) =$
$(\exists\ n.\ g;(power\ f\ n));(empty \vee (f \wedge inf))$
using *LeftChopEqvChop* **by** *blast*
have *3*: $\bigwedge\ n. \vdash g;(power\ f\ n) \longrightarrow h$
using *ChopInductR assms* **by** *blast*
have *31*: $\bigwedge\ n. \vdash g;((power\ f\ n);(f \wedge inf)) \longrightarrow h$
using *ChopInductInfR assms* **by** *blast*
have *32*: $\bigwedge n. \vdash g;((power\ f\ n);(empty \vee (f \wedge inf))) \longrightarrow h$
using *3 31*
by (*metis* (*no-types*, *lifting*) *ChopEmpty ChopOrEqv Prop02 inteq-reflection*)
have *4*: $\vdash (\exists\ n.\ g;((power\ f\ n);(empty \vee (f \wedge inf)))) \longrightarrow h$
using *32* **by** *fastforce*
from *1 11 2 21 4* **show** *?thesis*
by (*metis InfiniteSemantics.ChopExist InfiniteSemantics.ExistChop inteq-reflection*)
qed

lemma *ChopstarInductR*:
assumes $\vdash g \vee h;f \longrightarrow h$
shows $\vdash g;(chopstar\ f) \longrightarrow h$
proof −
have *1*: $\vdash g;(chopstar\ f) =$
$g;((\exists\ n.\ power\ (f \wedge more)\ n);(empty \vee ((f \wedge more) \wedge inf)))$
by (*simp add*: *chopstar-d-def powerstar-d-def*)
have *11*: $\vdash g;((\exists\ n.\ power\ (f \wedge more)\ n);(empty \vee ((f \wedge more) \wedge inf))) =$
$(g;(\exists\ n.\ power\ (f \wedge more)\ n));(empty \vee ((f \wedge more) \wedge inf))$
using *ChopAssoc* **by** *blast*
have *2*: $\vdash (g;(\exists\ n.\ power\ (f \wedge more)\ n));(empty \vee ((f \wedge more) \wedge inf)) =$
$((\exists\ n.\ g;power\ (f \wedge more)\ n));(empty \vee ((f \wedge more) \wedge inf))$
using *ChopExistPower LeftChopEqvChop* **by** *fastforce*
have *21*: $\vdash g \vee h;(f \wedge more) \longrightarrow h$
using *ChopAndA assms* **by** *fastforce*
have *3*: $\bigwedge\ n. \vdash g;(power\ (f \wedge more)\ n) \longrightarrow h$
using *21 ChopInductR*[*of g h LIFT*$(f \wedge more)$] *assms* **by** *auto*
have *31*: $\bigwedge\ n. \vdash g;((power\ (f \wedge more)\ n);(f \wedge inf)) \longrightarrow h$
using *assms 3*
by (*metis 21 AndMoreAndInfEqvAndInf ChopInductInfR inteq-reflection*)
have *32*: $\bigwedge n. \vdash g;((power\ (f \wedge more)\ n);(empty \vee (f \wedge inf))) \longrightarrow h$
using *3 31*
by (*metis* (*no-types*, *lifting*) *ChopEmpty ChopOrEqv Prop02 inteq-reflection*)
have *4*: $\vdash (\exists\ n.\ g;((power\ (f \wedge more)\ n);(empty \vee (f \wedge inf)))\ ) \longrightarrow h$
using *32* **by** *fastforce*
from *1 11 2 4* **show** *?thesis*
by (*metis InfiniteSemantics.ChopExist InfiniteSemantics.ExistChop AndMoreAndInfEqvAndInf inteq-reflection*)
qed

lemma *ChopstarInductMoreR*:
assumes $\vdash$ *g* $\vee$ *h*;(*f* $\wedge$ *more*) $\longrightarrow$ *h*
shows $\vdash$ *g*;(*chopstar f*) $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ *g*;(*chopstar f*) = *g*;(($\exists$ *n. power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)))
by (*simp add*: *chopstar-d-def powerstar-d-def*)
have *11*: $\vdash$ *g*;(($\exists$ *n. power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*))) =
(*g*;($\exists$ *n. power* (*f* $\wedge$ *more*) *n*));(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*))
using *ChopAssoc* **by** *blast*
have *2*: $\vdash$ (*g*;($\exists$ *n. power* (*f* $\wedge$ *more*) *n*));(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) =
(($\exists$ *n. g*;*power* (*f* $\wedge$ *more*) *n*));(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*))
using *ChopExistPower LeftChopEqvChop* **by** *fastforce*
have *3*: $\bigwedge$ *n*. $\vdash$ *g*;(*power* (*f* $\wedge$ *more*) *n*) $\longrightarrow$ *h*
using *ChopInductR assms* **by** (*metis*)
have *31*: $\bigwedge$ *n*. $\vdash$ *g*;((*power* (*f* $\wedge$ *more*) *n*);(*f* $\wedge$ *inf*)) $\longrightarrow$ *h*
using *assms*
by (*metis ChopInductInfR AndMoreAndInfEqvAndInf inteq-reflection*)
have *32*: $\bigwedge$*n*. $\vdash$ *g*;((*power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ (*f* $\wedge$ *inf*))) $\longrightarrow$ *h*
using *3 31*
by (*metis* (*no-types*, *lifting*) *ChopEmpty ChopOrEqv Prop02 inteq-reflection*)
have *4*: $\vdash$ ($\exists$ *n. g*;((*power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ (*f* $\wedge$ *inf*)))) $\longrightarrow$ *h*
using *32* **by** *fastforce*
from *1 11 2 4* **show** *?thesis*
by (*metis InfiniteSemantics.ChopExist InfiniteSemantics.ExistChop AndMoreAndInfEqvAndInf inteq-reflection*)
qed

lemma *FPSEqvEmptyOrFiniteChopFPS*:
$\vdash$ *fpowerstar f* = (*empty* $\vee$ (*f* $\wedge$ *finite*);*fpowerstar f*)
using *FPowerstarEqvSem Valid-def* **by** *blast*

lemma *FPSAndMoreImpFPS*:
$\vdash$ *fpowerstar* (*f* $\wedge$ *more*) $\longrightarrow$ *fpowerstar f*
proof −
have *1*: $\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\longrightarrow$ (*f* $\wedge$ *finite*)
by *auto*
have *11*: $\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);*fpowerstar f* $\longrightarrow$ (*f* $\wedge$ *finite*); *fpowerstar f*
using *1 LeftChopImpChop* **by** *blast*
have *2*: $\vdash$ *empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);*fpowerstar f* $\longrightarrow$ *fpowerstar f*
using *11 FPSEqvEmptyOrFiniteChopFPS*[*of f*]
by *fastforce*
have *3*: $\vdash$ *fpowerstar* (*f* $\wedge$ *more*);*empty* $\longrightarrow$ *fpowerstar f*
using *2 FPowerstarInductL* **by** *blast*
from *2 3* **show** *?thesis* **by** (*metis ChopEmpty int-eq*)
qed

lemma *FPSImpAndMoreFPS*:
$\vdash$ *fpowerstar f* $\longrightarrow$ *fpowerstar* (*f* $\wedge$ *more*)
by (*meson ChopEmpty FPSEqvEmptyOrFiniteChopFPS FPowerstarInductMoreL int-iffD2 lift-imp-trans*)

lemma *FPSAndMoreEqvFPS*:
$\vdash$ *fpowerstar* $(f \wedge$ *more*$)$ = *fpowerstar f*
using *FPSAndMoreImpFPS FPSImpAndMoreFPS* **by** *fastforce*

lemma *ChopstarImpPowerstar*:
$\vdash$ $f^\star \longrightarrow$ *powerstar f*
by (*metis ChopEmpty ChopstarInductL PSEqvEmptyOrChopPS int-eq int-iffD2*)

lemma *PowerstarImpChopstar*:
$\vdash$ *powerstar f* $\longrightarrow$ $f^\star$
by (*metis CSEqvOrChopCS ChopEmpty PowerstarInductL int-iffD2 inteq-reflection*)

lemma *ChopstarEqvPowerstar*:
$\vdash$ $f^\star$ = *powerstar f*
using *ChopstarImpPowerstar PowerstarImpChopstar* **by** *fastforce*

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 *AndMoreCSEqvAndFmoreOrInf*:
$\vdash$ $(f \wedge$ *more*$)$;$f^\star$ = $((f \wedge$ *fmore*$)$;$f^\star \vee (f \wedge$ *inf*$))$
proof −
have *1*: $\vdash$ $(f \wedge$ *more*$)$ = ($(f \wedge$ *fmore*$)$ $\vee$ $(f \wedge$ *inf*$))$
by (*simp add*: *Valid-def more-defs chop-defs fmore-defs infinite-defs sum.case-eq-if*)
hence *2*: $\vdash$ $(f \wedge$ *more*$)$;$f^\star$ = ($(f \wedge$ *fmore*$)$ $\vee$ $(f \wedge$ *inf*$))$;$f^\star$
by (*simp add*: *LeftChopEqvChop*)
have *3*: $\vdash$ ($(f \wedge$ *fmore*$)$ $\vee$ $(f \wedge$ *inf*$))$;$f^\star$ = $((f \wedge$ *fmore*$)$;$f^\star \vee (f \wedge$ *inf*$)$;$f^\star$)
by (*simp add*: *OrChopEqv*)
have *4*: $\vdash$ $(f \wedge$ *inf*$)$;$f^\star$ = $(f \wedge$ *inf*$)$
using *AndInfChopEqvAndInf* **by** *blast*
have *5*: $\vdash$$((f \wedge$ *fmore*$)$;$f^\star \vee (f \wedge$ *inf*$)$;$f^\star$) = $((f \wedge$ *fmore*$)$;$f^\star \vee (f \wedge$ *inf*$)$)

using *3 4* **by** *auto*
from *2 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *PowerAndMoreAndFinite*:
$\vdash$ ((*power* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*) = (*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) *n*)
proof
(*induct n*)
case *0*
then show *?case* **using** *FiniteAndEmptyEqvEmpty* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof $-$
have *1*: $\vdash$ (*power* (*f* $\wedge$ *more*) (*Suc n*) $\wedge$ *finite*) =
(((*f* $\wedge$ *more*) $\wedge$ *finite*) ; *power* (*f* $\wedge$ *more*) *n* $\wedge$ *finite*)

by *simp*
have *2*: $\vdash$ (((*f* $\wedge$ *more*) $\wedge$ *finite*) ; *power* (*f* $\wedge$ *more*) *n* $\wedge$ *finite*) =
((((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *finite*) ; (*power* (*f* $\wedge$ *more*) *n* $\wedge$ *finite*))
by (*simp add*: *ChopAndFiniteDist*)
have *3*: $\vdash$ (((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *finite*) = (((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*)
by *auto*
have *4*: $\vdash$ (((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*); (*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) *n*) =
power ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc n*)

by *simp*
show *?thesis*
by (*metis 1 2 3 4 Suc.hyps inteq-reflection*)
qed
qed

lemma *CSAndFiniteDist*:
$\vdash$ (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite*) =
(($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*)
proof $-$
have *1*: $\vdash$(($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite*) =
((($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*);
((*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite*))
using *ChopAndFiniteDist* **by** *blast*
have *2*: $\vdash$ ((*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite*) = *empty*
by (*simp add*: *Valid-def empty-defs more-defs infinite-defs finite-defs sum.case-eq-if*)
from *1 2* **show** *?thesis*
by (*metis ChopEmpty inteq-reflection*)
qed

lemma *CSAndFinite*:

$\vdash$ (*f** $\wedge$ *finite*) = (*f* $\wedge$ *finite*)*
proof −
 have *1*: $\vdash$ (*f** $\wedge$ *finite*) = (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite*)
  by (*simp add*: *chopstar-d-def powerstar-d-def intI*)
 have *11*: $\vdash$ (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite*) =
   (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*)
 using *CSAndFiniteDist* **by** *blast*
 have *2*: $\vdash$ (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*) =
   ($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n* $\wedge$ *finite*)
  by (*simp add*: *Valid-def*)
 have *3*: $\vdash$ ($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n* $\wedge$ *finite*) =
   ($\exists$ *n*. (*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) *n*))
  using *PowerAndMoreAndFinite* **by** *fastforce*
 have *31*: $\vdash$ (*empty* $\vee$ ((*f* $\wedge$ *finite*) $\wedge$ *inf*)) = *empty*
 by (*simp add*: *Valid-def empty-defs infinite-defs finite-defs sum.case-eq-if*)
 have *4*: $\vdash$ ($\exists$ *n*. (*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) *n*)) = (*f* $\wedge$ *finite*)*
 using *31*
 by (*metis ChopEmpty AndMoreAndInfEqvAndInf chopstar-d-def inteq-reflection powerstar-d-def*)
 from *1 11 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *PowerchopAndMore*:
$\vdash$ ((*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc n*)) $\wedge$ *more*) =
   (*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc n*))
proof (*induct n*)
case *0*
then show *?case*
 proof −
  have *01*: $\vdash$ (*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc 0*) $\wedge$ *more*) =
    ((((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*);*empty* $\wedge$ *more*)

   by *simp*
  have *02*: $\vdash$ ((((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*);*empty* $\wedge$ *more*) =
    ((((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *more*)

   by (*metis ChopEmpty inteq-reflection*)
  have *03*: $\vdash$ ((((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *more*) =
    ((((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*))
   by *auto*
  have *04*: $\vdash$ ((((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*)) =
    ((((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*));*empty*

   by (*simp add*: *ChopEmpty int-iffD1 int-iffD2 int-iffI*)
  have *05*: $\vdash$ ((((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*));*empty*=
    power ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc 0*)
   by *simp*
  show *?thesis*
  by (*metis 03 04 05 inteq-reflection*)
 qed
next

case (*Suc n*)
then show *?case*
 by (*metis LeftChopImpMoreRule Prop10 Prop11 Prop12 lift-and-com pow-Suc*)
qed

lemma *PowerchopAndFmore*:
$\vdash$ ((*power* (*f* $\wedge$ *more*) (*Suc n*)) $\wedge$ *fmore*) = (*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc n*))
proof –
 have *1*: $\vdash$ ((*power* (*f* $\wedge$ *more*) (*Suc n*)) $\wedge$ *fmore*) =
 (((*power* (*f* $\wedge$ *more*) (*Suc n*)) $\wedge$ *finite*) $\wedge$ *more*)
 by (*auto simp add*: *fmore-d-def*)
 have *2*: $\vdash$ (((*power* (*f* $\wedge$ *more*) (*Suc n*)) $\wedge$ *finite*) $\wedge$ *more*) =
 ((*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc n*)) $\wedge$ *more*)

 by (*metis PowerAndMoreAndFinite inteq-reflection lift-and-com*)
 have *3*: $\vdash$ ((*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc n*)) $\wedge$ *more*) =
 (*power* ((*f* $\wedge$ *finite*) $\wedge$ *more*) (*Suc n*))
 using *PowerchopAndMore* **by** *blast*
 show *?thesis* **using** *1 2 3* **by** *fastforce*
qed

lemma *ExistPowerAndMoreExpand*:
$\vdash$ ($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) = (*empty* $\vee$ ($\exists$ *n*. (*power* (*f* $\wedge$ *more*) (*Suc n*))))
using *powersem1*[*of LIFT*(*f* $\wedge$ *more*)] **by** *auto*

lemma *CSAndFmoreDist*:
$\vdash$ (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *fmore*) =
 (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *fmore*)
proof –
 have *1*: $\vdash$(($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *fmore*) =
 (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite* $\wedge$ *more*)
 by (*metis fmore-d-def inteq-reflection lift-and-com*)
 have *2*: $\vdash$ (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite*) =
 (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*)
 using *CSAndFiniteDist* **by** *blast*
 have *3*: $\vdash$ (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *finite* $\wedge$ *more*) =
 (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite* $\wedge$ *more*)
 using *2* **by** *fastforce*
 have *4*: $\vdash$ (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite* $\wedge$ *more*) =
 (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*) $\wedge$ *fmore*)
 by (*metis fmore-d-def inteq-reflection lift-and-com*)
 from *1 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndMoreEqvAndFMoreChop*:
 $\vdash$ ($f^\star$ $\wedge$ *fmore*) = (*f* $\wedge$ *fmore*); (*f* $\wedge$ *finite*)*
proof –
 have *1*: $\vdash$ ($f^\star$ $\wedge$ *fmore*) = (($\exists$ *n*. *power* (*f* $\wedge$ *more*) *n*);(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)) $\wedge$ *fmore*)
 by (*simp add*: *chopstar-d-def powerstar-d-def intI*)

have *11*: ⊢ ((∃ *n*. *power* (*f* ∧ *more*) *n*);(*empty* ∨ ((*f* ∧ *more*) ∧ *inf*)) ∧ *fmore*) =
((∃ *n*. *power* (*f* ∧ *more*) *n*) ∧ *fmore*)
using *CSAndFmoreDist* **by** *fastforce*
have *2*: ⊢ ((∃ *n*. *power* (*f* ∧ *more*) *n*) ∧ *fmore*) =
(∃ *n*. *power* (*f* ∧ *more*) *n* ∧ *fmore*)
by (*simp add*: *Valid-def*)
have *3*: ⊢ (∃ *n*. *power* (*f* ∧ *more*) *n* ∧ *fmore*) =
((*power* (*f* ∧ *more*) *0* ∨ (∃ *n*. (*power* (*f* ∧ *more*) (*Suc n*)))) ∧ *fmore*)
using *ExistPowerAndMoreExpand* **by** *fastforce*
have *4*: ⊢ ((*power* (*f* ∧ *more*) *0* ∨ (∃ *n*. (*power* (*f* ∧ *more*) (*Suc n*)))) ∧ *fmore*) =
(((*power* (*f* ∧ *more*) *0* ∧ *fmore*) ∨ ((∃ *n*. (*power* (*f* ∧ *more*) (*Suc n*))) ∧ *fmore*)))
by *auto*
have *5*: ⊢ (((*power* (*f* ∧ *more*) *0* ∧ *fmore*) ∨ ((∃ *n*. (*power* (*f* ∧ *more*) (*Suc n*))) ∧ *fmore*))) =
((∃ *n*. (*power* (*f* ∧ *more*) (*Suc n*))) ∧ *fmore*)
using *NotFmoreAndEmpty* **by** *fastforce*
have *6*: ⊢ ((∃ *n*. (*power* (*f* ∧ *more*) (*Suc n*))) ∧ *fmore*) =
(∃ *n*. (*power* ((*f* ∧ *finite*) ∧ *more*) (*Suc n*)))
using *PowerchopAndFmore* **by** *fastforce*
have *7*: ⊢ (∃ *n*. (*power* ((*f* ∧ *finite*) ∧ *more*) (*Suc n*))) =
(∃ *n*. ((((*f* ∧ *finite*) ∧ *more*) ∧ *finite*);(*power* ((*f* ∧ *finite*) ∧ *more*) *n*)))
by (*simp*)
have *8*: ⊢ (∃ *n*. ((((*f* ∧ *finite*) ∧ *more*) ∧ *finite*);(*power* ((*f* ∧ *finite*) ∧ *more*) *n*))) =
(((*f* ∧ *finite*) ∧ *more*) ∧ *finite*);(∃ *n*. (*power* ((*f* ∧ *finite*) ∧ *more*) *n*))
by (*auto simp add*: *Valid-def sum.case-eq-if chop-defs*)
have *9*: ⊢ (((*f* ∧ *finite*) ∧ *more*) ∧ *finite*) = (*f* ∧ *fmore*)
by (*auto simp add*: *fmore-d-def*)
have *10*: ⊢ (((*f* ∧ *finite*) ∧ *more*) ∧ *finite*);(∃ *n*. (*power* ((*f* ∧ *finite*) ∧ *more*) *n*)) =
(*f* ∧ *fmore*);(∃ *n*. (*power* ((*f* ∧ *finite*) ∧ *more*) *n*))
using *8 9* **by** (*simp add*: *LeftChopEqvChop*)
have *101*: ⊢ (*empty* ∨ ((*f* ∧ *finite*) ∧ *inf*)) = *empty*
by (*simp add*: *Valid-def empty-defs infinite-defs finite-defs sum.case-eq-if*)
have *11*: ⊢ (∃ *n*. (*power* ((*f* ∧ *finite*) ∧ *more*) *n*)) =
(*f* ∧ *finite*)*
using *101*
by (*metis ChopEmpty AndMoreAndInfEqvAndInf chopstar-d-def inteq-reflection powerstar-d-def*)
hence *12*: ⊢ (*f* ∧ *fmore*);(∃ *n*. (*power* ((*f* ∧ *finite*) ∧ *more*) *n*)) =
(*f* ∧ *fmore*);(*f* ∧ *finite*)*
by (*simp add*: *RightChopEqvChop*)
from *1 11 2 3 4 5 6 7 8 10 12* **show** *?thesis*
by (*metis CSAndFmoreDist inteq-reflection*)
qed

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

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

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

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

lemma *ChopplusCommuteImpA*:
$\vdash f^\star;f \longrightarrow f;f^\star$
by (*metis CSEqvOrChopCS ChopAndB ChopEmpty ChopstarInductL EmptyImpCS Prop02 Prop03 Prop10 inteq-reflection*)

lemma *ChopplusCommuteImpB*:
$\vdash f;f^\star \longrightarrow f^\star;f$

proof −
have *f2*: $\vdash (f^\star;f);f \longrightarrow f^\star;f$
by (*metis CSEqvOrChopCS ChopplusCommuteImpA LeftChopImpChop Prop05 inteq-reflection*)
have $\vdash f \longrightarrow f^\star;f$
by (*metis AndChopB EmptyChop EmptyImpCS Prop10 inteq-reflection*)
then show *?thesis*
using *f2 ChopstarInductR Prop02* **by** *blast*

qed

lemma *ChopplusCommute*:
$\vdash f;f^{\star} = f^{\star};f$
using *ChopplusCommuteImpA ChopplusCommuteImpB* **by** *fastforce*

lemma *CSEqvOrChopCSB*:
$\vdash \ f^{\star} = \ (empty \ \vee \ (f^{\star};f))$
by (*meson CSEqvOrChopCS ChopplusCommute Prop06*)

lemma *CSAndMoreImpCSChop*:
$\vdash \ f^{\star} \wedge \ more \ \longrightarrow f^{\star};\ f$
using *CSAndMoreEqvAndMoreChop ChopplusCommute CSAndMoreImpChopCS* **by** *fastforce*

lemma *PowerChopPower*:
$\vdash (power\ (f \wedge more)\ n);\ (power\ (f \wedge more)\ k) = (power\ (f \wedge more)\ (n+k))$
proof
(*induct n arbitrary*: *k*)
case *0*
then show *?case* **using** *EmptyChopSem* **by** *auto*
next
case (*Suc n*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *ChopAssoc add-Suc inteq-reflection pow-Suc*)
qed

lemma *CSChopCS*:
$\vdash f^{\star}\ ;\ f^{\star} = f^{\star}$
proof −
have *1*: $\vdash f^{\star}\ ;\ f^{\star} \ \longrightarrow \ f^{\star}$
by (*meson CSEqvOrChopCSB ChopstarImpPowerstar ChopstarInductR PowerstarImpChopstar Prop02 Prop03 lift-imp-trans*)
have *2*: $\vdash f^{\star} \longrightarrow f^{\star}\ ;\ f^{\star}$
by (*metis ChopEmpty EmptyImpCS RightChopImpChop inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

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

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

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

lemma *PowerAndInfB*:
$\vdash$ (((*f* $\wedge$ *more*);(*power* (*f* $\wedge$ *more*) *n*)) $\wedge$ *inf*) =
((*f* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *fmore*);((*power* (*f* $\wedge$ *more*) *n*) $\wedge$ *inf*))
proof −
have *1*: $\vdash$ (((*f* $\wedge$ *more*);(*power* (*f* $\wedge$ *more*) *n*)) $\wedge$ *inf*) =
(((*f* $\wedge$ *more*) $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);((*power* (*f* $\wedge$ *more*) *n*) $\wedge$ *inf*))
using *ChopAndInfB* **by** *blast*
have *2*: $\vdash$ (((*f* $\wedge$ *more*) $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);((*power* (*f* $\wedge$ *more*) *n*) $\wedge$ *inf*)) =
((*f* $\wedge$ *inf*) $\vee$ (*f* $\wedge$ *fmore*);((*power* (*f* $\wedge$ *more*) *n*) $\wedge$ *inf*))
using *AndMoreAndInfEqvAndInf AndMoreAndFiniteEqvAndFmore*
by (*metis 1 inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndInf*:
$\vdash$ ($f^\star$ $\wedge$ *inf*) = $f^\star$;(*f* $\wedge$ *inf*)
by (*meson AndChopA AndInfChopEqvAndInf CSEqvOrChopCSB ChopAndA ChopAndInf Prop03 Prop11 Prop12 lift-imp-trans*)

lemma *CSChopCSImpCS*:
$\vdash$ ($f^\star$; $f^\star$) $\longrightarrow$ $f^\star$
by (*simp add*: *CSChopCS int-iffD1*)

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*)

hence *3*: $\vdash f^\star;\ f \longrightarrow f^\star;\ f^\star$ **by** *auto*
have *4*: $\vdash f^\star;\ f^\star \longrightarrow f^\star$ **by** (*rule CSChopCSImpCS*)
from *2 3 4* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

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

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

lemma *CSElim*:
assumes $\vdash\ empty \longrightarrow g$
$\vdash (f \land\ more\ );\ g \longrightarrow g$
shows $\vdash f^\star \longrightarrow g$
proof −
have *1*: $\vdash empty \lor (f \land more);g \longrightarrow g$
using *assms* **using** *Prop02* **by** *blast*
have *2*: $\vdash (chopstar\ f);empty \longrightarrow g$
using *ChopstarInductMoreL 1* **by** *blast*
from *2* **show** *?thesis*
by (*metis ChopEmpty inteq-reflection*)
qed

lemma *ChopstarImp*:
assumes $\vdash f;(chopstar\ g) \lor empty \longrightarrow (chopstar\ g)$
shows $\vdash (chopstar\ f) \longrightarrow (chopstar\ g)$
using *assms ChopstarInductL ChopEmpty*
by (*metis int-eq int-simps*(*33*) *lift-and-com*)

lemma *CSCSImpCS*:
$\vdash\ (f^\star)^\star \longrightarrow f^\star$
proof −
have *1*: $\vdash ((chopstar\ f);(chopstar\ f)) \lor empty \longrightarrow (chopstar\ f)$
by (*meson CSChopCSImpCS EmptyImpCS Prop02*)
from *1* **show** *?thesis* **using** *ChopstarImp* **by** *blast*
qed

lemma *CSImpCSCS*:
$\vdash f^\star \longrightarrow (f^\star)^\star$
using *ImpCS* **by** *auto*

lemma *CSCSEqvCS*:
$\vdash (f^\star)^\star \ = f^\star$
by (*simp add*: *CSCSImpCS CSImpCSCS int-iffI*)

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 NotEmptyEqvMore Prop01 Prop05 Prop07 RightChopImpMoreRule empty-d-def int-iffD2*
int-simps(*15*) *inteq-reflection lift-imp-neg*)

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*: ⊢ (*f* ∧ *empty*);*empty* = (*f* ∧ *empty*)
by (*rule ChopEmpty*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

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

lemma *ChopPlusEqv*:
⊢ (*f*;*f*⋆) = (*f* ∨ (*f* ∧ *more*); (*f*;*f*⋆))
proof −
have *1*: ⊢ *f*⋆ = (*empty* ∨ (*f* ∧ *more*); *f*⋆)
by (*rule ChopstarEqv*)
have *2*: ⊢ *f*⋆ = (*empty* ∨ *f*;*f*⋆)
by (*rule CSEqvOrChopCS*)
hence *3*: ⊢ (*empty* ∨ *f*;*f*⋆) = (*empty* ∨ (*f* ∧ *more*);*f*⋆)
using *1 2* **by** *fastforce*
have *4*: ⊢ (*f* ∧ *more*);(*f*⋆) = (*f* ∧ *more*);(*empty* ∨ *f*;*f*⋆)
using *2* **using** *RightChopEqvChop* **by** *blast*
hence *5*: ⊢ *empty* ∨ *f*;*f*⋆ = *empty* ∨ (*f* ∧ *more*);(*empty* ∨ *f*;*f*⋆)
using *3 4* **by** *fastforce*
have *6*: ⊢ (*f* ∧ *more*); (*empty* ∨ *f*;*f*⋆) =
((*f* ∧ *more*); *empty* ∨ (*f* ∧ *more*); (*f*;*f*⋆))
using *ChopOrEqv* **by** *blast*
have *7*: ⊢ (*f* ∧ *more*); *empty* = (*f* ∧ *more*)
using *ChopEmpty* **by** *blast*
have *8*: ⊢ (*empty* ∨ *f*;*f*⋆) =
(*empty* ∨ (*f* ∧ *more*) ∨ (*f* ∧ *more*); (*f*;*f*⋆))
using *5 6 7* **by** (*metis 2 3 inteq-reflection*)
have *9*: ⊢ ((*empty* ∨ *f*;*f*⋆) ∧ *more*) = (*f*;*f*⋆ ∧ *more*)
by (*auto simp*: *empty-d-def*)
have *10*: ⊢ ((*empty* ∨ (*f* ∧ *more*) ∨ (*f* ∧ *more*); (*f*;*f*⋆)) ∧ *more*) =
(((*f* ∧ *more*) ∨ (*f* ∧ *more*); (*f*;*f*⋆)) ∧ *more*)
by (*auto simp*: *empty-d-def*)
have *11*: ⊢ (((*f* ∧ *more*) ∨ (*f* ∧ *more*); (*f*;*f*⋆)) ∧ *more*) =
((*f* ∧ *more*) ∨ (*f* ∧ *more*); (*f*;*f*⋆))
using *10 6 7 int-eq*
using *AndMoreChopAndMoreEqvAndMoreChop* **by** *fastforce*
have *12*: ⊢ (*f*;*f*⋆ ∧ *more*) = ((*f* ∧ *more*) ∨ (*f* ∧ *more*); (*f*;*f*⋆))
using *8 9 10 11* **by** *fastforce*
have *13*: ⊢ (*f*;*f*⋆ ∧ *empty*) = (*f* ∧ *empty*)
by (*rule ChopCSAndEmptyEqvAndEmpty*)
have *14*: ⊢ ((*f* ∧ *more*) ∨ (*f* ∧ *more*); (*f*;*f*⋆) ∨ (*f* ∧ *empty*)) =
(*f* ∨ (*f* ∧ *more*);(*f*;*f*⋆))
by (*auto simp*: *empty-d-def*)
have *15*: ⊢ *f*;*f*⋆ = ((*f*;*f*⋆ ∧ *empty*) ∨ (*f*;*f*⋆ ∧ *more*))
by (*auto simp*: *empty-d-def*)
from *12 13 14 15* **show** *?thesis* **by** *fastforce*

qed

lemma *ChopPlusImpChopPlus*:
assumes $\vdash\ \ f \longrightarrow g$
shows $\vdash\ \ f;f^\star \longrightarrow g;g^\star$
using *assms*
by (*metis AndChopB CSChopCS ChopImpChop ChopstarImp EmptyImpCS ImpCS Prop01 Prop02 Prop05 Prop10 inteq-reflection*)

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 $\vdash f \longrightarrow g \vee (g \wedge$ *more*); f
shows $\vdash f \wedge$ *finite* $\longrightarrow g;g^\star$
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** *ChopContraB* **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*: $\vdash f \vee (f \wedge$ *more*)$;g \longrightarrow g$

using *assms Prop02* **by** *blast*
have *2*: $\vdash f^\star;f \longrightarrow g$
using *ChopstarInductMoreL 1* **by** *blast*
from *2* **show** *?thesis*
using *ChopplusCommute* **by** *fastforce*
qed

lemma *ChopPlusElimWithoutMore*:
assumes $\vdash \; f \longrightarrow g$
$\vdash f;\, g \longrightarrow g$
shows $\vdash f;f^\star \longrightarrow g$
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 $\vdash f \wedge finite \longrightarrow g^\star$
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** *ChopContraB* **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 $\ \vdash f \wedge finite \longrightarrow g^\star;\ h$
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** *ChopContraB* **by** *blast*
qed

lemma *DiamondAndEmptyEqvAndEmpty*:

$\vdash (\Diamond\ f\ \wedge\ empty) = (f\ \wedge\ empty)$
by (*auto simp*: *sometimes-defs empty-defs sum.case-eq-if*)

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*) $\wedge$ *finite*); *f*
shows $\vdash$ (*init w* $\wedge$ *f*) $\wedge$ *finite* $\longrightarrow \Box$ (*init w*)
proof –
have *1*: $\vdash f \longrightarrow$ *empty* $\vee$ $((\Box($ *init w*) $\wedge$ *more*) $\wedge$ *finite*); *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*) $\wedge$ *finite*); *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*)
have *41*: $\vdash (\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite* $\longrightarrow$
$((\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *fin* (*init w*)
using *4* **by** *auto*
hence *5*: $\vdash ((\Box($ *init w*) $\wedge$ *more*) $\wedge$ *finite*); *f* $\longrightarrow$
$(((\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); *f*
by (*rule LeftChopImpChop*)
have *6*: $\vdash (((\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *fin* (*init w*)); *f* =
$((\Box($ *init w*) $\wedge$ *more*) $\wedge$ *finite*); (*init w* $\wedge$ *f*)
using *AndFinChopEqvStateAndChop* **by** *blast*
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*) $\wedge$ *finite*) *yields* $(\neg(\Box($ *init w*)))
using *AndYieldsA*
by (*metis AndMoreAndFiniteEqvAndFmore inteq-reflection*)
have *9*: $\vdash ((\Box($ *init w*) $\wedge$ *more*) $\wedge$ *finite*); (*init w* $\wedge$ *f*) $\wedge$
$((\Box($ *init w*) $\wedge$ *more*) $\wedge$ *finite*) *yields* $(\neg(\Box$ (*init w*)))
$\longrightarrow$
$((\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*); ((*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*) $\wedge$ *finite*); ((*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*) $\wedge$ *finite*); ((*init w* $\wedge$ *f*) $\wedge \neg(\Box$ (*init w*))) $\longrightarrow$
more ; ((*init w* $\wedge$ *f*) $\wedge \neg(\Box$ (*init w*)))
by (*metis 41 LeftChopImpChop Prop12*)
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** *MoreChopContraFiniteB* **by** *blast*
qed

lemma *BoxCSEqvBox*:
$\vdash$ (*init w* $\wedge (\Box($ *init w*$))^{\star}) = \Box$ (*init w*)
proof –
have *1*: $\vdash \Box($ *init w*);$\Box($ *init w*) $\longrightarrow \Box($ *init w*)
by (*simp add*: *BoxStateChopBoxEqvBox int-iffD1*)

have *2*: $\vdash$ (*init w* $\wedge$ *empty*) $\longrightarrow$ $\Box$(*init w*)
by (*simp add*: *StateAndEmptyImpBoxState*)
have *3*: $\vdash$ (*init w* $\wedge$ *empty*) $\vee$ $\Box$(*init w*);$\Box$(*init w*) $\longrightarrow$ $\Box$(*init w*)
using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*init w* $\wedge$ *empty*);($\Box$(*init w*))* $\longrightarrow$ $\Box$ (*init w*)
using *ChopstarInductR 3* **by** *blast*
have *5*: $\vdash$ *init w* $\wedge$ ($\Box$(*init w*))* $\longrightarrow$ $\Box$ (*init w*)
using *4 StateAndEmptyChop* **by** *fastforce*
have *11*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ (*init w*)
    using *BoxElim* **by** *blast*
have *12*: $\vdash$ $\Box$(*init w*) $\longrightarrow$ ($\Box$ (*init w*))*
    by (*rule ImpCS*)
have *13*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ *init w* $\wedge$ ($\Box$ (*init w*))*
    using *11 12* **by** *fastforce*
from *5 13* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateAndCSEqvCS*:
$\vdash$ ($\Box$(*init w*) $\wedge$ $f^\star$ $\wedge$ *finite*) = (*init w* $\wedge$ ($\Box$(*init w*) $\wedge$ *f*)* $\wedge$ *finite*)
proof −
have *1*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ *init w*
    using *BoxElim* **by** *blast*
have *2*: $\vdash$ ($f^\star$ $\wedge$ *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*) $\wedge$ $f^\star$) $\wedge$ *finite* $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *f*)*
    using *CSIntro* **by** *blast*
have *71*: $\vdash$ *init w* $\wedge$ $\Box$ (*init w*) $\wedge$ $f^\star$ $\wedge$ *finite* $\longrightarrow$ *init w* $\wedge$ ($\Box$ (*init w*) $\wedge$ *f*)* $\wedge$ *finite*
    using *7* **by** *fastforce*
have *8*: $\vdash$ $\Box$(*init w*) $\wedge$ $f^\star$ $\wedge$ *finite* $\longrightarrow$ *init w* $\wedge$ ($\Box$ (*init w*) $\wedge$ *f*)* $\wedge$ *finite*
    using *1 71* **by** *fastforce*
have *11*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *f*)* $\longrightarrow$ ($\Box$ (*init w*))*
    by (*rule AndCSA*)
have *12*: $\vdash$ (*init w* $\wedge$ ($\Box$ (*init w*))*) = $\Box$ (*init w*)
    by (*rule BoxCSEqvBox*)
have *13*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *f*)* $\longrightarrow$ $f^\star$
    by (*rule AndCSB*)
have *14*: $\vdash$ *init w* $\wedge$ ($\Box$ (*init w*) $\wedge$ *f*)* $\longrightarrow$ *init w* $\wedge$ ($\Box$ (*init w*))* $\wedge$ $f^\star$
    using *11 13* **by** *fastforce*

have *15*: $\vdash$ *init w* $\wedge$ ($\Box$ (*init w*))* $\wedge$ $f^\star$ $\longrightarrow$ $\Box$ (*init w*) $\wedge$ $f^\star$
using *12* **by** *auto*
have *16*: $\vdash$ *init w* $\wedge$ ($\Box$ (*init w*) $\wedge$ *f*)* $\longrightarrow$ $\Box$ (*init w*) $\wedge$ $f^\star$
using *14 15 lift-imp-trans* **by** *blast*
from *8 16* **show** *?thesis* **by** *fastforce*
qed

lemma *BaCSImpCS*:
$\vdash$ *ba* ($f \longrightarrow g$) $\wedge$ *finite* $\longrightarrow$ $f^\star \longrightarrow g^\star$
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*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ((*ba* ($f \longrightarrow g$)) $\wedge$ ($f^\star$) $\wedge$ ($\neg$ ($g^\star$)))
using *MoreChopLoopFiniteB* **by** *blast*
from *12* **show** *?thesis* **by** (*simp add*: *Valid-def*)

qed

lemma *BaCSEqvCS*:
$\vdash$ *ba* $(f = g) \wedge$ *finite* $\longrightarrow (f^\star = g^\star)$
proof $-$
have *1*: $\vdash$ *ba* $(f = g) = ($*ba* $(f \longrightarrow g) \wedge$ *ba* $(g \longrightarrow f))$ **by** (*auto simp*: *ba-defs sum.case-eq-if*)
have *2*: $\vdash$ *ba* $(f \longrightarrow g) \wedge$ *finite* $\longrightarrow (f^\star \longrightarrow g^\star)$ **by** (*rule BaCSImpCS*)
have *3*: $\vdash$ *ba* $(g \longrightarrow f) \wedge$ *finite* $\longrightarrow (g^\star \longrightarrow f^\star)$ **by** (*rule BaCSImpCS*)
have *4*: $\vdash$ *ba* $(f = g) \wedge$ *finite* $\longrightarrow (f^\star \longrightarrow g^\star) \wedge (g^\star \longrightarrow f^\star)$ **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*:
$\vdash$ *ba* $f \wedge g^\star \wedge$ *finite* $\longrightarrow (f \wedge g)^\star$
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* $(g \longrightarrow f \wedge g) \wedge$ *finite* $\longrightarrow g^\star \longrightarrow (f \wedge g)^\star$ **by** (*rule BaCSImpCS*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *CSSkipImpFinite*:
$\vdash$ *skip*$^\star \longrightarrow$ *finite*
using *CSElimWithoutMore EmptyImpFinite SkipChopFiniteImpFinite* **by** *blast*

lemma *FiniteImpCSSkip*:
$\vdash$ *finite* $\longrightarrow$ *skip**
using *CSIntro*
by (*metis* (*no-types*, *lifting*) *CSSkipImpFinite ChopAndB FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite ImpChopPlus Prop10 Prop12 int-iffD1 inteq-reflection*)

lemma *CSSkipEqvFinite*:
$\vdash$ *skip*$^\star =$ *finite*
using *CSSkipImpFinite FiniteImpCSSkip* **by** *fastforce*

13.9 Properties of Omega

lemma *NotOmegaEmpty*:
$\vdash \neg(($*empty*$)^\omega)$
proof $-$
have *1*: $\vdash ($*empty*$)^\omega = ($*empty* $\wedge$ *fmore*$);($*empty*$)^\omega$
by (*simp add*: *OmegaUnroll*)
have *2*: $\vdash ($*empty* $\wedge$ *fmore*$) = \#$*False*
using *NotFmoreAndEmpty* **by** *auto*
have *3*: $\vdash \#$*False*$;($*empty*$)^\omega = \#$*False*
by (*metis AndInfChopEqvAndInf int-eq int-simps*(*22*))

from *1 2 3* **show** *?thesis*
by (*metis TrueW int-simps*(*3*) *inteq-reflection*)
qed

lemma *NotOmegaFalse*:
$\vdash \neg((\#False)^{\omega})$
by (*metis ChopImpDi DiIntro NotDiFalse OmegaUnroll int-iffI int-simps*(*14*)
int-simps(*19*) *inteq-reflection*)

lemma *NotOmegaInf*:
$\vdash \neg((inf)^{\omega})$
proof −
have *1*: $\vdash (inf)^{\omega} = (inf \wedge fmore);(inf)^{\omega}$
by (*simp add*: *OmegaUnroll*)
have *2*: $\vdash (inf \wedge fmore) = \#False$
using *FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*
FmoreEqvSkipChopFinite InfEqvNotFinite **by** *fastforce*
have *3*: $\vdash \#False;(inf)^{\omega} = \#False$
by (*metis AndInfChopEqvAndInf int-eq int-simps*(*22*))
from *1 2 3* **show** *?thesis*
by (*metis TrueW int-simps*(*3*) *inteq-reflection*)
qed

lemma *OmegaLenPlusOneImpInf*:
$\vdash (len(Suc\ n))^{\omega} \longrightarrow inf$
by (*simp add*: *Valid-def infinite-defs omega-d-def len-defs sum.case-eq-if*)

lemma *InfImpOmegaLenPlusOne*:
$\vdash inf \longrightarrow (len(Suc\ n))^{\omega}$
proof −
have *1*: $\vdash inf \wedge \#True \wedge \Box(\#True \longrightarrow (len(Suc\ n) \wedge fmore);\#True) \longrightarrow (len(Suc\ n))^{\omega}$
using *OmegaInduct* **by** *blast*
have *2*: $\vdash \Box(\#True \longrightarrow (len(Suc\ n) \wedge fmore);\#True) = inf$
by (*auto simp add*: *Valid-def len-defs fmore-defs chop-defs iprefix-length infinite-defs always-defs*
sum.case-eq-if)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *OmegaLenPlusOneEqvInf*:
$\vdash (len(Suc\ n))^{\omega} = inf$
using *OmegaLenPlusOneImpInf InfImpOmegaLenPlusOne* **by** *fastforce*

lemma *OmegaSkipEqvInf*:
$\vdash (skip)^{\omega} = inf$
proof −
have *1*: $\vdash skip = (len\ 1)$
by (*simp add*: *Valid-def skip-defs len-defs sum.case-eq-if*)
have *2*: $\vdash (skip)^{\omega} = (len\ 1)^{\omega}$
using *1* **by** (*metis OmegaUnroll inteq-reflection*)
from *2* **show** *?thesis* **using** *OmegaLenPlusOneEqvInf* **by** *fastforce*

qed

lemma *OmegaTrueImpInf*:
$\vdash (\# True)^{\omega} \longrightarrow inf$
by (*simp add*: *Valid-def infinite-defs omega-d-def skip-defs sum.case-eq-if*)

lemma *InfImpOmegaTrue*:
$\vdash inf \longrightarrow (\# True)^{\omega}$
proof −
 have *1*: $\vdash inf \wedge \# True \wedge \Box(\# True \longrightarrow (\# True \wedge fmore);\# True) \longrightarrow \# True^{\omega}$
 using *OmegaInduct* **by** *blast*
 have *2*: $\vdash \Box(\# True \longrightarrow (\# True \wedge fmore);\# True) = inf$
 by (*auto simp add*: *Valid-def skip-defs fmore-defs chop-defs iprefix-length infinite-defs always-defs sum.case-eq-if*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *OmegaTrueEqvInf*:
$\vdash (\# True)^{\omega} = inf$
using *OmegaTrueImpInf InfImpOmegaTrue* **by** *fastforce*

lemma *OmegaMoreImpInf*:
$\vdash (more)^{\omega} \longrightarrow inf$
by (*simp add*: *Valid-def infinite-defs omega-d-def more-defs sum.case-eq-if*)

lemma *InfImpOmegaMore*:
$\vdash inf \longrightarrow (more)^{\omega}$
proof −
 have *1*: $\vdash inf \wedge \# True \wedge \Box(\# True \longrightarrow (more \wedge fmore);\# True) \longrightarrow more^{\omega}$
 using *OmegaInduct* **by** *blast*
 have *2*: $\vdash \Box(\# True \longrightarrow (more \wedge fmore);\# True) = inf$
 by (*auto simp add*: *Valid-def skip-defs more-defs fmore-defs chop-defs iprefix-length infinite-defs always-defs sum.case-eq-if*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *OmegaMoreEqvInf*:
$\vdash (more)^{\omega} = inf$
using *OmegaMoreImpInf InfImpOmegaMore* **by** *fastforce*

lemma *OmegaFiniteImpInf*:
$\vdash (finite)^{\omega} \longrightarrow inf$
by (*simp add*: *Valid-def infinite-defs omega-d-def more-defs sum.case-eq-if*)

lemma *InfImpOmegaFinite*:
$\vdash inf \longrightarrow (finite)^{\omega}$
proof −
 have *1*: $\vdash inf \wedge \# True \wedge \Box(\# True \longrightarrow (finite \wedge fmore);\# True) \longrightarrow finite^{\omega}$
 using *OmegaInduct* **by** *blast*

have *2*: $\vdash \Box(\# True \longrightarrow (finite \wedge fmore);\# True) = inf$
by (*auto simp add*: *Valid-def skip-defs more-defs finite-defs fmore-defs chop-defs iprefix-length infinite-defs always-defs sum.case-eq-if*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *OmegaFiniteEqvInf*:
$\vdash (finite)^{\omega} = inf$
using *OmegaFiniteImpInf InfImpOmegaFinite* **by** *fastforce*

lemma *BoxStateAndInfImpOmegaBoxState*:
$\vdash inf \wedge \Box(init\ w) \longrightarrow (\Box(init\ w))^{\omega}$
proof –
have *1*: $\vdash inf \wedge (inf \wedge \Box(init\ w)) \wedge$
$\Box((inf \wedge \Box(init\ w)) \longrightarrow (\Box(init\ w) \wedge fmore);(inf \wedge \Box(init\ w))) \longrightarrow (\Box(init\ w))^{\omega}$
using *OmegaInduct* **by** *blast*
have *2*: $\vdash (inf \wedge \Box(init\ w)) = (inf \wedge (\Box(init\ w) \wedge fmore);\Box(init\ w))$
by (*metis* (*no-types*, *lifting*) *BoxStateAndChopEqvChop ChopAndInf FmoreEqvSkipChopFinite OmegaFiniteEqvInf OmegaUnroll Prop10 SkipChopFiniteImpFinite inteq-reflection lift-and-com*)
have *3*: $\vdash (inf \wedge (\Box(init\ w) \wedge fmore);\Box(init\ w)) = (\Box(init\ w) \wedge fmore);(inf \wedge \Box(init\ w))$
by (*metis ChopAndInf inteq-reflection lift-and-com*)
have *4*: $\vdash inf \wedge \Box(init\ w) \longrightarrow \Box((inf \wedge \Box(init\ w)) \longrightarrow (\Box(init\ w) \wedge fmore);(inf \wedge \Box(init\ w)))$
using *2 3* **by** (*metis* (*mono-tags*, *lifting*) *BoxGen intD intI inteq-reflection unl-lift2*)
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *OmegaBoxStateImpBoxState*:
$\vdash (\Box(init\ w))^{\omega} \wedge inf \longrightarrow \Box(init\ w)$
proof –
have *1*: $\vdash (\Box(init\ w))^{\omega} \longrightarrow init\ w$
by (*metis AndChopA BoxEqvAndEmptyOrNextBox OmegaUnroll Prop12 StateAndChop inteq-reflection*)
have *2*: $\vdash (\Box(init\ w))^{\omega} \longrightarrow (\Box(init\ w) \wedge fmore);((\Box(init\ w))^{\omega})$
by (*simp add*: *OmegaUnroll int-iffD1*)
have *21*: $\vdash (\Box(init\ w) \wedge fmore) \longrightarrow \bigcirc(\Box(init\ w))$
by (*metis AndChopB BoxStateAndChopEqvChop FmoreEqvSkipChopFinite NextAndEqvNextAndNext Prop12 inteq-reflection next-d-def*)
have *22*: $\vdash finite = (empty \vee fmore)$
by (*auto simp add*: *Valid-def finite-defs empty-defs fmore-defs sum.case-eq-if*)
have *23*: $\vdash (\Box\ (init\ w) \wedge finite) = ((\Box\ (init\ w) \wedge empty) \vee (\Box\ (init\ w) \wedge fmore))$
using *22* **by** *fastforce*
have *24*: $\vdash (\Box\ (init\ w) \wedge empty) = (init\ w \wedge empty)$
using *BoxEqvAndBox StateAndEmptyImpBoxState* **by** *fastforce*
have *25*: $\vdash \bigcirc(\Box(init\ w)) \wedge fmore \longrightarrow \bigcirc((\ (init\ w) \wedge empty) \vee (\Box\ (init\ w) \wedge fmore))$
using *23 24* **by** (*metis FmoreEqvSkipChopFinite NextAndEqvNextAndNext SkipChopEqvNext int-iffD2 inteq-reflection*)
have *26*: $\bigwedge g.\ \vdash (\bigcirc((\ (init\ w) \wedge empty) \vee (\Box\ (init\ w) \wedge fmore)));g =$
$(\bigcirc(init\ w \wedge g) \vee \bigcirc((\Box(init\ w) \wedge fmore);g))$
by (*metis* (*mono-tags*, *lifting*) *ChopOrEqvRule NextChop OrChopEqv StateAndEmptyChop inteq-reflection next-d-def*)

have *3*: $\vdash$ $(\Box(init\ w) \wedge fmore);((\Box(init\ w))^\omega) \longrightarrow$
$(\bigcirc(init\ w \wedge (\Box(init\ w))^\omega) \vee \bigcirc((\Box(init\ w) \wedge fmore);((\Box(init\ w))^\omega)))$
using *23 24 26*
by (*metis AndChopB BoxStateAndChopEqvChop FmoreEqvSkipChopFinite LeftChopImpChop inteq-reflection next-d-def*)
have *4*: $\vdash$ $(\bigcirc(init\ w \wedge (\Box(init\ w))^\omega) \vee \bigcirc((\Box(init\ w) \wedge fmore);((\Box(init\ w))^\omega))) \longrightarrow$
$\bigcirc(\ (\Box(init\ w))^\omega)$
by (*metis ChopAndB NextImpNext OmegaUnroll Prop02 Prop11 next-d-def*)
have *5*: $\vdash$ $(\Box(init\ w))^\omega \longrightarrow \bigcirc(\ (\Box(init\ w))^\omega)$
using *2 3 4* **by** *fastforce*
from *1 5* **show** *?thesis* **using** *BoxIntro* **by** (*metis Prop01 Prop05 inteq-reflection lift-and-com*)
qed

lemma *OmegaIntro*:
assumes $\vdash$ $h \longrightarrow (f \wedge fmore);h$
shows $\vdash$ $h \wedge inf \longrightarrow f^\omega$
proof −
have *1*: $\vdash$ $h \longrightarrow (f \wedge fmore);h$ **using** *assms* **by** *auto*
have *2*: $\vdash$ $\Box\ (h \longrightarrow (f \wedge fmore);h)$ **by** (*simp add*: *BoxGen assms*)
from *1 2* **show** *?thesis* **using** *OmegaInduct* **by** *fastforce*
qed

lemma *OmegaImpRule*:
assumes $\vdash$ $f \longrightarrow g$
shows $\vdash$ $f^\omega \wedge inf \longrightarrow g^\omega$
proof −
have *1*: $\vdash$ $(f \wedge fmore) \longrightarrow (g \wedge fmore)$
using *assms* **by** *auto*
have *2*: $\vdash$ $(f \wedge fmore);f^\omega \longrightarrow (g \wedge fmore);f^\omega$
using *1* **by** (*simp add*: *LeftChopImpChop*)
have *3*: $\vdash$ $\Box(f^\omega \longrightarrow (f \wedge fmore);f^\omega) \longrightarrow \Box(f^\omega \longrightarrow (g \wedge fmore);f^\omega)$
by (*metis 2 OmegaUnroll int-eq-true int-simps*(*13*) *inteq-reflection*)
have *4*: $\vdash$ $(inf \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (f \wedge fmore);f^\omega)) \longrightarrow$
$inf \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (g \wedge fmore);f^\omega)$
using *3* **by** *fastforce*
have *5*: $\vdash$ $inf \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (g \wedge fmore);f^\omega) \longrightarrow g^\omega$
using *OmegaInduct* **by** *blast*
have *6*: $\vdash$ $f^\omega \longrightarrow (f \wedge fmore);f^\omega$
by (*simp add*: *OmegaUnroll int-iffD1*)
have *7*: $\vdash\Box(\ f^\omega \longrightarrow (f \wedge fmore);f^\omega)$
using *6* **by** (*simp add*: *BoxGen*)
from *3 5 7* **show** *?thesis* **by** *fastforce*
qed

lemma *OmegaEqvRule*:
assumes $\vdash$ $f = g$
shows $\vdash$ $f^\omega = g^\omega$
using *assms* **using** *int-eq* **by** *force*

lemma *AndOmegaA*:
$\vdash (f \wedge g)^{\omega} \wedge inf \longrightarrow f^{\omega}$
by (*meson OmegaImpRule Prop12 int-iffD2 lift-and-com*)

lemma *AndOmegaB*:
$\vdash (f \wedge g)^{\omega} \wedge inf \longrightarrow g^{\omega}$
by (*meson OmegaImpRule Prop12 int-iffD2 lift-and-com*)

lemma *BaOmegaImpOmega*:
$\vdash ba\ (f \longrightarrow g) \wedge inf \longrightarrow f^{\omega} \longrightarrow g^{\omega}$
proof −
have *1*: $\vdash ba\ (f \longrightarrow g) \wedge (f \wedge fmore);f^{\omega} \longrightarrow ((f \longrightarrow g) \wedge (f \wedge fmore)\ );((f \longrightarrow g) \wedge f^{\omega})$
using *BaAndChopImport* **by** *fastforce*
have *2*: $\vdash (f \longrightarrow g) \wedge (f \wedge fmore) \longrightarrow (\ g \wedge fmore)$
by *auto*
have *3*: $\vdash (f \longrightarrow g) \wedge f^{\omega} \longrightarrow f^{\omega}$
by *auto*
have *4*: $\vdash ba\ (f \longrightarrow g) \wedge (f \wedge fmore);f^{\omega} \longrightarrow (g \wedge fmore);\ f^{\omega}$
using *1 2 3*
by (*metis* (*no-types*, *lifting*) *AndChopB ChopAndB Prop10 int-eq lift-imp-trans*)
have *5*: $\vdash ba\ (f \longrightarrow g) \wedge (f \wedge fmore);f^{\omega} \longrightarrow ((f \longrightarrow g) \wedge (f \wedge fmore));(ba(f \longrightarrow g) \wedge f^{\omega})$
using *BaAndChopImportB* **by** *blast*
have *6*: $\vdash ((f \longrightarrow g) \wedge (f \wedge fmore));(ba(f \longrightarrow g) \wedge f^{\omega}) \longrightarrow$
$((g \wedge fmore));(ba(f \longrightarrow g) \wedge f^{\omega})$
using *2 LeftChopImpChop* **by** *blast*
have *7*: $\vdash (ba\ (f \longrightarrow g) \wedge f^{\omega}) \longrightarrow (g \wedge fmore);(ba\ (f \longrightarrow g) \wedge f^{\omega})$
using *OmegaUnroll 5 6* **by** *fastforce*
have *8*: $\vdash (ba\ (f \longrightarrow g) \wedge f^{\omega}) \wedge inf \longrightarrow g^{\omega}$
using *7 OmegaIntro* **by** *blast*
from *8* **show** *?thesis* **by** *fastforce*
qed

lemma *BaOmegaEqvOmega*:
$\vdash ba\ (f = g) \wedge inf \longrightarrow (f^{\omega} = g^{\omega})$
proof −
have *1*: $\vdash ba\ (f = g) = (ba\ (f \longrightarrow g) \wedge ba\ (g \longrightarrow f))$ **by** (*auto simp*: *ba-defs sum.case-eq-if*)
have *2*: $\vdash ba\ (f \longrightarrow g) \wedge inf \longrightarrow (f^{\omega} \longrightarrow g^{\omega})$ **using** *BaOmegaImpOmega* **by** *blast*
have *3*: $\vdash ba\ (g \longrightarrow f) \wedge inf \longrightarrow (g^{\omega} \longrightarrow f^{\omega})$ **using** *BaOmegaImpOmega* **by** *blast*
have *4*: $\vdash ba\ (f = g) \wedge inf \longrightarrow (f^{\omega} \longrightarrow g^{\omega}) \wedge (g^{\omega} \longrightarrow f^{\omega})$ **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash ((f^{\omega} \longrightarrow g^{\omega}) \wedge (g^{\omega} \longrightarrow f^{\omega})) = (f^{\omega} = g^{\omega})$ **by** *auto*
from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *BaAndOmegaImport*:
$\vdash ba\ f \wedge g^{\omega} \wedge inf \longrightarrow (f \wedge g)^{\omega}$
proof −
have *1*: $\vdash f \longrightarrow (g \longrightarrow (f \wedge g))$ **by** *auto*
hence *2*: $\vdash ba\ f \longrightarrow ba\ (g \longrightarrow f \wedge g)$ **by** (*rule BaImpBa*)
have *3*: $\vdash\ ba\ \ (g \longrightarrow f \wedge g) \wedge inf \longrightarrow g^{\omega} \longrightarrow (f \wedge g)^{\omega}$ **by** (*rule BaOmegaImpOmega*)

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

13.10 Properties of While

lemma *WhileEqvIf*:
$\vdash ((while\ \ (init\ w)\ \ do\ \ f) \wedge finite) =$
$\quad (\ if_i\ \ (init\ w)\ \ then\ \ (f;\ (\ while\ \ (init\ w)\ \ do\ \ f))\ \ else\ \ \ empty\ \wedge finite)$
proof −
have 1: $\vdash\ (while\ \ (init\ w)\ \ do\ \ f \wedge finite) =$
$\quad ((((init\ w) \wedge f)^{\star} \wedge\ fin\ (\neg\ \ (init\ w))) \wedge finite)$
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: $\vdash (((init\ w) \wedge f)^{\star} \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite) =$
$\quad ((empty\ \ \vee\ \ ((init\ w \wedge f);\ (init\ w \wedge f)^{\star})) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite)$
using *2* **by** *fastforce*
have 22: $\vdash ((empty\ \ \vee\ \ ((init\ w \wedge f);\ (init\ w \wedge f)^{\star})) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite) =$
$\quad ((\ empty \wedge fin\ (\neg(init\ w)) \wedge finite) \vee$
$\quad (\ ((init\ w \wedge f);\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite))$
by *auto*
have 23: $\vdash ((((init\ w) \wedge f)^{\star} \wedge\ \ fin\ (\neg\ \ (init\ w))) \wedge finite) =$
$\quad ((\ empty \wedge fin\ (\neg(init\ w)) \wedge finite) \vee$
$\quad (\ ((init\ w \wedge f);\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite))$
using *21 22* **by** *auto*
have 3: $\vdash\ \ (empty\ \ \wedge\ \ fin\ (\neg\ (\ init\ w))) = (\neg\ (\ init\ w) \wedge\ \ empty)$
by (*metis FinAndEmpty Prop04 lift-and-com*)
hence 31: $\vdash\ \ (empty\ \ \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite) = (\neg\ (\ init\ w) \wedge\ \ empty \wedge finite)$
by *auto*
have 32: $\vdash (\neg\ (\ init\ w) \wedge\ \ empty \wedge finite) = (\neg\ (\ init\ w) \wedge\ \ empty)$
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have 33: $\vdash (\ empty \wedge fin\ (\neg(init\ w)) \wedge finite) = (\neg\ (\ init\ w) \wedge\ \ empty)$
using *31 32* **by** *fastforce*
have 34: $\vdash ((\ empty \wedge fin\ (\neg(init\ w)) \wedge finite) \vee$
$\quad (\ ((init\ w \wedge f);\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite)) =$
$\quad ((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$\quad (\ ((init\ w \wedge f);\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite))$
using *23 33* **by** *fastforce*
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: $\vdash (((init\ w \wedge f);\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite) =$
$\quad (init\ w \wedge (f;\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite)$
using *4* **by** *auto*
have 42: $\vdash (init\ w \wedge ((f;\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite)) =$
$\quad (init\ w \wedge ((f;\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ \ \ (init\ (\neg\ w)) \wedge finite))$
using *Initprop*(*2*) **by** (*metis StateAndEmptyChop int-eq*)
have 5: $\vdash ((f;\ ((init\ w \wedge f)^{\star})) \wedge\ \ (fin\ \ (\ init\ (\neg\ w))) \wedge finite)$
$\quad = ((f\ \wedge finite);\ ((init\ w \wedge f)^{\star} \wedge\ \ (fin\ \ (\ init\ (\neg\ w))) \wedge finite))$
using *ChopAndFin* **by** *fastforce*
hence 49: $\vdash (init\ w \wedge (f;\ (init\ w \wedge f)^{\star}) \wedge\ \ fin\ (\neg\ \ (init\ w)) \wedge finite\ ) =$

$(init\ \ w \wedge (f \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ (fin\ \ (\ init\ (\neg\ w))) \wedge finite))$
using *42* **by** *fastforce*
have *50*: $\vdash\ (((init\ w \wedge f); (init\ w \wedge f)^{\star}) \wedge\ fin\ (\neg\ \ (init\ w)) \wedge finite) =$
$(init\ \ w \wedge (f \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ (fin\ \ (\ init\ (\neg\ w))) \wedge finite))$
using *49 41* **by** *fastforce*
have *51*: $\vdash (init\ \ w \wedge (f \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ (fin\ \ (\ init\ (\neg\ w))) \wedge finite\ )) =$
$(init\ \ w \wedge (f \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ (fin\ (\neg\ (\ init\ \ w))) \wedge finite))$
using *Initprop*(*2*)
by (*metis int-simps*(*1*) *inteq-reflection*)
have *52*: $\vdash (((init\ w \wedge f); (init\ w \wedge f)^{\star}) \wedge\ fin\ (\neg\ \ (init\ w)) \wedge finite) =$
$(init\ w \wedge ((f \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ fin\ (\neg\ (\ init\ \ w)) \wedge finite))\ )$
using *50 51* **by** *fastforce*
have *53*: $\vdash ((\neg\ (\ init\ w) \wedge\ empty) \vee$
$(\ ((init\ w \wedge f); (init\ w \wedge f)^{\star}) \wedge\ fin\ (\neg\ \ (init\ w)) \wedge finite)) =$
$((\neg\ (\ init\ w) \wedge\ empty) \vee$
$(init\ w \wedge ((f \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ fin\ (\neg\ (\ init\ \ w)) \wedge finite))\ ))$
using *52 34* **by** *auto*
have *6*: $\vdash ((f \wedge finite); (((init\ w \wedge f)^{\star} \wedge\ fin\ (\neg\ (\ init\ w))) \wedge finite)) =$
$(f \wedge finite);\ (while\ \ (init\ w)\ \ do\ \ f \wedge finite)$
by (*simp add*: *while-d-def*)
have *61*: $\vdash (init\ w \wedge ((f \wedge finite); (((init\ w \wedge f)^{\star} \wedge\ fin\ (\neg\ (\ init\ \ w))) \wedge finite))) =$
$(init\ w \wedge\ ((f \wedge finite);\ (while\ \ (init\ w)\ \ do\ \ f \wedge finite)))$
using *6*
by *auto*
have *62*: $\vdash ((\neg\ (\ init\ w) \wedge\ empty) \vee$
$(init\ w \wedge ((f \wedge finite); (((init\ w \wedge f)^{\star} \wedge\ fin\ (\neg\ (\ init\ \ w))) \wedge finite))\ ))$
$=((\neg\ (\ init\ w) \wedge\ empty\ ) \vee$
$(init\ w \wedge ((f \wedge finite);(while\ \ (init\ w)\ \ do\ \ f \wedge finite))\ )\ )$
using *61* **by** *fastforce*
have *7*: $\vdash\ (while\ \ (init\ w)\ \ do\ \ f\ \ \wedge finite)$
$= (((\neg\ (\ init\ w) \wedge\ empty\ ) \vee$
$(init\ w \wedge (f\ ;\ \ while\ \ (init\ w)\ \ do\ \ f) \wedge finite)))$
using *1 23 34 53 62*
by (*metis 2 22 ChopAndFiniteDist inteq-reflection*)
have *71*: $\vdash ((if_i\ \ (init\ w)\ then\ (f; (\ while\ \ (init\ w)\ \ do\ \ f))\ \ else\ \ \ empty)\ \wedge finite)=$
$(((\neg\ (\ init\ w) \wedge\ empty\ ) \vee\ (init\ w \wedge (f;\ \ while\ \ (init\ w)\ \ do\ \ f)) \wedge finite\ ))$
using *FiniteAndEmptyEqvEmpty* **by** (*auto simp*: *ifthenelse-d-def*)
from *7 71* **show** *?thesis* **by** *fastforce*
qed

lemma *IfAndFiniteDist*:
$\vdash (if_i\ (init\ w)\ then\ (f;g)\ \ else\ \ empty \wedge finite) =$
$(if_i\ (init\ w)\ then\ ((f \wedge finite);(g \wedge finite))\ \ else\ \ empty)$
proof −
have *1*: $\vdash (if_i\ (init\ w)\ then\ (f;g)\ \ else\ \ empty \wedge finite) =$
$((\ (init\ w \wedge (f;g)) \vee (\neg(init\ w) \wedge empty)) \wedge finite)$
by (*auto simp*: *ifthenelse-d-def*)
have *2*: $\vdash ((\ (init\ w \wedge (f;g)) \vee (\neg(init\ w) \wedge empty)) \wedge finite) =$
$((\ (init\ w \wedge (f;g) \wedge finite) \vee (\neg(init\ w) \wedge empty \wedge finite)))$
by *auto*

have *3*: $\vdash (init\ w \wedge (f;g) \wedge finite) = (init\ w \wedge (f \wedge finite);(g \wedge finite))$
using *ChopAndFiniteDist* **by** *fastforce*
have *4*: $\vdash (\neg(init\ w) \wedge empty \wedge finite) = (\neg(init\ w) \wedge empty)$
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have *5*: $\vdash ((\ (init\ w \wedge (f;g) \wedge finite) \vee (\neg(init\ w) \wedge empty \wedge finite))) =$
$((init\ w \wedge (f \wedge finite);(g \wedge finite)) \vee (\neg(init\ w) \wedge empty))$
using *3 4* **by** *fastforce*
have *6*: $\vdash ((init\ w \wedge (f \wedge finite);(g \wedge finite)) \vee (\neg(init\ w) \wedge empty)) =$
$(if_i\ (init\ w)\ then\ ((f \wedge finite);(g \wedge finite))\ \ else\ \ empty)$
by (*auto simp*: *ifthenelse-d-def*)
from *1 2 5 6* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *WhileChopEqvIf*:
$\vdash\ \ (\ (while\ (\ init\ w)\ do\ f) \wedge finite);\ g =$
$if_i\ (init\ w)\ then\ ((f \wedge finite);\ (\ ((while\ (\ init\ w)\ do\ f) \wedge finite)\ ;\ g))\ \ else\ \ g$
proof −
have *1*: $\vdash\ (while\ \ (init\ w)\ \ do\ \ f \wedge finite)=$
$(if_i\ \ (init\ w)\ \ then\ \ (f;\ (\ while\ \ (init\ w)\ \ do\ \ f))\ \ else\ \ \ empty\ \ \wedge finite)$
by (*rule WhileEqvIf*)
have *11*: $\vdash (if_i\ \ (init\ w)\ \ then\ \ (f;\ (\ while\ \ (init\ w)\ \ do\ \ f))\ \ else\ \ \ empty\ \ \wedge finite) =$
$(if_i\ \ (init\ w)\ then\ \ ((f \wedge finite);\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite))\ \ else\ \ \ empty)$
using *IfAndFiniteDist* **by** *fastforce*
have *12*: $\vdash (while\ \ (init\ w)\ \ do\ \ f \wedge finite) =$
$(if_i\ \ (init\ w)\ then\ \ ((f \wedge finite);\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite))\ \ else\ \ \ empty)$
using *1 11* **by** *fastforce*
hence *2*: $\vdash (\ (while\ \ (init\ w)\ \ do\ \ f) \wedge finite);\ g =$
$(if_i\ \ (init\ w)$
$then\ (((f \wedge finite);\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite));g)$
$else\ \ \ (empty;g))$
by (*rule IfChopEqvRule*)
have *3*: $\vdash\ \ empty\ ;\ g = g$
by (*rule EmptyChop*)
have *4*: $\vdash (if_i\ \ (init\ w)$
$then\ (((f \wedge finite);\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite));g)$
$else\ \ \ (empty;g))\ \ =$
$(if_i\ \ (init\ w)$
$then\ (((f \wedge finite);\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite));g)$
$else\ \ \ g)$
using *3* **using** *inteq-reflection* **by** *fastforce*
have *5*: $\vdash (((f \wedge finite);\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite));g) =$
$((f \wedge finite);(\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite);g))$
by (*rule ChopAssocB*)
have *6*: $\vdash (if_i\ \ (init\ w)$
$then\ (((f \wedge finite);\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite));g)$
$else\ \ \ g) =$
$(if_i\ \ (init\ w)$
$then\ ((f \wedge finite);(\ (\ while\ \ (init\ w)\ \ do\ \ f \wedge finite);g))$
$else\ \ \ g)$

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* $g \wedge finite); h$
shows $\vdash f =$ if_i $(init\ w)$ *then* $((g \wedge finite); f)$ *else* h
proof −
have *1*: $\vdash f = ($ *while* $(init\ w)$ *do* $g \wedge finite); h$
using *assms* **by** *auto*
have *2*: $\vdash ($ *while* $(init\ w)$ *do* $g \wedge finite); h =$
if_i $(init\ w)$ *then* $((g \wedge finite); (($ *while* $(init\ w)$ *do* $g \wedge finite); h))$ *else* h
by (*rule WhileChopEqvIf*)
have *3*: $\vdash ((g \wedge finite); f) = ((g \wedge finite); (($ *while* $(init\ w)$ *do* $g \wedge finite); h))$
using *1* **by** (*rule RightChopEqvChop*)
have *4*: $\vdash ((g \wedge finite); (($ *while* $(init\ w)$ *do* $g \wedge finite); h)) = ((g \wedge finite); f)$
using *3* **by** *auto*
have *5*: $\vdash$ if_i $(init\ w)$ *then* $((g \wedge finite); (($ *while* $(init\ w)$ *do* $g \wedge finite); h))$ *else* $h =$
if_i $(init\ w)$ *then* $((g \wedge finite); f)$ *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* $f \wedge finite)=$
$((\neg\ (init\ w) \wedge empty) \vee (init\ w \wedge ((f \wedge more) \wedge finite);($*while* $(init\ w)$ *do* $f \wedge finite)))$
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*: $\vdash ((init\ w \wedge f)^\star \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$
$((empty \wedge fin\ (\neg\ (init\ w)) \wedge finite) \vee$
$((init\ w \wedge f \wedge more); (init\ w \wedge f)^\star \wedge fin\ (\neg\ (init\ w)) \wedge finite))$
using *1 4* **by** *fastforce*
have *51*: $\vdash ((empty \wedge fin\ (\neg\ (init\ w))) \wedge finite) = ((\neg\ (init\ w) \wedge empty) \wedge finite)$

by (*metis FinAndEmpty inteq-reflection lift-and-com*)
have *52*:$\vdash$ $((\neg\ (\ init\ w) \wedge\ \ empty) \wedge finite) = (\neg\ (\ init\ w) \wedge\ \ empty)$
using *EmptyImpFinite* **by** *auto*
have *6*: $\vdash$ $((empty\ \ \wedge\ \ fin\ \ (\neg\ (\ init\ w))) \wedge finite) = (\neg\ (\ init\ w) \wedge\ \ empty)$
using *51 52* **by** *fastforce*
have *61*: $\vdash$ $((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ =$
$((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$((init\ w \wedge f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} \wedge fin\ (\neg\ (\ init\ w)) \wedge finite))$
using *5 6* **by** *fastforce*
have *70*: $\vdash$ $(init\ w \wedge f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} = (init\ w \wedge (f \wedge\ \ more\ ); (init\ w \wedge f)^{\star})$
by (*rule StateAndChop*)
have *7*: $\vdash$ $((init\ w \wedge f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} \wedge fin\ \ (\ init\ (\neg\ w)) \wedge finite) =$
$(init\ w \wedge (f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} \wedge fin\ \ (\ init\ (\neg\ w)) \wedge finite)$
using *70* **by** *auto*
have *71*: $\vdash$ $((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$((init\ w \wedge f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} \wedge fin\ (\neg\ (\ init\ w)) \wedge finite)) =$
$((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$(init\ w \wedge (f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} \wedge fin\ \ (\ init\ (\neg\ w)) \wedge finite)\ )$
using *7*
by (*metis* (*no-types*, *hide-lams*) *ChopEmpty Initprop*(*2*) *inteq-reflection*)
have *8*: $\vdash$ $((\ (f \wedge\ \ more\ ); (init\ w \wedge f)^{\star}) \wedge\ \ fin\ \ (\ init\ (\neg\ w)) \wedge finite)\ =$
$(((f \wedge\ \ more) \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\ \ init\ (\neg\ w)) \wedge finite))$
using *ChopAndFin* **by** *fastforce*
have *81*: $\vdash$ $fin\ (\ \ init\ (\neg\ w)) = fin\ (\neg\ (\ init\ w))$
using *FinEqvFin Initprop*(*2*) **by** *fastforce*
have *82*: $\vdash$ $((f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite) =$
$(((f \wedge\ \ more) \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ )$
using *8 81*
by (*metis inteq-reflection*)
have *83*: $\vdash$ $(init\ w \wedge (f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite) =$
$(init\ w \wedge ((f \wedge\ \ more) \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ )$
using *82* **by** *fastforce*
have *84*: $\vdash$ $((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$(init\ w \wedge (f \wedge\ \ more\ ); (init\ w \wedge f)^{\star} \wedge fin\ \ (\ init\ (\neg\ w)) \wedge finite)\ )\ =$
$((\neg\ (\ init\ w) \wedge\ \ empty) \vee$
$(init\ w \wedge ((f \wedge\ \ more) \wedge finite); ((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ )\ )$
using *83* **by** (*metis 71 81 inteq-reflection*)
have *9*: $\vdash$ $((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ =$
$((\neg\ (\ init\ w) \wedge\ \ empty\ ) \vee$
$(init\ w \wedge ((f \wedge\ \ more) \wedge finite);$
$((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ )\ )$
using *84 61* **by** (*metis 71 inteq-reflection*)
have *10*: $\vdash$ $((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite) =$
$(((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w))) \wedge finite)$
by *auto*
hence *11*: $\vdash$ $((\neg\ (\ init\ w) \wedge\ \ empty\ ) \vee$
$(init\ w \wedge ((f \wedge\ \ more) \wedge finite);$
$((init\ w \wedge f)^{\star} \wedge\ \ fin\ (\neg\ (\ init\ w)) \wedge finite)\ )\ ) =$
$((\neg\ (\ init\ w) \wedge\ \ empty\ ) \vee$
$(init\ w \wedge ((f \wedge\ \ more) \wedge finite);$

```
                    (((init w ∧ f)⋆ ∧  fin (¬ ( init w))) ∧ finite) ) )
    by (metis 84 inteq-reflection)
 have 12: ⊢ ((init w ∧ f)⋆ ∧  fin (¬ ( init w)) ∧ finite)  =
          ((¬ ( init w) ∧  empty ) ∨
            (init w ∧ ((f ∧  more) ∧ finite);
                   (((init w ∧ f)⋆ ∧  fin (¬ ( init w))) ∧ finite) ) )
     using 11 9 by fastforce
 from 12 show ?thesis by (metis 10 inteq-reflection while-d-def )
qed
```

```
lemma WhileIntro:
 assumes ⊢  ¬ ( init w) ∧ f ⟶  empty
        ⊢ init w ∧ f ⟶ ((g ∧  more) ∧ finite ); f
 shows  ⊢ f ∧ finite ⟶  while ( init w) do  g
proof −
 have  1: ⊢  ¬ ( init w) ∧ f ⟶  empty
     using assms by blast
 have  2: ⊢ init w ∧ f ⟶ ((g ∧  more) ∧ finite ); f
     using assms by blast
 have  3: ⊢  (while ( init w)  do  g ∧ finite)=
         ((¬  (init w) ∧  empty ) ∨
          (init w ∧ ((g ∧  more) ∧ finite ); (while  (init w)  do  g ∧ finite)))
     by (rule WhileEqvEmptyOrChopWhile)
 hence 31: ⊢ ¬ ( while  (init w)  do  g ∧ finite) =
         (¬( (¬  (init w) ∧  empty ) ∨
            (init w ∧ ((g ∧  more) ∧ finite );  (while  (init w)  do  g ∧ finite))))
     by fastforce
 hence 32: ⊢ (f ∧ ¬ ( while  (init w)  do  g ∧ finite)) =
           (f ∧ ¬( (¬(init w) ∧ empty) ∨
                  (init w ∧ ((g ∧  more) ∧ finite );  (while  (init w)  do  g ∧ finite))))
     by fastforce
 have  33: ⊢ (f ∧ ¬( (¬(init w) ∧ empty) ∨
                    (init w ∧ ((g ∧ more) ∧ finite);  (while  (init w)  do  g ∧ finite)))) =
           (f ∧ ¬(¬(init w) ∧  empty ) ∧
                 ¬(init w ∧ ((g ∧  more) ∧ finite );  (while  (init w)  do  g ∧ finite)))
     by auto
 have  34: ⊢ (f ∧ ¬(¬(init w) ∧ empty) ∧
                 ¬((init w) ∧ (((g ∧  more) ∧ finite );  (while  (init w)  do  g ∧ finite)))) =
            (f ∧  ( (init w) ∨ more ) ∧
                  (¬(init w) ∨ ¬(((g ∧ more) ∧ finite); (while (init w)  do  g ∧ finite))))
     by (auto simp: empty-d-def )
 have  35: ⊢ (f ∧ ((init w) ∨ more) ∧
               (¬(init w) ∨ ¬(((g ∧ more) ∧ finite);(while (init w) do g ∧ finite)))) =
           ((f ∧ (init w) ∧ ¬ (((g ∧  more) ∧ finite );  (while ( init w) do  g ∧ finite))) ∨
           (f ∧ (init w) ∧ ¬(init w)) ∨
           (f ∧ more ∧ ¬ (((g ∧  more) ∧ finite );  (while ( init w) do  g ∧ finite))) ∨
           (f ∧ more ∧ ¬(init w)))
     by auto
```

```
 have 36: ⊢ (f ∧ ¬ ( while (init w) do g ∧ finite)) =
           ((f ∧ (init w) ∧ ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite))) ∨
           (f ∧ (init w) ∧ ¬(init w)) ∨
           (f ∧ more ∧ ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite))) ∨
           (f ∧ more ∧ ¬(init w))) using 32 33 34 35 by fastforce
 have 37: ⊢ ¬(f ∧ more ∧ ¬(init w))
      using 1 by (auto simp: empty-d-def)
 have 38: ⊢ (f ∧ more ∧ ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite))) ⟶
          (((g ∧ more) ∧ finite ); f ∧
            ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite)))
      using 1 2 by (auto simp: empty-d-def Valid-def)
 have 39: ⊢ (f ∧ (init w) ∧ ¬ (((g ∧ more)∧ finite ); (while ( init w) do g ∧ finite))) ⟶
          (((g ∧ more) ∧ finite ); f ∧
            ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite)))
      using 2 by auto
 have 40: ⊢ ((f ∧ (init w) ∧ ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite))) ∨
           (f ∧ (init w) ∧ ¬(init w)) ∨
           (f ∧ more ∧ ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite))) ∨
           (f ∧ more ∧ ¬(init w))) ⟶
           ((g ∧ more) ∧ finite ); f ∧
             ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite))
       using 39 38 37 38 by fastforce
 have  4: ⊢ f ∧ ¬ ( while (init w) do g ∧ finite) ⟶
           ((g ∧ more) ∧ finite ); f ∧
            ¬ (((g ∧ more) ∧ finite ); (while ( init w) do g ∧ finite))
       using 36 40 by fastforce
 have  50: ⊢ g ∧ more ⟶ more
       by auto
 have  5: ⊢ (g ∧ more) ∧ finite ⟶ more
  by (simp add: 50 Prop05 Prop07 finite-d-def)
 have  6: ⊢ f ∧ finite ⟶ ( while (init w) do g ∧ finite)
     using 4 5 ChopContraB by blast
 from 6 show ?thesis by (simp add: Prop12)
qed


lemma WhileElim:
 assumes ⊢ ¬ ( init w) ∧ empty ⟶ g
        ⊢ init w ∧ ((f ∧ more) ∧ finite ); g ⟶ g
 shows ⊢ while (init w) do f ∧ finite ⟶ g
proof −
 have  1: ⊢ (while ( init w) do f ∧ finite)=
          ((¬ ( init w) ∧ empty ) ∨
            (init w ∧ ((f ∧ more) ∧ finite ); (while ( init w) do f ∧ finite)))
       by (rule WhileEqvEmptyOrChopWhile)
 hence 11: ⊢ ((while ( init w) do f ∧ finite) ∧ ¬ g) =
          (((¬(init w) ∧ empty) ∨
            (init w ∧ ((f ∧ more) ∧ finite);(while (init w) do f ∧ finite))) ∧ ¬ g)
      by auto
```

have *2*: $\vdash \neg$ (*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*) $\wedge$ *finite*);(*while* (*init w*) *do f* $\wedge$ *finite*))) $\wedge \neg$ *g* $\longrightarrow$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do f* $\wedge$ *finite*))
using *21* **by** *auto*
have *23*: $\vdash$ (*while* (*init w*) *do f* $\wedge$ *finite*) $\wedge \neg$ *g* $\longrightarrow$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do f* $\wedge$ *finite*)) $\wedge \neg$ *g*
using *11 21* **by** *fastforce*
have *3*: $\vdash$ (*init w*) $\wedge$ (((*f* $\wedge$ *more*) $\wedge$ *finite*); *g*) $\longrightarrow$ *g*
using *assms* **by** *blast*
hence *31*: $\vdash \neg$ *g* $\longrightarrow \neg$((*init w*) $\wedge$ (((*f* $\wedge$ *more*) $\wedge$ *finite*); *g*))
by *fastforce*
have *32*: $\vdash$ (*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do f* $\wedge$ *finite*)) $\wedge \neg$ *g* $\longrightarrow$
((((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do f* $\wedge$ *finite*)) $\wedge$
$\neg$ (((*f* $\wedge$ *more*) $\wedge$ *finite*); *g*)) $\wedge \neg$*g*
using *31* **by** *auto*
have *4*: $\vdash$ (*while* (*init w*) *do f* $\wedge$ *finite*) $\wedge \neg$ *g* $\longrightarrow$
(((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do f* $\wedge$ *finite*)) $\wedge$
$\neg$ (((*f* $\wedge$ *more*) $\wedge$ *finite*); *g*)
using *23 32* **by** *fastforce*
have *5*: $\vdash$ (*f* $\wedge$ *more*) $\wedge$ *finite* $\longrightarrow$ *more*
by *auto*
from *4 5* **show** *?thesis* **using**
ChopContraB[*of LIFT*(*while* (*init w*) *do f* $\wedge$ *finite*) *LIFT*(*g*) *LIFT*(((*f* $\wedge$ *more*) $\wedge$ *finite*))]
by *auto*
qed

lemma *BaWhileImpWhile*:
$\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ *finite* $\longrightarrow$ (*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*))
using *BaImpBa* **by** *blast*
have *3*: $\vdash$ *ba* ((*init w* $\wedge$ *f*) $\longrightarrow$ (*init w* $\wedge$ *g*)) $\wedge$ *finite* $\longrightarrow$ ((*init w* $\wedge$ *f*)* $\longrightarrow$ (*init w* $\wedge$ *g*)*)
by (*rule BaCSImpCS*)
have *4*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ *finite* $\longrightarrow$ ((*init w* $\wedge$ *f*)$^\star \wedge$ *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 $\vdash (\ \mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ f) \land \mathit{finite} \longrightarrow (\ \mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ g)$
proof −
have *1*: $\vdash f \longrightarrow g$
using *assms* **by** *auto*
hence *2*: $\vdash\ \ \mathit{ba}\ (f \longrightarrow g)$
by (*rule BaGen*)
have *3*: $\vdash\ \ \mathit{ba}\ (f \longrightarrow g) \land \mathit{finite} \longrightarrow (\ \mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ f) \longrightarrow (\ \mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ g)$
by (*rule BaWhileImpWhile*)
have *4*: $\vdash \mathit{ba}\ (f \longrightarrow g)\ \ \longrightarrow (\ \mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ f \land \mathit{finite}) \longrightarrow (\ \mathit{while}\ \ (\mathit{init}\ w)\ \ \mathit{do}\ \ g)$
using *3* **by** (*auto simp*: *Valid-def*)
from *2 4* **show** *?thesis* **using** *MP* **by** *blast*
qed

13.11 Properties of Halt

lemma *WnextAndMoreEqvNext*:
$\vdash (\mathit{wnext}\ f \land \mathit{more}) = \bigcirc\ f$
by (*auto simp*: *wnext-defs more-defs next-defs sum.case-eq-if*)

lemma *BoxStateAndEmptyEqvStateAndEmpty*:
$\vdash (\Box(\mathit{empty} = (\mathit{init}\ w)) \land \mathit{empty}) = ((\mathit{init}\ w) \land \mathit{empty})$
by (*auto simp*: *always-defs init-defs empty-defs sum.case-eq-if*)

lemma *BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*:
$\vdash \Box(\mathit{empty} = (\mathit{init}\ w)) = ((\mathit{empty} \land \mathit{init}\ w) \lor (\neg(\mathit{init}\ w) \land \bigcirc(\Box(\mathit{empty} = (\mathit{init}\ w)))))$
proof −
have *1*: $\vdash \Box(\mathit{empty} = (\mathit{init}\ w)) =$
$((\Box(\mathit{empty} = (\mathit{init}\ w)) \land \mathit{empty}) \lor (\Box(\mathit{empty} = (\mathit{init}\ w)) \land \mathit{more}))$
by (*auto simp*: *empty-d-def*)
have *2*: $\vdash (\Box(\mathit{empty} = (\mathit{init}\ w)) \land \mathit{empty}) = ((\mathit{init}\ w) \land \mathit{empty})$
using *BoxStateAndEmptyEqvStateAndEmpty* **by** *blast*
have *3*: $\vdash \Box(\mathit{empty} = (\mathit{init}\ w))\ = ((\mathit{empty} = (\mathit{init}\ w)) \land \mathit{wnext}(\Box(\mathit{empty} = (\mathit{init}\ w))))$
using *BoxEqvAndWnextBox* **by** *blast*
hence *4*: $\vdash (\Box(\mathit{empty} = (\mathit{init}\ w)) \land \mathit{more})\ =$
$(((\mathit{empty} = (\mathit{init}\ w)) \land \mathit{more}) \land (\mathit{wnext}(\Box(\mathit{empty} = (\mathit{init}\ w))) \land \mathit{more}))$
by *auto*
have *5*: $\vdash ((\mathit{empty} = (\mathit{init}\ w)) \land \mathit{more}) = (\neg(\mathit{init}\ w) \land \mathit{more})$
by (*auto simp*: *empty-d-def*)
have *6*: $\vdash (\mathit{wnext}(\Box(\mathit{empty} = (\mathit{init}\ w))) \land \mathit{more}) = \bigcirc(\Box(\mathit{empty} = (\mathit{init}\ w)))$
using *WnextAndMoreEqvNext* **by** *metis*
have *7*: $\vdash (\Box(\mathit{empty} = (\mathit{init}\ w)) \land \mathit{more}) =$
$((\neg\ (\mathit{init}\ w) \land \mathit{more}) \land (\mathit{wnext}(\Box(\mathit{empty} = (\mathit{init}\ w))) \land \mathit{more})\ )$
using *4 5* **by** *fastforce*
have *8*: $\vdash ((\neg\ (\mathit{init}\ w) \land \mathit{more}) \land (\mathit{wnext}(\Box(\mathit{empty} = (\mathit{init}\ w))) \land \mathit{more})\ ) =$
$((\neg\ (\mathit{init}\ w)) \land (\mathit{wnext}(\Box(\mathit{empty} = (\mathit{init}\ w))) \land \mathit{more})\ )$ **by** *auto*
have *9*: $\vdash ((\neg\ (\mathit{init}\ w)) \land (\mathit{wnext}(\Box(\mathit{empty} = (\mathit{init}\ w))) \land \mathit{more})\ ) =$
$((\neg\ (\mathit{init}\ w)) \land \bigcirc(\Box(\mathit{empty} = (\mathit{init}\ w))))$ **using** *8 6* **by** *auto*
have *10*: $\vdash \Box(\mathit{empty} = (\mathit{init}\ w)) = (((\mathit{init}\ w) \land \mathit{empty}) \lor (\Box(\mathit{empty} = (\mathit{init}\ w)) \land \mathit{more})\ )$
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*: ⊢ *halt* (*init w*); *f* = *if*$_i$ (*init w*) *then* *f* *else* (○(*halt* (*init w*); *f*))
by (*rule HaltChopEqv*)
have *2*: ⊢ ¬ (*init w*) ∧ *if*$_i$ (*init w*) *then* *f* *else* (○(*halt* (*init w*); *f*)) ⟶
○(*halt* (*init w*); *f*)
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotAndHaltChopImpSkipYields*:
⊢ ¬ (*init w*) ∧ (*halt* (*init w*); *f*) ⟶ *skip* *yields* (*halt* (*init w*); *f*)
proof −
have *1*: ⊢ ¬ (*init w*) ∧ (*halt* (*init w*); *f*) ⟶ ○(*halt* (*init w*); *f*)
by (*rule NotAndHaltChopImpNext*)
have *2*: ⊢ ○(*halt* (*init w*); *f*) ⟶ *skip* *yields* (*halt* (*init w*); *f*)
by (*rule NextImpSkipYields*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteChopAndEmptyEqvChopAndEmpty*:
⊢ ((*finite*;(*f* ∧ *empty*)) ∧ *g*) = ((*g* ∧ *finite*);(*f*∧ *empty*))
proof −
have *1*: ⊢ *g* ∧ *finite*;(*f* ∧ *empty*) ⟶ *fin f*
by (*metis ChopAndA DiamondFin FinAndEmpty Prop01 Prop05 inteq-reflection sometimes-d-def*)
have *2*: ⊢ *g* ∧ *finite*;(*f* ∧ *empty*) ⟶ (*finite* ∧ *g*) ∧ *fin f*
using *1* **by** (*metis* (*no-types*, *lifting*) *ChopAndB ChopEmpty Prop10 Prop12 int-iffD1*
inteq-reflection)
have *3*: ⊢ ((*finite*;(*f* ∧ *empty*)) ∧ *g*) ⟶ ((*g* ∧ *finite*);(*f*∧ *empty*))
using *2* **using** *AndFinEqvChopAndEmpty* **by** *fastforce*
have *4*: ⊢ ((*g* ∧ *finite*);(*f*∧ *empty*)) ⟶ ((*finite*;(*f* ∧ *empty*)) ∧ *g*)
by (*metis AndChopB ChopAndB ChopEmpty Prop12 inteq-reflection*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *WprevEqvEmptyOrPrev*:
⊢ *wprev f* = (*empty* ∨ *prev f*)
by (*auto simp*: *wprev-defs empty-defs prev-defs sum.case-eq-if*)

lemma *NotChopSkipEqvMoreAndNotChopSkip*:
⊢ (¬ *f*);*skip* = (*more* ∧ ¬(*f*;*skip*))
proof −
have *1*: ⊢ *wprev f* = (*empty* ∨ *prev f*) **using** *WprevEqvEmptyOrPrev* **by** *auto*
hence *2*: ⊢ (¬(*wprev f*)) = (¬(*empty* ∨ *prev f*)) **by** *auto*
have *3*: ⊢ ¬(*wprev f*) = ((¬ *f*);*skip*) **by** (*simp add*: *wprev-d-def prev-d-def*)
have *31*: ⊢ (*empty* ∨ *prev f*) = (*empty* ∨ (*f*;*skip*)) **by** (*simp add*: *prev-d-def*)
have *32*: ⊢ (*empty* ∨ (*f*;*skip*)) = (¬*more* ∨ ¬¬(*f*;*skip*)) **by** (*simp add*: *empty-d-def*)
have *33*: ⊢ (¬*more* ∨ ¬¬(*f*;*skip*)) = (¬(*more* ∧ ¬(*f*;*skip*))) **by** *fastforce*
have *34*: ⊢ (*empty* ∨ *prev f*) = (¬(*more* ∧ ¬(*f*;*skip*))) **using** *31 32 33* **by** (*metis int-eq*)
have *4*: ⊢ ¬(*empty* ∨ *prev f*) = (*more* ∧ ¬(*f*;*skip*)) **using** *34* **by** *fastforce*

from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltChopImpNotHaltChopNot*:
$\vdash$ *halt* (*init* *w*); *f* $\wedge$ *finite* $\longrightarrow$ $\neg$ (*halt* (*init* *w*); ($\neg$ *f*))
proof –
have *1*: $\vdash$ *halt* (*init* *w*); *f* = *if*$_i$ (*init* *w*) *then* *f* *else* ($\bigcirc$(*halt* (*init* *w*); *f*))
by (*rule HaltChopEqv*)
have *2*: $\vdash$ *if*$_i$ (*init* *w*) *then* *f* *else* ($\bigcirc$(*halt* (*init* *w*); *f*)) $\longrightarrow$
(((*init* *w*) $\longrightarrow$ *f*) $\wedge$ ($\neg$(*init* *w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init* *w*); *f*))))
by (*rule IfThenElseImp*)
have *3*: $\vdash$ *halt* (*init* *w*); ($\neg$*f*) =
if$_i$ (*init* *w*) *then* ($\neg$*f*) *else* ($\bigcirc$(*halt* (*init* *w*); ($\neg$*f*)))
by (*rule HaltChopEqv*)
have *4*: $\vdash$ *if*$_i$ (*init* *w*) *then* ($\neg$*f*) *else* ($\bigcirc$(*halt* (*init* *w*); ($\neg$*f*))) $\longrightarrow$
(((*init* *w*) $\longrightarrow$ $\neg$*f*) $\wedge$ ($\neg$(*init* *w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init* *w*); ($\neg$*f*)))))
by (*rule IfThenElseImp*)
have *5*: $\vdash$ *halt* (*init* *w*); *f* $\wedge$ *halt* (*init* *w*); ($\neg$*f*) $\longrightarrow$
(((*init* *w*) $\longrightarrow$ *f*) $\wedge$ ($\neg$(*init* *w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init* *w*); *f*)))) $\wedge$
(((*init* *w*) $\longrightarrow$ $\neg$*f*) $\wedge$ ($\neg$(*init* *w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init* *w*); ($\neg$*f*)))))
using *1 2 3 4* **by** *fastforce*
have *6*: $\vdash$ (((*init* *w*) $\longrightarrow$ *f*) $\wedge$ ($\neg$(*init* *w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init* *w*); *f*)))) $\wedge$
(((*init* *w*) $\longrightarrow$ $\neg$*f*) $\wedge$ ($\neg$(*init* *w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init* *w*); ($\neg$*f*))))) $\longrightarrow$
($\bigcirc$(*halt* (*init* *w*); *f*)) $\wedge$ ($\bigcirc$(*halt* (*init* *w*); ($\neg$*f*)))
by *auto*
have *7*: $\vdash$ *halt* (*init* *w*); *f* $\wedge$ *halt* (*init* *w*); ($\neg$*f*) $\longrightarrow$
($\bigcirc$(*halt* (*init* *w*); *f*)) $\wedge$ ($\bigcirc$(*halt* (*init* *w*); ($\neg$*f*)))
using *5 6 lift-imp-trans* **by** *blast*
have *8*: $\vdash$ (($\bigcirc$(*halt* (*init* *w*); *f*)) $\wedge$ ($\bigcirc$(*halt* (*init* *w*); ($\neg$*f*)))) =
$\bigcirc$ (*halt* (*init* *w*); *f* $\wedge$ *halt* (*init* *w*); ($\neg$*f*))
using *NextAndEqvNextAndNext* **by** *fastforce*
have *9*: $\vdash$ *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*: $\vdash$ *finite* $\longrightarrow$ $\neg$(*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* $\wedge$ *finite* $\longrightarrow$ (*halt* (*init* *w*)) *yields* *f*
proof –
have *1*: $\vdash$ *halt* (*init* *w*); *f* $\wedge$ *finite* $\longrightarrow$ $\neg$ (*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* $\wedge$ *finite* $\longrightarrow$ (*halt* (*init* *w*)); (*f* $\wedge$ *g*)
proof –

have *1*: ⊢ (*halt* (*init w*)); *g* ∧ *finite* ⟶ (*halt* (*init w*)) *yields* *g*
by (*rule HaltChopImpHaltYields*)
hence *2*: ⊢ (*halt* (*init w*)); *f* ∧ (*halt* (*init w*)); *g* ∧ *finite* ⟶
(*halt* (*init w*)); *f* ∧ (*halt* (*init w*)) *yields* *g* **by** *auto*
have *3*: ⊢ (*halt* (*init w*)); *f* ∧ (*halt* (*init w*)) *yields* *g*⟶
(*halt* (*init w*)); (*f* ∧ *g*) **by** (*rule ChopAndYieldsImp*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltAndChopAndHaltChopImpHaltAndChopAnd*:
⊢ (*halt* (*init w*) ∧ *f*);*f1* ∧ (*halt* (*init w*); *g*) ∧ *finite* ⟶ (*halt* (*init w*) ∧ *f*); (*f1* ∧ *g*)
proof −
have *1*: ⊢ *f1* ⟶ ¬ *g* ∨ (*f1* ∧ *g*)
by *auto*
hence *2*: ⊢ (*halt* (*init w*) ∧ *f*); *f1* ⟶
(*halt* (*init w*) ∧ *f*); (¬ *g*) ∨ ((*halt* (*init w*) ∧ *f*); (*f1* ∧ *g*))
by (*rule ChopOrImpRule*)
have *3*: ⊢ (*halt* (*init w*) ∧ *f*); (¬ *g*) ⟶ *halt* (*init w*); (¬ *g*)
by (*rule AndChopA*)
have *31*: ⊢ (*halt* (*init w*) ∧ *f*); *f1* ⟶
halt (*init w*); (¬ *g*) ∨ ((*halt* (*init w*) ∧ *f*); (*f1* ∧ *g*))
using *23* **by** *fastforce*
have *4*: ⊢ *halt* (*init w*); *g* ∧ *finite* ⟶ ¬ (*halt* (*init w*); (¬ *g*))
by (*rule HaltChopImpNotHaltChopNot*)
hence *41*: ⊢ (*halt* (*init w*); (¬ *g*)) ∧ *finite* ⟶ ¬(*halt* (*init w*); *g*)
by *auto*
have *42*: ⊢ (*halt* (*init w*) ∧ *f*); *f1* ∧ *finite* ⟶
¬(*halt* (*init w*); *g*) ∨ ((*halt* (*init w*) ∧ *f*); (*f1* ∧ *g*))
using *31 41* **by** *fastforce*
from *42* **show** *?thesis* **by** *auto*
qed

lemma *HaltImpBoxYields*:
⊢ (*halt* (*init w*)); *f* ∧ *finite* ⟶ (□(¬ (*init w*))) *yields* ((*halt* (*init w*)); *f*)
proof −
have *1*: ⊢ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) ⟶ *di* (□ (¬ (*init w*)))
by (*rule ChopImpDi*)
have *2*: ⊢ □ (¬ (*init w*)) ⟶ ¬ (*init w*)
by (*rule BoxElim*)
hence *3*: ⊢ *di* (□ (¬ (*init w*))) ⟶ *di* (¬ (*init w*))
by (*rule DiImpDi*)
have *4*: ⊢ *di* (*init* (¬ *w*)) = (*init* (¬*w*))
by (*rule DiState*)
have *41*: ⊢ (*init* (¬ *w*)) = (¬ (*init w*))
using *Initprop*(*2*) **by** *fastforce*
have *42*: ⊢ *di* (¬ (*init w*)) = (¬(*init w*))
using *4 41* **by** (*metis inteq-reflection*)
have *5*: ⊢ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) ⟶ ¬ (*init w*)
using *1 2 42* **using** *3* **by** *fastforce*
hence *51*: ⊢ (*halt* (*init w*); *f*) ∧ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) ⟶

(*halt* (*init w*); *f*) ∧ ¬ (*init w*)
by *fastforce*
have *6*: ⊢ *halt* (*init w*); *f* = *if*$_i$ (*init w*) *then* *f* *else* (○(*halt* (*init w*); *f*))
by (*rule HaltChopEqv*)
hence *61*: ⊢ (*halt* (*init w*); *f* ∧ ¬ (*init w*)) =
((*if*$_i$ (*init w*) *then* *f* *else* (○(*halt* (*init w*); *f*))) ∧ ¬ (*init w*))
using *6* **by** *auto*
have *62*: ⊢ (*if*$_i$ (*init w*) *then* *f* *else* (○(*halt* (*init w*); *f*))) ∧
¬ (*init w*) ⟶ (○(*halt* (*init w*); *f*))
by (*auto simp*: *ifthenelse-d-def*)
have *63*: ⊢ *halt* (*init w*); *f* ∧ ¬ (*init w*) ⟶ (○(*halt* (*init w*); *f*))
using *61 62* **by** *fastforce*
have *7*: ⊢ (*halt* (*init w*); *f*) ∧ (□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) ⟶
○((*halt* (*init w*)); *f*)
using *51 63* **using** *lift-imp-trans* **by** *blast*
have *8*: ⊢ □ (¬ (*init w*)) ⟶ *empty* ∨ ○(□(¬(*init w*)))
using *BoxBoxImpBox BoxEqvAndEmptyOrNextBox* **by** *fastforce*
hence *9*: ⊢ ((□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*))) ⟶
¬ (*halt* (*init w*); *f*) ∨ ○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
by (*rule EmptyOrNextChopImpRule*)
hence *10*: ⊢ ((*halt* (*init w*)); *f*) ∧ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) ⟶
○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
by *fastforce*
have *11*: ⊢ (*halt* (*init w*)); *f* ∧ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) ⟶
○((*halt* (*init w*)); *f*) ∧ ○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
using *7 10* **by** *fastforce*
have *12*: ⊢ ○((*halt* (*init w*)); *f*) ∧ ○((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
⟶ ○(((*halt* (*init w*)); *f*) ∧ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))))
using *NextAndEqvNextAndNext* **by** *fastforce*
have *13*: ⊢ (*halt* (*init w*)); *f* ∧ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)) ⟶
○(((*halt* (*init w*)); *f*) ∧ ((□(¬ (*init w*))); (¬ (*halt* (*init w*); *f*))))
using *11 12* **by** *fastforce*
hence *14*: ⊢ *finite* ⟶ ¬ ((*halt* (*init w*)); *f* ∧ (□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
using *NextLoop* **by** *blast*
hence *15*: ⊢ (*halt* (*init w*)); *f* ∧ *finite* ⟶ ¬ ((□ (¬ (*init w*))); (¬ (*halt* (*init w*); *f*)))
by *auto*
from *15* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

13.12 Properties of Groups of chops

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

qed

end

14 Infinite ITL theorems using strong chop

theory *InfiniteSChopTheorems*
imports
InfiniteTheorems
begin

We give the proofs of a list of Infinite ITL theorems but now using the strong chop.

14.1 Strong Chop axioms

lemma *SChopAssoc*:
$\vdash f \frown (g \frown h) = (f \frown g) \frown h$
proof $-$
have *1*: $\vdash f \frown (g \frown h) = (f \wedge \mathit{finite});((g \wedge \mathit{finite});h)$
by (*simp add*: *schop-d-def*)
have *2*: $\vdash (f \wedge \mathit{finite});((g \wedge \mathit{finite});h) = ((f \wedge \mathit{finite});(g \wedge \mathit{finite}));h$
using *ChopAssoc* **by** *blast*
have *3*: $\vdash ((f \wedge \mathit{finite});(g \wedge \mathit{finite}));h = (f \frown (g \wedge \mathit{finite}));h$
by (*simp add*: *schop-d-def*)
have *4*: $\vdash f \frown (g \wedge \mathit{finite}) = (f \frown g \wedge \mathit{finite})$
by (*simp add*: *schop-d-def*)
(*metis AndChopA ChopAndA ChopAndFiniteDist Prop11 Prop12 inteq-reflection*)
have *5*: $\vdash (f \frown (g \wedge \mathit{finite}));h = (f \frown g \wedge \mathit{finite});h$
using *4* **by** (*simp add*: *LeftChopEqvChop*)
have *6*: $\vdash (f \frown g \wedge \mathit{finite});h = (f \frown g) \frown h$
by (*simp add*: *schop-d-def*)
from *1 2 3 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *OrSChopImp* :
$\vdash (f \vee g) \frown h \longrightarrow f \frown h \vee g \frown h$
by (*auto simp add*: *schop-defs Valid-def sum.case-eq-if*)

lemma *SChopOrImp* :
$\vdash f \frown (g \vee h) \longrightarrow f \frown g \vee f \frown h$
by (*auto simp add*: *schop-defs Valid-def sum.case-eq-if*)

lemma *EmptySChop* :
$\vdash \mathit{empty} \frown f = f$
by (*metis EmptyChopSem FiniteAndEmptyEqvEmpty intI inteq-reflection lift-and-com schop-d-def*)

lemma *SChopEmpty* :

$\vdash$ *finite* $\longrightarrow$ *f* $\frown$*empty* $=$ *f*
by (*auto simp add*: *schop-defs finite-defs empty-defs Valid-def sum.case-eq-if*)

lemma *StateImpBf* :
$\vdash$ *init f* $\longrightarrow$ *bf* (*init f*)
by (*simp add*: *Valid-def bf-defs init-defs sum.case-eq-if*)
(*metis conc-def conc-iprefix-isuffix interval-intlen-gr-zero iprefix-0*)

lemma *BfBoxSChopImpSChop* :
$\vdash$ *bf* (*f* $\longrightarrow$ *f1*) $\wedge$ $\square$(*g* $\longrightarrow$ *g1*) $\longrightarrow$ *f*$\frown$*g* $\longrightarrow$ *f1*$\frown$*g1*
by (*auto simp add*: *Valid-def schop-defs bf-defs always-defs sum.case-eq-if*)

lemma *SChopstarEqv* :
$\vdash$ (*schopstar f*) $=$ (*empty* $\vee$ (*f* $\wedge$ *more*)$\frown$ (*schopstar f*))
using *SChopstarEqvSem Valid-def* **by** *blast*

lemma *AndMoreSChopEqvAndFmoreChop*:
$\vdash$ (*f* $\wedge$ *more*)$\frown$*g* $=$ (*f* $\wedge$ *fmore*);*g*
by (*simp add*: *LeftChopEqvChop AndMoreAndFiniteEqvAndFmore schop-d-def*)

lemma *SOmegaUnroll*:
$\vdash$ f^{ω} $=$ (*f* $\wedge$ *more*)$\frown$$f^{\omega}$
using *OmegaUnroll AndMoreSChopEqvAndFmoreChop* **by** *fastforce*

lemma *SOmegaInduct*:
$\vdash$ (*inf* $\wedge$ *g* $\wedge$ $\square$(*g* $\longrightarrow$ (*f* $\wedge$ *more*)$\frown$*g*)) $\longrightarrow$ *omega f*
using *OmegaInductSem AndMoreSChopEqvAndFmoreChop Valid-def*
by (*metis inteq-reflection*)

lemma *FiniteBfGen*:
assumes $\vdash$ *finite* $\longrightarrow$ *f*
shows $\vdash$ *bf f*
using *assms*
by (*simp add*: *Valid-def bf-defs finite-defs sum.case-eq-if*)

lemma *BfGen*:
assumes $\vdash$ *f*
shows $\vdash$ *bf f*
using *assms*
by (*metis EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteBfGen Prop09 int-eq-true inteq-reflection*)

14.2 ITL operators in terms of SChop

lemma *NextSChopdef* :
$\vdash$ $\bigcirc$ *f* $=$ *skip*$\frown$*f*
by (*metis FiniteChopSkipEqvSkipChopFinite NowImpDiamond Prop10 SkipChopFiniteImpFinite inteq-reflection lift-imp-trans next-d-def schop-d-def sometimes-d-def*)

lemma *DiamondSChopdef* :

$\vdash \Diamond f = \#True \frown f$
by (*simp add*: *schop-d-def sometimes-d-def*)

lemma *FiniteSChopdef*:
$\vdash$ *finite* $= \Diamond$ *empty*
by (*simp add*: *DiamondEmptyEqvFinite int-iffD1 int-iffD2 int-iffI*)

lemma *ChopSChopdef*:
$\vdash f;g = ((f \frown g) \vee (f \wedge inf))$
by (*metis AndInfChopEqvAndInf OrChopEqv OrFiniteInf inteq-reflection schop-d-def*)

lemma *PowerSpowerdef*:
$\vdash$ *power f n* $=$ *spower f n*
proof
(*induct n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case*
by (*metis PowerCommute inteq-reflection pow-Suc schop-d-def spow-Suc*)
qed

lemma *SChopstarFPowerstardef*:
$\vdash$ *schopstar f* $=$ *fpowerstar f*
proof $-$
have *1*: $\vdash$ *schopstar f* $= (\exists\, k.$ *spower* $(f \wedge more)\ k)$
by (*simp add*: *schopstar-d-def spowerstar-d-def*)
have *2*: $\vdash$ *fpowerstar f* $=$ *fpowerstar* $(f \wedge more)$
using *FPSAndMoreEqvFPS* **by** *fastforce*
have *3*: $\vdash$ *fpowerstar* $(f \wedge more) = (\exists\, k.$ *power* $(f \wedge more)\ k)$
by (*simp add*: *fpowerstar-d-def*)
have *4*: $\bigwedge n. \vdash$ *spower* $(f \wedge more)\ n =$ *power* $(f \wedge more)\ n$
using *PowerSpowerdef* **by** *fastforce*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinprop* :
$\vdash ((\#True \frown (f \wedge empty)) \wedge (\#True \frown (g \wedge empty))) = (\#True \frown ((f \wedge g) \wedge empty))$
$\vdash ((\#True \frown (f \wedge empty)) \vee (\#True \frown (g \wedge empty))) = (\#True \frown ((f \vee g) \wedge empty))$
$\vdash$ *finite* $\longrightarrow (\neg\, (\#True \frown (f \wedge empty))) = (\#True \frown (\neg f \wedge empty))$
$\vdash (\neg\, (\#True \frown (f \wedge empty))) = ((\#True \frown (\neg f \wedge empty)) \vee inf)$
using *le-neq-implies-less*
by (*auto simp add*: *Valid-def finite-defs infinite-defs schop-defs empty-defs sum.case-eq-if*)

14.3 Basic Theorems

lemma *BfSChopImpSChop* :
$\vdash bf\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$
proof $-$

have *1*: $\vdash$ *g* $\longrightarrow$ *g* **by** *auto*
hence *2*: $\vdash$ $\Box$ (*g* $\longrightarrow$ *g*) **by** (*rule BoxGen*)
have *3*: $\vdash$ *bf* (*f* $\longrightarrow$ *f1*) $\wedge$ $\Box$(*g* $\longrightarrow$ *g*) $\longrightarrow$ *f*$\frown$*g* $\longrightarrow$ *f1*$\frown$*g* **by** (*rule BfBoxSChopImpSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BiImpBf*:
$\vdash$ *bi f* $\longrightarrow$ *bf f*
by (*simp add*: *bi-defs bf-defs Valid-def sum.case-eq-if*)

lemma *BiSChopImpSChop* :
$\vdash$ *bi* (*f* $\longrightarrow$ *f1*) $\longrightarrow$ *f*$\frown$*g* $\longrightarrow$ *f1*$\frown$*g*
proof –
have *1*: $\vdash$ *g* $\longrightarrow$ *g* **by** *auto*
hence *2*: $\vdash$ $\Box$ (*g* $\longrightarrow$ *g*) **by** (*rule BoxGen*)
have *3*: $\vdash$ *bi* (*f* $\longrightarrow$ *f1*) $\wedge$ $\Box$(*g* $\longrightarrow$ *g*) $\longrightarrow$ *f*$\frown$*g* $\longrightarrow$ *f1*$\frown$*g*
using *BiImpBf BfBoxSChopImpSChop* **using** *BfSChopImpSChop* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *AndSChopA*:
$\vdash$ (*f* $\wedge$ *f1*)$\frown$*g* $\longrightarrow$ *f*$\frown$*g*
proof –
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f* **by** *auto*
hence *2*: $\vdash$ *bf* (*f* $\wedge$ *f1* $\longrightarrow$ *f*) **by** (*rule BfGen*)
have *3*: $\vdash$ *bf* (*f* $\wedge$ *f1* $\longrightarrow$ *f*) $\longrightarrow$ (*f* $\wedge$ *f1*)$\frown$*g* $\longrightarrow$ *f*$\frown$*g* **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *AndSChopB*:
$\vdash$ (*f* $\wedge$ *f1*)$\frown$*g* $\longrightarrow$ *f1*$\frown$*g*
proof –
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f1* **by** *auto*
hence *2*: $\vdash$ *bf* (*f* $\wedge$ *f1* $\longrightarrow$ *f1*) **by** (*rule BfGen*)
have *3*: $\vdash$ *bf* (*f* $\wedge$ *f1* $\longrightarrow$ *f1*) $\longrightarrow$ (*f* $\wedge$ *f1*)$\frown$*g* $\longrightarrow$ *f1*$\frown$*g* **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *NextSChop*:
$\vdash$ ($\bigcirc$ *f*)$\frown$*g* = $\bigcirc$(*f*$\frown$*g*)
proof –
have *1*: $\vdash$ *skip*$\frown$(*f*$\frown$*g*) = (*skip*$\frown$*f*)$\frown$*g* **by** (*rule SChopAssoc*)
from *1* **show** *?thesis* **using** *NextSChopdef* **by** (*metis inteq-reflection*)
qed

lemma *BoxSChopImpSChop* :

$\vdash\ \square\,(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$
proof −
have *1*: $\vdash g \longrightarrow g$ **by** *auto*
hence *2*: $\vdash$ *bf* $(\ g \longrightarrow g)$ **by** (*rule BfGen*)
have *3*: $\vdash$ *bf* $(\ f \longrightarrow f) \wedge \square(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BfBoxSChopImpSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftSChopImpSChop*:
assumes $\vdash f \longrightarrow f1$
shows $\vdash f \frown g \longrightarrow f1 \frown g$
proof −
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ *bf* $(f \longrightarrow f1)$ **by** (*rule BfGen*)
have *3*: $\vdash$ *bf* $(f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightSChopImpSChop*:
assumes $\vdash g \longrightarrow g1$
shows $\vdash f \frown g \longrightarrow f \frown g1$
proof −
have *1*: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence *2*: $\vdash \square\,(g \longrightarrow g1)$ **by** (*rule BoxGen*)
have *3*: $\vdash \square\,(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BoxSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightSChopEqvSChop*:
assumes $\vdash g = g1$
shows $\vdash (f \frown g) = (f \frown g1)$
proof −
have *1*: $\vdash g = g1$ **using** *assms* **by** *auto*
have *2*: $(\vdash g \longrightarrow g1) \Longrightarrow (\vdash f \frown g \longrightarrow f \frown g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: $(\vdash g1 \longrightarrow g) \Longrightarrow (\vdash f \frown g1 \longrightarrow f \frown g)$ **by** (*rule RightSChopImpSChop*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxRightSChopEqvSChop*:
$\vdash \square\,(g = g1) \longrightarrow (f \frown g) = (f \frown g1)$
proof −
have *1*: $\vdash \square\,(g = g1) = (\square\,(g \longrightarrow g1) \wedge \square\,(\ g1 \longrightarrow g))$
by (*auto simp add*: *Valid-def always-defs sum.case-eq-if*)
have *2*: $\vdash \square\,(g \longrightarrow g1) \longrightarrow (f \frown g) \longrightarrow (f \frown g1)$ **by** (*simp add*: *BoxSChopImpSChop*)
have *3*: $\vdash \square\,(g1 \longrightarrow g) \longrightarrow (f \frown g1) \longrightarrow (f \frown g)$ **by** (*simp add*: *BoxSChopImpSChop*)
from *1 2 3* **show** *?thesis* **by** *fastforce*

qed

lemma *FiniteRightSChopEqvSChop*:
assumes $\vdash$ *finite* $\longrightarrow g = g1$
shows $\vdash$ *finite* $\longrightarrow (f\frown g) = (f\frown g1)$
using *assms*
by (*simp add*: *Valid-def finite-defs schop-defs sum.case-eq-if*)

lemma *SChopOrEqv*:
$\vdash f\frown(g \lor g1) = (f\frown g \lor f\frown g1)$
proof –
have *1*: $\vdash g \longrightarrow g \lor g1$ **by** *auto*
hence *2*: $\vdash f\frown g \longrightarrow f\frown(g \lor g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash g1 \longrightarrow g \lor g1$ **by** *auto*
hence *4*: $\vdash f\frown g1 \longrightarrow f\frown(g \lor g1)$ **by** (*rule RightSChopImpSChop*)
from *2 4* **show** *?thesis* **by** (*meson SChopOrImp Prop02 Prop11*)
qed

lemma *OrSChopEqv*:
$\vdash (f \lor f1)\frown g = (f\frown g \lor f1\frown g)$
proof –
have *1*: $\vdash f \longrightarrow f \lor f1$ **by** *auto*
hence *2*: $\vdash f\frown g \longrightarrow (f\lor f1)\frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash f1 \longrightarrow f \lor f1$ **by** *auto*
hence *4*: $\vdash f1\frown g \longrightarrow (f\lor f1)\frown g$ **by** (*rule LeftSChopImpSChop*)
from *2 4* **show** *?thesis*
by (*meson OrSChopImp int-iffI Prop02*)
qed

lemma *OrSChopImpRule*:
assumes $\vdash f \longrightarrow f1 \lor f2$
shows $\vdash f\frown g \longrightarrow (f1\frown g) \lor (f2\frown g)$
proof –
have *1*: $\vdash f \longrightarrow f1 \lor f2$ **using** *assms* **by** *auto*
hence *2*: $\vdash f\frown g \longrightarrow (f1 \lor f2)\frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash (f1 \lor f2)\frown g = (f1\frown g \lor f2\frown g)$ **by** (*rule OrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftSChopEqvSChop*:
assumes $\vdash f = f1$
shows $\vdash f\frown g = (f1\frown g)$
proof –
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow f1$ **by** *auto*
hence *3*: $\vdash f\frown g \longrightarrow f1\frown g$ **by** (*rule LeftSChopImpSChop*)
have $\vdash f1 \longrightarrow f$ **using** *1* **by** *auto*
hence *4*: $\vdash f1\frown g \longrightarrow f\frown g$ **by** (*rule LeftSChopImpSChop*)

from *3 4* **show** *?thesis* **by** (*simp add*: *int-iffI*)
qed

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

lemma *SChopOrImpRule*:
assumes $\vdash g \longrightarrow g1 \vee g2$
shows $\vdash f \frown g \longrightarrow (f \frown g1) \vee (f \frown g2)$
proof −
have *1*: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown (g1 \vee g2)$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash f \frown (g1 \vee g2) = (f \frown g1 \vee f \frown g2)$ **by** (*rule SChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopImpDiamond*:
$\vdash f \frown g \longrightarrow \Diamond\, g$
proof −
have *1*: $\vdash f \longrightarrow \#True$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow \#True \frown g$ **by** (*rule LeftSChopImpSChop*)
from *2* **show** *?thesis* **using** *DiamondSChopdef* **by** *fastforce*
qed

lemma *BfImpDfImpDf*:
$\vdash bf\ (f \longrightarrow g) \longrightarrow\ df\ f \longrightarrow\ df\ g$
proof −
have *1*: $\vdash bf\ (f \longrightarrow g) \longrightarrow (f \frown \#True) \longrightarrow (g \frown \#True)$ **by** (*rule BfSChopImpSChop*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfImpDf*:
assumes $\vdash f \longrightarrow g$
shows $\vdash\ df\ f \longrightarrow\ df\ g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown \#True \longrightarrow g \frown \#True$ **by** (*rule LeftSChopImpSChop*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)

qed

lemma *BfImpBfRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash bf\, f \longrightarrow bf\, g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg\ g \longrightarrow \neg\ f$ **by** *auto*
hence *3*: $\vdash\ df\ (\neg\ g) \longrightarrow\ df\ (\neg\ f)$ **by** (*rule DfImpDf*)
hence *4*: $\vdash \neg\ (\ df\ (\neg\ f)) \longrightarrow \neg\ (\ df\ (\neg\ g))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bf-d-def*)
qed

lemma *DfEqvDf*:
assumes $\vdash f = g$
shows $\vdash\ df\, f =\ df\, g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown \#True = g \frown \#True$ **by** (*rule LeftSChopEqvSChop*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *BfEqvBf*:
assumes $\vdash f = g$
shows $\vdash bf\, f =\ bf\ g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ f) = (\neg\ g)$ **by** *auto*
hence *3*: $\vdash\ df\ (\neg\ f) =\ df\ (\neg\ g)$ **by** (*rule DfEqvDf*)
hence *4*: $\vdash (\neg\ (df\ (\neg\ f))) = (\neg\ (\ df\ (\neg\ g)))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bf-d-def*)
qed

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

lemma *AndSChopCommute* :
$\vdash (f \land f1) \frown g = (f1 \land f) \frown g$

proof −
have *1*: ⊢ $(f \wedge f1) = (f1 \wedge f)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftSChopEqvSChop*)
qed

lemma *BfAndSChopImport*:
⊢ $bf\ f \wedge (f1 \frown g) \longrightarrow (f \wedge f1) \frown g$
proof −
have *1*: ⊢ $f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence *2*: ⊢ $bf\ f \longrightarrow bf\ (f1 \longrightarrow f \wedge f1)$ **by** (*rule BfImpBfRule*)
have *3*: ⊢ $bf\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1 \frown g \longrightarrow (f \wedge f1) \frown g$ **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

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

lemma *StateAndSChopImport*:
⊢ $(init\ w) \wedge (f \frown g) \longrightarrow ((init\ w) \wedge f) \frown g$
proof −
have *1*: ⊢ $(init\ w) \longrightarrow bf\ (init\ w)$ **by** (*rule StateImpBf*)
hence *2*: ⊢ $(init\ w) \wedge (f \frown g) \longrightarrow bf\ (init\ w) \wedge (f \frown g)$ **by** *auto*
have *3*: ⊢ $bf\ (init\ w) \wedge (f \frown g) \longrightarrow ((init\ w) \wedge f) \frown g$ **by** (*rule BfAndSChopImport*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

14.4 Further Properties Df and Bf

lemma *AndFiniteImpDf*:
⊢ $f \wedge finite \longrightarrow df\ f$
proof −
have *1*: ⊢ $finite \longrightarrow f \frown empty = f$ **by** (*rule SChopEmpty*)
have *2*: ⊢ $empty \longrightarrow \#True$ **by** *auto*
hence *3*: ⊢ $f \frown empty \longrightarrow f \frown \#True$ **by** (*rule RightSChopImpSChop*)
have *4* : ⊢ $f \wedge finite \longrightarrow f \frown \#True$ **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfState*:
⊢ $df\ (init\ w) = (init\ w)$

proof −
 have *0*: ⊢ (*init* (¬*w*)) ⟶ *bf* (*init* (¬*w*)) **using** *StateImpBf* **by** *fastforce*
 hence *1*: ⊢ ¬(*init w*) ⟶ *bf* (¬ (*init w*)) **using** *Initprop*(*2*) **by** (*metis inteq-reflection*)
 hence *2*: ⊢ (¬ (*init w*)) ⟶ ¬ (*df* (¬ ¬ (*init w*))) **by** (*simp add*: *bf-d-def*)
 have *3*: ⊢ (¬ (*init w*) ⟶¬ (*df* (¬¬(*init w*)))) ⟶ (*df* (¬¬(*init w*)) ⟶ (*init w*)) **by** *auto*
 have *4*: ⊢ *df* (¬ ¬ (*init w*)) ⟶ (*init w*) **using** *2 3 MP* **by** *blast*
 have *5*: ⊢ (*init w*) ⟶ ¬ ¬ (*init w*) **by** *auto*
 hence *6*: ⊢ *df* (*init w*) ⟶ *df* (¬ ¬ (*init w*)) **by** (*rule DfImpDf*)
 have *7*: ⊢ *df* (*init w*) ⟶ (*init w*) **using** *6 4* **using** *lift-imp-trans* **by** *metis*
 have *8*: ⊢ (*init w*) ∧ *finite* ⟶ *df* (*init w*) **by** (*rule AndFiniteImpDf*)
 from *7 8* **show** *?thesis*
 by (*metis NowImpDiamond Prop10 StateAndChop df-d-def int-simps*(*17*) *inteq-reflection*
 lift-and-com schop-d-def sometimes-d-def)
qed

lemma *StateSChop*:
 ⊢ (*init w*)⌢ *f* ⟶ (*init w*)
by (*simp add*: *StateChopExportA schop-d-def*)

lemma *StateSChopExportA*:
 ⊢ ((*init w*) ∧ *f*)⌢ *g* ⟶ (*init w*)
by (*meson AndSChopA StateSChop lift-imp-trans*)

lemma *StateAndSChop*:
 ⊢ ((*init w*) ∧ *f*)⌢ *g* = ((*init w*) ∧ (*f*⌢ *g*))
by (*simp add*: *AndSChopB StateAndSChopImport StateSChopExportA Prop11 Prop12*)

lemma *StateAndSChopImpSChopRule*:
 assumes ⊢ (*init w*) ∧ *f* ⟶ *f1*
 shows ⊢ (*init w*) ∧ (*f*⌢ *g*) ⟶ (*f1*⌢ *g*)
proof −
 have *1*: ⊢ (*init w*) ∧ *f* ⟶ *f1* **using** *assms* **by** *auto*
 hence *2*: ⊢ ((*init w*) ∧ *f*)⌢ *g* ⟶ *f1*⌢ *g* **by** (*rule LeftSChopImpSChop*)
 have *3*: ⊢ ((*init w*) ∧ *f*)⌢ *g* = ((*init w*) ∧ (*f*⌢ *g*)) **by** (*rule StateAndSChop*)
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpSChopEqvSChop* :
 assumes ⊢ (*init w*) ⟶ (*f* = *f1*)
 shows ⊢ (*init w*) ⟶ ((*f*⌢ *g*) = (*f1*⌢ *g*))
proof −
 have *1*: ⊢ (*init w*) ⟶ (*f*= *f1*) **using** *assms* **by** *auto*
 hence *2*: ⊢ (*init w*) ∧ *f* ⟶ *f1* **by** *auto*
 hence *3*: ⊢ (*init w*) ∧ (*f*⌢ *g*) ⟶ (*f1*⌢ *g*) **by** (*rule StateAndSChopImpSChopRule*)
 have *4*: ⊢ (*init w*) ∧ *f1* ⟶ *f* **using** *1* **by** *auto*
 hence *5*: ⊢ (*init w*) ∧ (*f1*⌢ *g*) ⟶ (*f*⌢ *g*) **by** (*rule StateAndSChopImpSChopRule*)
 from *3 5* **show** *?thesis* **by** *fastforce*

qed

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

lemma *DfIntro*:
$\vdash f \wedge finite \longrightarrow df\ f$
proof −
  have *1*: $\vdash finite \longrightarrow f \frown empty = f$ **by** (*rule SChopEmpty*)
  have *2*: $\vdash empty \longrightarrow \#True$ **by** *auto*
  hence *3*: $\vdash \Box(empty \longrightarrow \#True)$ **by** (*rule BoxGen*)
  have *4*: $\vdash \Box(empty \longrightarrow \#True) \longrightarrow (f;\ empty \longrightarrow f; \#True)$ **by** (*rule BoxChopImpChop*)
  have *5*: $\vdash f \frown empty \longrightarrow f \frown \#True$ **using** *3 4 MP* **by** (*simp add*: *RightSChopImpSChop*)
  hence *6*: $\vdash f \frown empty \longrightarrow df\ f$ **by** (*simp add*: *df-d-def*)
  from *1 6* **show** *?thesis* **using** *AndFiniteImpDf* **by** *blast*
qed

lemma *BfElim*:
$\vdash bf\ f \wedge finite \longrightarrow f$
proof −
  have *1*: $\vdash \neg f \wedge finite \longrightarrow df\ (\neg f)$ **by** (*rule DfIntro*)
  have *2*: $\vdash (\neg f \wedge finite \longrightarrow df\ (\neg f)) \longrightarrow (\neg(df\ (\neg f)) \longrightarrow \neg(\neg f \wedge finite))$
  by *simp*
  have *21*: $\vdash \neg(\neg f \wedge finite) = (f \vee inf)$ **by** (*simp add*: *Valid-def finite-d-def*)
  have *3*: $\vdash \neg (df\ (\neg f)) \longrightarrow f \vee inf$ **using** *1 2 21* **by** *fastforce*
  from *3* **show** *?thesis* **by** (*simp add*: *Prop13 bf-d-def finite-d-def*)
qed

lemma *BfContraPosImpDist*:
 $\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (bf\ f) \longrightarrow (bf\ g)$
proof −
  have *1*: $\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (df\ (\neg g)) \longrightarrow (df\ (\neg f))$ **by** (*rule BfImpDfImpDf*)
  hence *2*: $\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (\neg(df\ (\neg f))) \longrightarrow (\neg(df\ (\neg g)))$ **by** *auto*
  from *2* **show** *?thesis* **by** (*metis bf-d-def*)
qed

lemma *BfImpDist*:
$\vdash bf\ (f \longrightarrow g) \longrightarrow (bf\ f) \longrightarrow (bf\ g)$
proof −
 have *1*: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$ **by** *auto*
 hence *2*: $\vdash \neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g)$ **by** *auto*

hence *3*: $\vdash$ *bf* $(\neg (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg (f \longrightarrow g))$ **by** (*rule BfGen*)
have *4*: $\vdash$ *bf* $(\neg (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg (f \longrightarrow g))$
$\longrightarrow$
bf $(f \longrightarrow g) \longrightarrow$ *bf* $(\neg\ g \longrightarrow \neg\ f)$ **by** (*rule BfContraPosImpDist*)
have *5*: $\vdash$ *bf* $(f \longrightarrow g) \longrightarrow$ *bf* $(\neg\ g \longrightarrow \neg\ f)$ **using** *3 4 MP* **by** *blast*
have *6*: $\vdash$ *bf* $(\neg\ g \longrightarrow \neg\ f) \longrightarrow$ (*bf* $f) \longrightarrow$ (*bf* g) **by** (*rule BfContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FiniteImpBfImpBfRule*:
assumes $\vdash$ *finite* $\longrightarrow (f \longrightarrow g)$
shows $\vdash$ *bf f* $\longrightarrow$ *bf g*
proof –
have *1*: $\vdash$ *finite* $\longrightarrow f \longrightarrow g$ **using** *assms* **by** *auto*
have *2*: $\vdash$ *bf*$(f \longrightarrow g)$ **using** *1* **by** (*simp add*: *FiniteBfGen*)
have *3*: $\vdash$ *bf*$(f \longrightarrow g) \longrightarrow$ *bf f* $\longrightarrow$ *bf g* **using** *BfImpDist* **by** *blast*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteImpBfEqvRule*:
assumes $\vdash$ *finite* $\longrightarrow (f = g)$
shows $\vdash$ *bf f* $=$ *bf g*
proof –
have *1*: $\vdash$ *finite* $\longrightarrow (f = g)$ **using** *assms* **by** *blast*
have *2*: $\vdash$ *finite* $\longrightarrow (f \longrightarrow g)$ **using** *1* **by** *auto*
have *3*: $\vdash$ *bf f* $\longrightarrow$ *bf g* **by** (*simp add*: *2 FiniteImpBfImpBfRule*)
have *4*: $\vdash$ *finite* $\longrightarrow (g \longrightarrow f)$ **using** *1* **by** *auto*
have *5*: $\vdash$ *bf g* $\longrightarrow$ *bf f* **by** (*simp add*: *4 FiniteImpBfImpBfRule*)
from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *IfSChopEqvRule*:
assumes $\vdash f =$ *if*$_i$ (*init w*) *then f1 else f2*
shows $\vdash f \frown g =$ *if*$_i$ (*init w*) *then* $(f1 \frown g)$ *else* $(f2 \frown g)$
proof –
have *1*: $\vdash f =$ *if*$_i$ (*init w*) *then f1 else f2*
using *assms* **by** *auto*
hence *2*: $\vdash f = (((init\ w) \wedge f1) \vee ((init\ (\neg\ w)) \wedge f2))$
by (*simp add*: *ifthenelse-d-def init-defs Valid-def sum.case-eq-if*)
hence *3*: $\vdash f \frown g = (((init\ w) \wedge f1) \frown g \vee ((init\ (\neg\ w)) \wedge f2) \frown g)$
by (*rule OrSChopEqvRule*)
have *4*: $\vdash ((init\ w) \wedge f1) \frown g = ((init\ w) \wedge (f1 \frown g))$
by (*rule StateAndSChop*)
have *5*: $\vdash ((init\ (\neg\ w)) \wedge f2) \frown g = ((init\ (\neg\ w)) \wedge (f2 \frown g))$
by (*rule StateAndSChop*)
have *6*: $\vdash f \frown g = (((init\ w) \wedge f1 \frown g) \vee ((init\ (\neg\ w)) \wedge f2 \frown g))$
using *3 4 5* **by** *fastforce*
from *6* **show** *?thesis* **by** (*simp add*: *ifthenelse-d-def init-defs Valid-def sum.case-eq-if*)

qed

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

lemma *EmptyOrSChopEqv*:
$\vdash (\ empty \vee f) \frown g = (g \vee (f \frown g))$
proof −
have *1*: $\vdash (\ empty \vee f) \frown g = ((\ empty \frown g) \vee (f \frown g))$ **by** (*rule OrSChopEqv*)
have *2*: $\vdash empty \frown g = g$ **by** (*rule EmptySChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqv*:
$\vdash (\ empty \vee \bigcirc f) \frown g = (g \vee \bigcirc(f \frown g))$
proof −
have *1*: $\vdash (\ empty \vee \bigcirc f) \frown g = (g \vee ((\bigcirc f) \frown g))$ **by** (*rule EmptyOrSChopEqv*)
have *2*: $\vdash (\bigcirc f) \frown g = \bigcirc(f \frown g)$ **by** (*rule NextSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrSChopImpRule*:
assumes $\vdash f \longrightarrow empty \vee f1$
shows $\vdash f \frown g \longrightarrow g \vee (f1 \frown g)$
proof −
have *1*: $\vdash f \longrightarrow empty \vee f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow (\ empty \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash (\ empty \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (*rule EmptyOrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

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

lemma *EmptyOrNextSChopImpRule*:
assumes $\vdash f \longrightarrow empty \vee \bigcirc f1$

shows $\vdash f \frown g \longrightarrow g \vee \bigcirc(f1 \frown g)$
proof −
have *1*: $\vdash f \longrightarrow empty \vee \bigcirc f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow (empty \vee \bigcirc f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash (empty \vee \bigcirc f1) \frown g = (g \vee \bigcirc(f1 \frown g))$ **by** (*rule EmptyOrNextSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqvRule*:
assumes $\vdash f = (empty \vee \bigcirc f1)$
shows $\vdash f \frown g = (g \vee \bigcirc(f1 \frown g))$
proof −
have *1*: $\vdash f = (empty \vee \bigcirc f1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g = ((empty \vee \bigcirc f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash (empty \vee \bigcirc f1) \frown g = (g \vee \bigcirc(f1 \frown g))$ **by** (*rule EmptyOrNextSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopEmptyOrImpRule*:
assumes $\vdash g \longrightarrow empty \vee g1$
shows $\vdash f \frown g \wedge finite \longrightarrow f \vee (f \frown g1)$
proof −
have *1*: $\vdash g \longrightarrow empty \vee g1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow (f \frown empty) \vee (f \frown g1)$ **by** (*rule SChopOrImpRule*)
have *3*: $\vdash finite \longrightarrow f \frown empty = f$ **by** (*rule SChopEmpty*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateSChopBoxAndInfImpBox*:
$\vdash \Box (init\ w) \frown \Box (init\ w) \wedge inf \longrightarrow \Box (init\ w)$
by (*simp add*: *Valid-def always-defs schop-defs init-defs sum.case-eq-if infinite-defs iprefix-length iprefix-0*)
(*metis add.right-neutral iprefix-length iprefix-nth isuffix-def le-cases le-iff-add*)

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

have *6*: $\vdash (\Box\ (init\ w) \frown \Box\ (init\ w)) \wedge \neg(\ \Box\ (init\ w)) \longrightarrow$
$\bigcirc(\Box\ (init\ w) \frown \Box\ (init\ w)) \wedge \neg(\ \bigcirc(\Box\ (init\ w)))$
using *4 5* **by** *fastforce*
hence *7*: $\vdash \Box\ (init\ w) \frown \Box\ (init\ w) \wedge finite \longrightarrow \Box\ (init\ w)$
by (*rule NextContra*)
have *8*: $\vdash \Box\ (init\ w) \frown \Box\ (init\ w) \wedge inf \longrightarrow \Box\ (init\ w)$
by (*rule BoxStateSChopBoxAndInfImpBox*)
have *9*: $\vdash \Box\ (init\ w) \frown \Box\ (init\ w) \wedge (finite \vee inf) \longrightarrow \Box\ (init\ w)$
using *7 8* **by** *fastforce*
hence *10*: $\vdash\ \Box\ (init\ w) \frown \Box\ (init\ w) \longrightarrow \Box\ (init\ w)$
using *FiniteOrInfinite* **by** *fastforce*
have *11*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge \Box\ (init\ w))$
by (*rule BoxEqvAndBox*)
have *12*: $\vdash empty \frown \Box\ (init\ w) = \Box\ (init\ w)$
by (*rule EmptySChop*)
have *13*: $\vdash ((init\ w) \wedge\ empty\ ) \frown \Box\ (init\ w) = ((init\ w) \wedge (\ empty \frown \Box\ (init\ w)))$
by (*rule StateAndSChop*)
have *14*: $\vdash \Box\ (init\ w) = ((init\ w) \wedge\ empty\ ) \frown \Box\ (init\ w)$
using *11 12 13* **by** *fastforce*
have *15*: $\vdash (init\ w) \wedge\ empty\ \longrightarrow \Box\ (init\ w)$
by (*rule StateAndEmptyImpBoxState*)
hence *16*: $\vdash ((init\ w) \wedge\ empty\ ) \frown \Box\ (init\ w) \longrightarrow \Box\ (init\ w) \frown \Box\ (init\ w)$
by (*rule LeftSChopImpSChop*)
have *17*: $\vdash \Box\ (init\ w) \longrightarrow \Box\ (init\ w) \frown \Box\ (init\ w)$
using *14 16* **by** *fastforce*
from *10 17* **show** *?thesis* **by** *fastforce*
qed

lemma *NotBoxStateImpBoxSYieldsNotBox*:
$\vdash\ \neg(\ \Box\ (init\ w)) \longrightarrow (\Box\ (init\ w))\ syields\ (\neg(\ \Box\ (init\ w)))$
proof −
have *1*: $\vdash \Box\ (init\ w) \frown \Box\ (init\ w) = \Box\ (init\ w)$ **by** (*rule BoxStateSChopBoxEqvBox*)
have *2*: $\vdash \Box\ (init\ w) = (\neg\ \neg(\ \Box\ (init\ w)))$ **by** *auto*
hence *3*: $\vdash \Box\ (init\ w) \frown \Box(init\ w) = \Box(init\ w) \frown (\neg\neg(\Box(init\ w)))$ **by** (*rule RightSChopEqvSChop*)
have *4*: $\vdash \neg(\ \Box\ (init\ w)) \longrightarrow \neg\ (\Box\ (init\ w) \frown (\neg\ \neg\ (\Box\ (init\ w))))$ **using** *1 3* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *StateEqvBf*:
$\vdash\ (init\ w) = bf\ (init\ w)$
proof −
have *1*: $\vdash (init\ w) \longrightarrow bf\ (init\ w)$ **by** (*rule StateImpBf*)
have *2*: $\vdash bf\ (init\ w) \wedge finite \longrightarrow (init\ w)$ **by** (*rule BfElim*)
from *1 2* **show** *?thesis*
by (*metis DfState Initprop*(*2*) *Prop11 bf-d-def int-simps*(*4*) *inteq-reflection*)
qed

lemma *TrueSChopEqvDiamond*:

$\vdash \ \ \#True \frown f = \Diamond f$
using *DiamondSChopdef* **by** *fastforce*

lemma *BfAndEqvBfAndBf*:
$\vdash bf(f \wedge g) = (bf\, f \wedge bf\, g)$
proof −
 have *1*: $\vdash f \wedge g \longrightarrow f$ **by** *auto*
 have *2*: $\vdash bf(f \wedge g) \longrightarrow bf\, f$ **by** (*simp add*: *1 BfImpBfRule*)
 have *3*: $\vdash f \wedge g \longrightarrow g$ **by** *auto*
 have *4*: $\vdash bf(f \wedge g) \longrightarrow bf\, g$ **by** (*simp add*: *3 BfImpBfRule*)
 have *5*: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
 have *6*: $\vdash bf\, f \longrightarrow bf\ (g \longrightarrow f \wedge g)$ **by** (*simp add*: *5 BfImpBfRule*)
 have *7*: $\vdash bf\ (g \longrightarrow f \wedge g) \longrightarrow (bf\, g \longrightarrow bf(f \wedge g))$ **by** (*simp add*: *BfImpDist*)
 have *8*: $\vdash bf\, f \wedge bf\, g \longrightarrow bf\ (f \wedge g)$ **using** *6 7* **by** *fastforce*
 from *2 4 8* **show** *?thesis* **by** *fastforce*
qed

lemma *BfEqvBfImpAndBfImp*:
$\vdash bf(f = g) = (bf\ (f \longrightarrow g) \wedge bf(g \longrightarrow f))$
proof −
 have *1*: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** *auto*
 have *2*: $\vdash bf(f = g) = bf((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (*simp add*: *1 BfEqvBf*)
 have *3*: $\vdash bf((f \longrightarrow g) \wedge (g \longrightarrow f)) = (bf(f \longrightarrow g) \wedge bf(g \longrightarrow f))$ **by** (*simp add*: *BfAndEqvBfAndBf*)
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BfEqvImpSChopEqvSChop*:
$\vdash bf(f = f1) \longrightarrow f \frown g = f1 \frown g$
proof −
 have *1*: $\vdash bf(f = f1) = (bf\ (f \longrightarrow f1) \wedge bf(f1 \longrightarrow f))$ **by** (*simp add*: *BfEqvBfImpAndBfImp*)
 have *2*: $\vdash bf\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*simp add*: *BfSChopImpSChop*)
 have *3*: $\vdash bf(f1 \longrightarrow f) \longrightarrow f1 \frown g \longrightarrow f \frown g$ **by** (*simp add*: *BfSChopImpSChop*)
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BfEqvDfEqvDf*:
$\vdash bf(f = g) \longrightarrow (df\, f = df\, g)$
proof −
 have *1*: $\vdash bf(f = g) \longrightarrow (f \frown \#True) = (g \frown \#True)$
  using *BfEqvImpSChopEqvSChop* **by** *fastforce*
 from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *FiniteImpEqvDfImpRule*:

assumes $\vdash$ *finite* $\longrightarrow$ $f = g$
shows $\vdash$ *df f* $=$ *df g*
proof $-$
have *1*: $\vdash$ *finite* $\longrightarrow$ $f = g$ **using** *assms* **by** *auto*
have *2*: $\vdash$ $bf(f = g)$ **using** *1* **by** (*simp add*: *FiniteBfGen*)
have *3*: $\vdash$ $bf(f = g) \longrightarrow (df\,f = df\,g)$ **by** (*simp add*: *BfEqvDfEqvDf*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfEmpty*:
$\vdash$ *df empty*
proof $-$
have *1*: $\vdash$ *#True* **by** *auto*
have *2*: $\vdash$ *empty* $\frown$ *#True* $=$ *#True* **by** (*rule EmptySChop*)
have *3*: $\vdash$ *empty* $\frown$ *#True* **using** *1 2* **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *BfImpDf*:
$\vdash$ *bf f* $\longrightarrow$ *df f*
proof $-$
have *1*: $\vdash$ $f \longrightarrow (empty \longrightarrow f)$ **by** *auto*
have *2*: $\vdash$ *bf f* $\longrightarrow$ $bf(empty \longrightarrow f)$ **by** (*simp add*: *1 BfImpBfRule*)
have *3*: $\vdash$ $bf(empty \longrightarrow f) \longrightarrow$ *df empty* $\longrightarrow$ *df f* **by** (*simp add*: *BfImpDfImpDf*)
have *4*: $\vdash$ *bf f* $\longrightarrow$ *df empty* $\longrightarrow$ *df f* **using** *2 3 lift-imp-trans* **by** *blast*
have *5*: $\vdash$ *df empty* **by** (*simp add*: *DfEmpty*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

14.5 Properties of SDa and SBa

lemma *SDaEqvDtDf*:
$\vdash$ *sda f* $= \Diamond$ (*df f*)
proof $-$
have *1*: $\vdash$ *#True*$\frown$ (*f*$\frown$ *#True*) $=$ *#True*$\frown$ (*f*$\frown$ *#True*) **by** *auto*
hence *2*: $\vdash$ *#True*$\frown$ (*f*$\frown$ *#True*) $=$ *#True*$\frown$ *df f* **by** (*simp add*: *df-d-def*)
have *3*: $\vdash$ *#True*$\frown$ (*df f*) $= \Diamond$(*df f*) **by** (*simp add*: *TrueSChopEqvDiamond*)
have *4*: $\vdash$ *#True*$\frown$ (*f*$\frown$ *#True*) $= \Diamond$(*df f*) **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*:*sda-d-def*)
qed

lemma *SDaEqvDfDt*:
$\vdash$ *sda f* $=$ *df* ($\Diamond$ *f*)
proof $-$
have *1*: $\vdash$ *#True*$\frown$ *f* $= \Diamond$ *f* **by** (*rule TrueSChopEqvDiamond*)
hence *2*: $\vdash$ (*#True*$\frown$ *f*)$\frown$ *#True* $=$ ($\Diamond$ *f*)$\frown$ *#True* **by** (*rule LeftSChopEqvSChop*)
hence *3*: $\vdash$ (*#True*$\frown$ *f*)$\frown$ *#True* $=$ *df*($\Diamond$ *f*) **by** (*simp add*: *df-d-def*)

have *4*: $\vdash \#True\frown (f\frown \#True) = (\#True\frown f)\frown \#True$ **by** (*rule SChopAssoc*)
have *5*: $\vdash \#True\frown (f\frown \#True) = df\ (\Diamond\ f)$ **using** *3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *sda-d-def*)
qed

lemma *DtDfEqvDfDt*:
$\vdash\ \Diamond\ (df\ f) = df\ (\Diamond\ f)$
by (*meson Prop04 SDaEqvDfDt SDaEqvDtDf*)

lemma *SBaEqvBfBt*:
$\vdash\ sba\ f = bf(\Box\ f)$
proof −
have *1*: $\vdash sda\ (\neg\ f) = df(\Diamond\ (\neg\ f))$ **by** (*rule SDaEqvDfDt*)
have *2*: $\vdash \Diamond\ (\neg\ f) = (\neg(\Box\ f))$ **by** (*rule DiamondNotEqvNotBox*)
hence *3*: $\vdash df\ (\Diamond(\neg\ f)) = df\ (\neg\ (\Box\ f))$ **by** (*rule DfEqvDf*)
have *4*: $\vdash sda\ (\neg\ f) = df\ (\neg(\Box\ f))$ **using** *1 3* **by** *fastforce*
hence *5*: $\vdash (\neg\ (sda\ (\neg\ f))) = (\neg\ (df\ (\neg(\Box\ f))))$ **by** *auto*
hence *6*: $\vdash (\neg\ (sda\ (\neg\ f))) = bf(\Box\ f)$ **by** (*simp add*: *bf-d-def*)
from *6* **show** *?thesis* **by** (*simp add*: *sba-d-def*)
qed

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

lemma *DfDfNotEqvNotBfBf*:
$\vdash df\ (df\ (\neg\ f)) = (\neg\ (bf\ (bf\ f)))$
proof −
have *1*: $\vdash df\ (\neg\ f) = (\neg\ bf\ f)$ **by** (*simp add*: *DfNotEqvNotBf*)
have *2*: $\vdash df\ (df\ (\neg\ f)) = df\ (\neg\ bf\ f)$ **by** (*simp add*: *1 DfEqvDf*)
have *3*: $\vdash df\ (\neg\ bf\ f) = (\neg\ bf\ (bf\ f))$ **by** (*simp add*: *DfNotEqvNotBf*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfDtEqvDtDf*:
$\vdash df(\Diamond\ f) = \Diamond(df\ f)$
proof −
have *1*: $\vdash (\#True\frown f)\frown\#True = \#True\frown(f\frown\#True)$
using *SChopAssoc* **by** *fastforce*
have *2*: $\vdash (\Diamond\ f)\frown\#True = \Diamond(f\frown\#True)$
using *1* **by** (*metis TrueSChopEqvDiamond int-eq*)
from *1 2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfDtNotEqvNotBfBt*:
$\vdash df(\diamond(\neg\ f)) = (\neg(bf(\Box\ f)))$
proof −
 have *1*: $\vdash \diamond\ (\neg f) = (\neg\ (\Box\ f))$ **by** (*simp add*: *DiamondNotEqvNotBox*)
 have *2*: $\vdash df(\diamond\ (\neg f)) = df(\neg\ (\Box\ f))$ **by** (*simp add*: *1 DfEqvDf*)
 have *3*: $\vdash df(\neg(\Box\ f)) = (\neg(bf\ (\Box\ f)))$ **by** (*simp add*: *DfNotEqvNotBf*)
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DtDfNotEqvNotBtBf*:
$\vdash \diamond(df\ (\neg\ f)) = (\neg\ (\Box(bf\ f)))$
proof −
 have *1*: $\vdash df(\neg\ f) = (\neg\ (bf\ f))$ **using** *DfNotEqvNotBf* **by** *blast*
 have *2*: $\vdash \diamond(df(\neg\ f)) = \diamond(\neg\ (bf\ f))$ **by** (*simp add*: *1 DiamondEqvDiamond*)
 have *3*: $\vdash \diamond(\neg\ (bf\ f)) = (\neg\ \Box(bf\ f))$ **by** (*simp add*: *DiamondNotEqvNotBox*)
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaEqvBtBf*:
$\vdash\ \ sba\ \ f = \Box\ (bf\ \ f)$
proof −
 have *1*: $\vdash\ \ sda\ (\neg\ \ f) = \diamond\ (df\ (\neg\ \ f))$ **by** (*rule SDaEqvDtDf*)
 have *2*: $\vdash\ \ df\ (\neg\ \ f) = (\neg\ (bf\ \ f))$ **by** (*rule DfNotEqvNotBf*)
 hence *3*: $\vdash \diamond\ (df\ (\neg\ \ f)) = \diamond(\neg\ (bf\ \ f))$ **by** (*rule DiamondEqvDiamond*)
 have *4*: $\vdash (\neg\ (\diamond(\neg\ (bf\ \ f)))) = \Box(bf\ \ f)$ **by** (*rule NotDiamondNotEqvBox*)
 have *5*: $\vdash (\neg\ (\ sda\ (\neg\ \ f))) = \Box(bf\ \ f)$ **using** *1 2 3 4* **by** *fastforce*
 from *5* **show** *?thesis* **by** (*simp add*: *sba-d-def*)
qed

lemma *BaImpSBa*:
$\vdash ba\ f \longrightarrow sba\ f$
using *BaEqvBiBt BiImpBf SBaEqvBfBt* **by** *fastforce*

lemma *SDaImpDa*:
$\vdash sda\ f \longrightarrow da\ f$
proof −
 have *1*: $\vdash ba\ (\neg\ f) \longrightarrow sba\ (\neg f)$
  using *BaImpSBa* **by** *blast*
 have *2*: $\vdash \neg\ sba\ (\neg f) \longrightarrow \neg\ ba\ (\neg\ f)$
  using *1* **by** *fastforce*
 from *2* **show** *?thesis* **by** (*simp add*: *sba-d-def ba-d-def*)
qed

lemma *BtBfEqvBfBt*:
$\vdash\ \ \Box\ (bf\ \ f) = bf(\ \Box\ f)$
proof −

have *1*: ⊢ *sba f* = □ (*bf f*) **by** (*rule SBaEqvBtBf*)
have *2*: ⊢ *sba f* = *bf*(□ *f*) **by** (*rule SBaEqvBfBt*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateEqvSBaBoxState*:
⊢ □ (*init w*) = *sba* (□ (*init w*))
proof −
have *1*: ⊢ (*init w*) = *bf* (*init w*) **by** (*rule StateEqvBf*)
hence *2*: ⊢ □ (*init w*) = □ (*bf* (*init w*)) **by** (*rule BoxEqvBox*)
have *3*: ⊢ □ (*bf* (*init w*)) = *bf*(□ (*init w*)) **by** (*rule BtBfEqvBfBt*)
have *4*: ⊢ □ (*init w*) = □(□ (*init w*)) **by** (*rule BoxEqvBoxBox*)
hence *5*: ⊢ *bf*(□ (*init w*)) = *bf* (□(□ (*init w*))) **by** (*rule BfEqvBf*)
have *6*: ⊢ *sba*(□ (*init w*)) = *bf*(□(□ (*init w*))) **by** (*rule SBaEqvBfBt*)
from *2 3 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpBf*:
⊢ *sba f* ⟶ *bf f*
proof −
have *1*: ⊢ *sba f* = □(*bf f*) **by** (*rule SBaEqvBtBf*)
have *2*: ⊢ □(*bf f*) ⟶ *bf f* **by** (*rule BoxElim*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *BaImpBf*:
⊢ *ba f* ⟶ *bf f*
proof −
have *1*: ⊢ *ba f* = □(*bi f*) **by** (*rule BaEqvBtBi*)
have *2*: ⊢ □(*bi f*) ⟶ *bi f* **by** (*rule BoxElim*)
have *3*: ⊢ *bi f* ⟶ *bf f* **by** (*simp add*: *BiImpBf*)
from *1 2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *SBaImpBt*:
⊢ *sba f* ∧ *finite* ⟶ □ *f*
proof −
have *1*: ⊢ *sba f* = *bf*(□ *f*) **by** (*rule SBaEqvBfBt*)
have *2*: ⊢ *bf*(□ *f*) ∧ *finite* ⟶ □ *f* **by** (*rule BfElim*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondImpSDa*:
⊢ ◇ *f* ∧ *finite* ⟶ *sda f*
using *AndFiniteImpDf SDaEqvDfDt* **by** *force*

lemma *DfImpSDa*:
⊢ *df f* ⟶ *sda f*
using *NowImpDiamond SDaEqvDtDf* **by** *fastforce*

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

lemma *SBaAndSChopImport*:
$\vdash \ \ sba\ f \wedge finite \wedge (g \frown g1) \ \longrightarrow (f \wedge g) \frown (f \wedge g1)$
proof –
have *1*: $\vdash sba\ \ f \longrightarrow bf\ \ f$ **by** (*rule SBaImpBf*)
have *2*: $\vdash bf\ \ f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (*rule BfAndSChopImport*)
have *3*: $\vdash sba\ \ f \wedge finite \longrightarrow \Box\ f$ **by** (*rule SBaImpBt*)
have *4*: $\vdash \Box\ f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (*rule BoxAndSChopImport*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndSChopImport*:
$\vdash \ \ ba\ f \wedge (g \frown g1) \ \longrightarrow (f \wedge g) \frown (f \wedge g1)$
proof –
have *1*: $\vdash ba\ \ f \longrightarrow bi\ \ f$ **by** (*rule BaImpBi*)
have *2*: $\vdash bi\ \ f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (*rule BiAndSChopImport*)
have *3*: $\vdash ba\ \ f \longrightarrow \Box\ f$ **by** (*rule BaImpBt*)
have *4*: $\vdash \Box\ f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (*rule BoxAndSChopImport*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndCommute*:
$\vdash \ \ f \frown (g \wedge g1) = f \frown (g1 \wedge g)$
proof –
have *1*: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightSChopEqvSChop*)
qed

lemma *SChopAndA*:
$\vdash \ \ f \frown (g \wedge g1) \longrightarrow f \frown g$
proof –
have *1*: $\vdash (g \wedge g1) \longrightarrow\ \ g$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightSChopImpSChop*)
qed

lemma *SChopAndB*:
$\vdash \ \ f \frown (g \wedge g1) \longrightarrow f \frown g1$
proof –
have *1*: $\vdash (g \wedge g1) \longrightarrow\ \ g1$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightSChopImpSChop*)
qed

lemma *BoxStateAndSChopEqvSChop*:
$\vdash$ $(\Box\ (init\ w) \wedge finite \wedge (f \frown g)) = ((\Box\ (init\ w) \wedge f) \frown (\Box\ (init\ w) \wedge g) \wedge finite)$
proof –
have *1*: $\vdash \Box\ (init\ w) = sba(\ \Box\ (init\ w))$
by (*rule BoxStateEqvSBaBoxState*)
have *2*: $\vdash sba(\ \Box\ (init\ w)) \wedge finite \wedge (f \frown g) \longrightarrow (\Box\ (init\ w) \wedge f) \frown (\Box\ (init\ w) \wedge g)$
by (*rule SBaAndSChopImport*)
have *3*: $\vdash \Box\ (init\ w) \wedge finite \wedge (f \frown g) \longrightarrow (\Box\ (init\ w) \wedge f) \frown (\Box\ (init\ w) \wedge g)$
using *1 2* **by** *fastforce*
have *11*: $\vdash (\Box\ (init\ w) \wedge f) \frown (\Box\ (init\ w) \wedge g) \longrightarrow (\Box\ (init\ w)) \frown (\Box\ (init\ w) \wedge g)$
by (*rule AndSChopA*)
have *12*: $\vdash (\Box\ (init\ w)) \frown (\Box\ (init\ w) \wedge g) \longrightarrow (\Box\ (init\ w)) \frown (\Box\ (init\ w))$
by (*rule SChopAndA*)
have *13*: $\vdash (\Box\ (init\ w)) \frown (\Box\ (init\ w)) = \Box\ (init\ w)$
by (*rule BoxStateSChopBoxEqvBox*)
have *14*: $\vdash (\Box\ (init\ w) \wedge f) \frown (\Box\ (init\ w) \wedge g) \longrightarrow f \frown (\Box\ (init\ w) \wedge g)$
by (*rule AndSChopB*)
have *15*: $\vdash f \frown (\Box\ (init\ w) \wedge g) \longrightarrow f \frown g$
by (*rule SChopAndB*)
have *16*: $\vdash (\Box\ (init\ w) \wedge f) \frown (\Box\ (init\ w) \wedge g) \longrightarrow \Box\ (init\ w) \wedge (f \frown g)$
using *11 12 13 14 15* **by** *fastforce*
from *3 16* **show** *?thesis* **by** *fastforce*
qed

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

lemma *SChopAndBoxImport*:
$\vdash$ $f \frown g \wedge \Box\ h \longrightarrow f \frown (g \wedge h)$
proof –
have *1*: $\vdash \Box\ h \wedge f \frown g \longrightarrow f \frown (h \wedge g)$ **by** (*rule BoxAndSChopImport*)
have *2*: $\vdash f \frown (h \wedge g) = f \frown (g \wedge h)$ **by** (*rule SChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *AndSChopAndCommute*:
$\vdash$ $(f \wedge g) \frown (f1 \wedge g1) = (g \wedge f) \frown (g1 \wedge f1)$
proof –
have *1*: $\vdash (f \wedge g) \frown (f1 \wedge g1) = (g \wedge f) \frown (f1 \wedge g1)$ **by** (*rule AndSChopCommute*)
have *2*: $\vdash (g \wedge f) \frown (f1 \wedge g1) = (g \wedge f) \frown (g1 \wedge f1)$ **by** (*rule SChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopImpSChop*:
assumes $\vdash f \longrightarrow f1$
$\vdash g \longrightarrow g1$
shows $\vdash f \frown g \longrightarrow f1 \frown g1$
proof –
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f1 \frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence *4*: $\vdash f1 \frown g \longrightarrow f1 \frown g1$ **by** (*rule RightSChopImpSChop*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopEqvSChop*:
assumes $\vdash f = f1$
$\vdash g = g1$
shows $\vdash f \frown g = f1 \frown g1$
proof –
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g = f1 \frown g$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash g = g1$ **using** *assms* **by** *auto*
hence *4*: $\vdash f1 \frown g = f1 \frown g1$ **by** (*rule RightSChopEqvSChop*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxSChopImpSChopBox*:
$\vdash \Box\, h \longrightarrow f \frown g \longrightarrow f \frown (\Box\, h \wedge g)$
proof –
have *1*: $\vdash \Box\, h \longrightarrow \Box(g \longrightarrow \Box\, h \wedge g\,)$ **by** (*rule BoxImpBoxImpBox*)
have *2*: $\vdash \Box(g \longrightarrow \Box\, h \wedge g\,) \longrightarrow f \frown g \longrightarrow f \frown (\Box\, h \wedge g)$ **by** (*rule BoxSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotChopEqvSYieldsNot*:
$\vdash (\neg (f \frown g)) = f$ *syields* $(\neg\ g)$
proof –
have *1*: $\vdash g = (\neg \neg\ g)$ **by** *auto*
hence *2*: $\vdash f \frown g = f \frown (\neg \neg\ g)$ **by** (*rule RightSChopEqvSChop*)
hence *3*: $\vdash (\neg (f \frown g)) = (\neg (f \frown (\neg \neg\ g)))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *NotDfFalse*:
$\vdash \neg (\ df\ \#False)$
proof –
have *1*: $\vdash (init\ \#True) \longrightarrow bf\ (init\ \#True)$ **by** (*rule StateImpBf*)
hence *2*: $\vdash \#True \longrightarrow bf\ \#True$ **by** (*auto simp*: *bf-defs sum.case-eq-if*)
have *3*: $\vdash \#True$ **by** *auto*

have *4*: $\vdash$ *bf* #*True* **using** *2 3 MP* **by** *auto*
hence *5*: $\vdash \neg$ (*df* ($\neg$ #*True*)) **by** (*simp add*: *bf-d-def*)
have *6*: $\vdash$ ($\neg$ #*True*) = #*False* **by** *auto*
hence *7*: $\vdash$ *df* ($\neg$ #*True*) = *df* #*False* **by** (*rule DfEqvDf*)
from *5 7* **show** *?thesis* **by** *auto*
qed

lemma *StateAndEmptySChop*:
$\vdash$ ((*init w*) $\wedge$ *empty*)$\frown$ *f* = ((*init w*) $\wedge$ *f*)
proof −
have *1*: $\vdash$ ((*init w*) $\wedge$ *empty*)$\frown$ *f* = ((*init w*) $\wedge$ *empty* $\frown$ *f*) **by** (*rule StateAndSChop*)
have *2*: $\vdash$ *empty* $\frown$ *f* = *f* **by** (*rule EmptySChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndNextSChop*:
$\vdash$ ((*init w*) $\wedge \bigcirc$ *f*)$\frown$ *g* = ((*init w*) $\wedge \bigcirc$(*f*$\frown$ *g*))
proof −
have *1*: $\vdash$ ((*init w*) $\wedge \bigcirc$ *f*)$\frown$ *g* = ((*init w*) $\wedge$ ($\bigcirc$ *f*)$\frown$ *g*) **by** (*rule StateAndSChop*)
have *2*: $\vdash$ ($\bigcirc$ *f*)$\frown$ *g* = $\bigcirc$(*f*$\frown$ *g*) **by** (*rule NextSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NextStateAndSChop*:
$\vdash$ $\bigcirc$(((*init w*) $\wedge$ *f*)$\frown$ *g*) = ($\bigcirc$ (*init w*) $\wedge \bigcirc$(*f*$\frown$ *g*))
proof −
have *1*: $\vdash$ ((*init w*) $\wedge$ *f*)$\frown$ *g* = ((*init w*) $\wedge$ *f*$\frown$ *g*) **by** (*rule StateAndSChop*)
hence *2*: $\vdash$ $\bigcirc$(((*init w*) $\wedge$ *f*)$\frown$ *g*) = $\bigcirc$((*init w*) $\wedge$ *f*$\frown$ *g*) **by** (*rule NextEqvNext*)
have *3*: $\vdash$ $\bigcirc$((*init w*) $\wedge$ *f*$\frown$ *g*) = ($\bigcirc$ (*init w*) $\wedge \bigcirc$(*f*$\frown$ *g*)) **by** (*rule NextAndEqvNextAndNext*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateSYieldsEqv*:
$\vdash$ ((*init w*) $\longrightarrow$ (*f syields g*)) = ((*init w*) $\wedge$ *f*) *syields g*
proof −
have *1*: $\vdash$ ((*init w*) $\wedge$ *f*)$\frown$ ($\neg$ *g*) = ((*init w*) $\wedge$ *f*$\frown$ ($\neg$ *g*)) **by** (*rule StateAndSChop*)
hence *2*: $\vdash$ ((*init w*) $\longrightarrow \neg$ (*f*$\frown$ ($\neg$ *g*))) = ($\neg$ (((*init w*) $\wedge$ *f*)$\frown$ ($\neg$ *g*))) **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *StateAndDf*:
$\vdash$ ((*init w*) $\wedge$ *df f*) = *df* ((*init w*) $\wedge$ *f*)
proof −
have *1*: $\vdash$ ((*init w*) $\wedge$ *f*)$\frown$ #*True*= ((*init w*) $\wedge$ *f*$\frown$ #*True*) **by** (*rule StateAndSChop*)
from *1* **show** *?thesis* **by** (*metis df-d-def inteq-reflection*)
qed

lemma *DfNext*:

$\vdash \quad df(\bigcirc f) = \bigcirc (df \ f)$
proof −
have *1*: $\vdash (\bigcirc f)\frown \#True = \bigcirc(f\frown \#True)$ **by** (*rule NextSChop*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfNextState*:
$\vdash \quad df(\bigcirc (init\ w)) = \bigcirc (init\ w)$
proof −
have *1*: $\vdash \ df(\bigcirc (init\ w)) = \bigcirc(df \ (init\ w))$ **by** (*rule DfNext*)
have *2*: $\vdash \ df \ (init\ w) = (init\ w)$ **by** (*rule DfState*)
hence *3*: $\vdash \bigcirc(df \ (init\ w)) = \bigcirc (init\ w)$ **by** (*rule NextEqvNext*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfStateAndNextStateEqvStateAndNextState*:
$\vdash df(init\ w \wedge \bigcirc(init\ w1)) = (init\ w \wedge \bigcirc(init\ w1))$
proof −
have *1*: $\vdash (init\ w \wedge \bigcirc(init\ w1))\frown\#True = (init\ w \wedge \bigcirc((init\ w1)\frown \#True))$
using *StateAndNextSChop* **by** *blast*
have *2*: $\vdash df(init\ w \wedge \bigcirc(init\ w1)) = (init\ w \wedge \bigcirc((init\ w1)\frown \#True))$
using *1* **by** (*simp add*: *df-d-def*)
have *3*: $\vdash df(init\ w1) = init\ w1$
by (*simp add*: *DfState*)
have *4*: $\vdash skip\frown df(init\ w1) = skip\frown(init\ w1)$
by (*simp add*: *3 RightSChopEqvSChop*)
have *5*: $\vdash \bigcirc(df(init\ w1)) = \bigcirc(init\ w1)$
by (*simp add*: *3 NextEqvNext*)
from *2 5* **show** *?thesis* **by** (*metis df-d-def int-eq*)
qed

lemma *StateImpBfGen*:
assumes $\vdash (init\ w) \longrightarrow f$
shows $\vdash (init\ w) \longrightarrow bf \ f$
proof −
have *1*: $\vdash (init\ w) \longrightarrow f$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg \ f \longrightarrow \neg \ (init\ w)$ **by** *auto*
hence *3*: $\vdash \ df \ (\neg \ f) \longrightarrow \ df \ (\neg \ (init\ w))$ **by** (*rule DfImpDf*)
hence *4*: $\vdash \ df \ (\neg \ f) \longrightarrow \ df \ (init\ (\neg w))$ **by** (*metis Initprop*(*2*) *inteq-reflection*)
have *5*: $\vdash \ df \ (init\ (\neg\ w)) = \ (init\ (\neg\ w))$ **by** (*rule DfState*)
have *6*: $\vdash \ df \ (\neg \ f) \longrightarrow \neg \ (init\ w)$ **using** *4 5* **using** *Initprop*(*2*) **by** *fastforce*
hence *7*: $\vdash (init\ w) \longrightarrow \neg (\ df \ (\neg \ f))$ **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *bf-d-def*)
qed

lemma *SChopAndNotSChopImp*:
$\vdash \quad f\frown g \wedge \neg (f\frown g1) \longrightarrow f\frown (g \wedge \neg \ g1)$
proof −
have *1*: $\vdash g \longrightarrow (g \wedge \neg \ g1) \vee \ g1$ **by** *auto*

hence *2*: $\vdash f \frown g \longrightarrow f \frown ((g \wedge \neg\ g1) \vee\ g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash f \frown ((g \wedge \neg\ g1) \vee\ g1) \longrightarrow (f \frown (g \wedge \neg\ g1)) \vee\ (f \frown g1)$ **by** (*rule SChopOrImp*)
have *4*: $\vdash f \frown g \longrightarrow f \frown (g \wedge \neg\ g1) \vee\ f \frown g1$ **using** *2 3 MP* **by** *fastforce*
from *4* **show** *?thesis* **by** *auto*
qed

lemma *SChopAndSYieldsImp*:
$\vdash\ f \frown g \wedge f$ *syields* $g1 \longrightarrow f \frown (g \wedge g1)$
proof –
have *1*: $\vdash g \longrightarrow (g \wedge g1) \vee\ \neg\ g1$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown ((g \wedge\ g1) \vee\ \neg g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash f \frown ((g \wedge\ g1) \vee\ \neg g1) \longrightarrow (f \frown (g \wedge\ g1)) \vee\ (f \frown (\neg g1))$ **by** (*rule SChopOrImp*)
have *4*: $\vdash f \frown g \longrightarrow f \frown (g \wedge\ g1) \vee\ f \frown (\neg g1)$ **using** *2 3 MP* **by** *fastforce*
hence *5*: $\vdash f \frown g \wedge \neg\ (f \frown (\neg\ g1)) \longrightarrow f \frown (g \wedge g1)$ **by** *auto*
from *5* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SChopAndSYieldsMP*:
$\vdash\ f \frown g \wedge f$ *syields* $(g \longrightarrow g1) \longrightarrow f \frown g1$
proof –
have *1*: $\vdash f \frown g \wedge f$ *syields* $(g \longrightarrow g1) \longrightarrow f \frown (g \wedge (g \longrightarrow g1))$ **by** (*rule SChopAndSYieldsImp*)
have *2*: $\vdash g \wedge (g \longrightarrow g1) \longrightarrow g1$ **by** *auto*
hence *3*: $\vdash f \frown (g \wedge (g \longrightarrow g1)) \longrightarrow f \frown g1$ **by** (*rule RightSChopImpSChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *OrSYieldsImp*:
$\vdash\ (f \vee\ f1)$ *syields* $g = ((f$ *syields* $g) \wedge (f1$ *syields* $g))$
proof –
have *1*: $\vdash ((f \vee\ f1) \frown (\neg\ g)) = ((f \frown (\neg\ g)) \vee\ (f1 \frown (\neg\ g)))$ **by** (*rule OrSChopEqv*)
hence *2*: $\vdash (\neg ((f \vee\ f1) \frown (\neg\ g))) = (\neg (f \frown (\neg\ g)) \wedge\ \neg(f1 \frown (\neg\ g)))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *LeftSYieldsImpSYields*:
assumes $\vdash f \longrightarrow f1$
shows $\vdash (f1$ *syields* $g) \longrightarrow (f$ *syields* $g)$
proof –
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown (\neg\ g) \longrightarrow f1 \frown (\neg\ g)$ **by** (*rule LeftSChopImpSChop*)
hence *3*: $\vdash \neg (f1 \frown (\neg\ g)) \longrightarrow \neg (f \frown (\neg\ g))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *LeftSYieldsEqvSYields*:
assumes $\vdash\ f = f1$
shows $\vdash (f$ *syields* $g) = (f1$ *syields* $g)$
proof –
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown (\neg\ g) = f1 \frown (\neg\ g)$ **by** (*rule LeftSChopEqvSChop*)

hence *3*: $\vdash$ ($\neg$ ($f\frown$ ($\neg$ g))) = ($\neg$ ($f1\frown$ ($\neg$ g))) **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

14.6 Properties of SFin

lemma *SFinEqvTrueSChopAndEmpty*:
$\vdash$ *sfin f* = #*True*$\frown$(*f* $\wedge$ *empty*)
proof −
have *01*: $\vdash$ *sfin f* = ($\neg$ *fin* ($\neg$ *f*))
by (*simp add*: *sfin-d-def*)
have *02*:$\vdash$ ($\neg$ *fin* ($\neg$ *f*)) = ($\neg$ ($\square$ (*empty* $\longrightarrow$ $\neg$ *f*)))
by (*simp add*: *fin-d-def*)
have *03*:$\vdash$ ($\neg$ ($\square$ (*empty* $\longrightarrow$ $\neg$ *f*))) = $\diamond$($\neg$(*empty* $\longrightarrow$ $\neg$ *f*))
by (*simp add*: *always-d-def*)
have *04*: $\vdash$ $\neg$(*empty* $\longrightarrow$ $\neg$ *f*) = (*empty* $\wedge$ *f*)
by *auto*
have *05*:$\vdash$ $\diamond$($\neg$(*empty* $\longrightarrow$ $\neg$ *f*)) = $\diamond$ (*empty* $\wedge$ *f*)
using *04*
using *inteq-reflection* **by** *fastforce*
from *01 02 03 05* **show** *?thesis*
by (*metis SChopAndCommute TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *DiamondSFin*:
$\vdash$ $\diamond$(*sfin w*) = *sfin w*
by (*metis* (*no-types*, *lifting*) *ChopAssoc FiniteChopFiniteEqvFinite FiniteOr FiniteOrInfinite InfEqvNotFinite OrFiniteInf SFinEqvTrueSChopAndEmpty finite-d-def int-eq-true int-simps*(*21*) *inteq-reflection schop-d-def sometimes-d-def*)

lemma *SChopSFinExportA*:
$\vdash$ *f*$\frown$(*g* $\wedge$ *sfin w*) $\longrightarrow$ *sfin w*
using *DiamondSFin*
by (*metis SChopAndB SChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *SFinImpBox*:
$\vdash$ *sfin w* $\longrightarrow$ $\square$(*sfin w*)
by
(*metis* (*mono-tags*, *lifting*) *DiamondFin always-d-def intI int-eq int-simps*(*4*) *sfin-d-def unl-lift2*)

lemma *SFinAndSChopImport*:
$\vdash$ (*sfin w*) $\wedge$ (*f*$\frown$*g*) $\longrightarrow$ *f*$\frown$((*sfin w*) $\wedge$ *g*)
proof −
have *1*: $\vdash$ *sfin w* $\longrightarrow$ $\square$(*sfin w*) **by** (*rule SFinImpBox*)
hence *2*: $\vdash$ *sfin w* $\wedge$ (*f*$\frown$*g*) $\longrightarrow$ $\square$(*sfin w*) $\wedge$ (*f*$\frown$*g*) **by** *auto*
have *3*: $\vdash$ $\square$(*sfin w*) $\wedge$ (*f*$\frown$*g*) $\longrightarrow$ *f*$\frown$((*sfin w*) $\wedge$ *g*) **using** *BoxAndSChopImport* **by** *blast*
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SFinAndSChop*:
$\vdash (f \frown (g \wedge \mathit{sfin}\ w)) = (\mathit{sfin}\ w \wedge f \frown g)$
using *SFinAndSChopImport SChopSFinExportA SChopAndA SChopAndCommute*
by *fastforce*

lemma *SChopAndEmptyEqvEmptySChopEmpty*:
$\vdash ((f \frown g) \wedge \mathit{empty}) = (f \wedge \mathit{empty}) \frown (g \wedge \mathit{empty})$
by (*auto simp*: *empty-defs schop-defs sum.case-eq-if*)

lemma *SFinAndEmpty*:
$\vdash ((\mathit{sfin}\ w) \wedge \mathit{empty}) = (w \wedge \mathit{empty})$
proof −
 have *1*: $\vdash ((\mathit{sfin}\ w) \wedge \mathit{empty}) = (\#\mathit{True} \frown (w \wedge \mathit{empty}) \wedge \mathit{empty})$
  using *SFinEqvTrueSChopAndEmpty* **by** *fastforce*
 have *2*: $\vdash (\#\mathit{True} \frown (w \wedge \mathit{empty}) \wedge \mathit{empty}) = ((\#\mathit{True} \wedge \mathit{empty}) \frown (w \wedge \mathit{empty}))$
  using *SChopAndEmptyEqvEmptySChopEmpty*
   by (*metis* (*no-types*, *lifting*) *Prop11 Prop12 inteq-reflection lift-and-com*)
 have *3*: $\vdash (\#\mathit{True} \wedge \mathit{empty}) \frown (w \wedge \mathit{empty}) = (\mathit{empty} \frown (w \wedge \mathit{empty}))$
  using *LeftSChopEqvSChop* **by** *fastforce*
 have *4*: $\vdash (\mathit{empty} \frown (w \wedge \mathit{empty})) = (w \wedge \mathit{empty})$
  using *EmptySChop* **by** *blast*
 from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *AndSFinEqvSChopAndEmpty*:
$\vdash\ \ ((f \wedge \mathit{finite}) \wedge \mathit{sfin}\ g) = f \frown (g \wedge\ \mathit{empty}\ )$
proof −
 have *1*: $\vdash ((f \wedge \mathit{finite}) \wedge \mathit{sfin}\ g) = (f \frown \mathit{empty} \wedge \mathit{sfin}\ g)$
  using *SChopEmpty*
  by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite FiniteImpAnd Prop10 SChopImpDiamond inteq-reflection lift-and-com*)
 have *2*: $\vdash (\mathit{sfin}\ g \wedge f \frown \mathit{empty}) = (f \frown (\mathit{empty} \wedge \mathit{sfin}\ g))$
  using *SFinAndSChop* **by** *fastforce*
 have *3*: $\vdash (\mathit{empty} \wedge \mathit{sfin}\ g) =\ (\mathit{sfin}\ g \wedge \mathit{empty})$
  by *auto*
 have *4*: $\vdash (\mathit{sfin}\ g \wedge \mathit{empty}) = (g \wedge \mathit{empty})$
  using *SFinAndEmpty* **by** *metis*
 have *5*: $\vdash (\mathit{empty} \wedge \mathit{sfin}\ g) = (g \wedge \mathit{empty})$
  using *3 4* **by** *auto*
 hence *6*: $\vdash f \frown (\mathit{empty} \wedge \mathit{sfin}\ g) = f \frown (g \wedge \mathit{empty})$
  using *RightSChopEqvSChop* **by** *blast*
 from *1 2 5* **show** *?thesis* **by** (*metis inteq-reflection lift-and-com*)
qed

lemma *AndSFinEqvSChopStateAndEmpty*:
$\vdash\ \ ((f \wedge \mathit{finite}) \wedge\ \mathit{sfin}\ \ (\mathit{init}\ w)) = f\ \frown\ ((\mathit{init}\ w) \wedge\ \ \mathit{empty}\ )$
using *AndSFinEqvSChopAndEmpty* **by** *blast*

lemma *DiamondEqvEmptyOrNextDiamond*:

$\vdash \Diamond\ f = (f\ \lor \bigcirc(\Diamond\ f))$
proof $-$
have *1*: $\vdash \Box\ (\neg\ f) = ((\neg\ f) \land wnext(\Box\ (\neg\ f)))$
by (*simp add*: *BoxEqvAndWnextBox*)
have *2*: $\vdash (\neg\ \Diamond\ f) = ((\neg f) \land wnext(\Box\ (\neg\ f)))$
using *1* **by** (*simp add*: *always-d-def*)
have *3*: $\vdash\ \Diamond\ f = (f \lor \neg(wnext(\Box\ (\neg\ f))))$
using *2* **by** *auto*
have *4*: $\vdash (\neg(wnext(\Box\ (\neg\ f)))) = \bigcirc(\neg\Box(\neg f))$
by (*simp add*: *wnext-d-def*)
have *5*: $\vdash \neg\Box(\neg f) = \Diamond\ f$
by (*simp add*: *always-d-def*)
have *6*: $\vdash \bigcirc(\neg\Box(\neg f)) = \bigcirc(\Diamond\ f)$
using *5* **using** *inteq-reflection* **by** *force*
from *3 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinStateEqvStateAndEmptyOrNextSFinState*:
$\vdash\ sfin\ \ (init\ w) = (((init\ w) \land empty)\ \ \lor\ \ \bigcirc(\ sfin\ \ (init\ w)))$
proof $-$
have *01*: $\vdash sfin\ (init\ w) = \#True\frown((init\ w) \land empty)$
by (*simp add*: *SFinEqvTrueSChopAndEmpty*)
have *02*: $\vdash \#True\frown((init\ w) \land empty) = \Diamond\ ((init\ w) \land empty)$
by (*simp add*: *TrueSChopEqvDiamond*)
have *03*: $\vdash \Diamond\ ((init\ w) \land empty) = (((init\ w) \land empty)\ \ \lor\ \ \bigcirc(\ sfin\ \ (init\ w)))$
using *DiamondEqvEmptyOrNextDiamond 02 01* **by** (*metis inteq-reflection*)
from *01 02 03* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSChopEqvOr*:
$\vdash\ \ (\ sfin\ \ (init\ w))\frown\ f = (((init\ w) \land f) \lor\ \bigcirc((\ sfin\ \ (init\ w))\frown\ f))$
proof $-$
have *1*: $\vdash\ sfin\ \ (init\ w) = (((init\ w) \land\ \ empty\ ) \lor\ \bigcirc(\ sfin\ \ (init\ w)))$
by (*rule SFinStateEqvStateAndEmptyOrNextSFinState*)
hence *2*: $\vdash (\ sfin\ \ (init\ w))\frown\ f = (((init\ w) \land\ \ empty\ )\lor\ \bigcirc(\ sfin\ \ (init\ w)))\frown\ f$
by (*rule LeftSChopEqvSChop*)
have *3*: $\vdash (((init\ w) \land\ \ empty\ )\lor\ \bigcirc\ (sfin\ \ (init\ w)))\frown\ f$
$= (((init\ w) \land\ \ empty\ )\frown\ f \lor\ (\bigcirc\ (sfin\ \ (init\ w)))\frown\ f)$
by (*rule OrSChopEqv*)
have *4*: $\vdash ((init\ w) \land\ \ empty\ )\frown\ f = ((init\ w) \land f)$
by (*rule StateAndEmptySChop*)
have *5*: $\vdash (\bigcirc\ (sfin\ \ (init\ w)))\frown\ f = \bigcirc((\ sfin\ \ (init\ w))\frown\ f)$
by (*rule NextSChop*)
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSChopEqvDiamond*:
$\vdash\ \ (\ sfin\ \ (init\ w))\frown\ f = \Diamond\ ((init\ w) \land f)$
proof $-$

have *1*: $\vdash$ (*sfin* (*init w*)) = (#*True*⌢((*init w*) ∧ *empty*))
by (*simp add*: *SFinEqvTrueSChopAndEmpty*)
hence *2*: $\vdash$ (*sfin* (*init w*))⌢*f* = (#*True*⌢((*init w*) ∧ *empty*))⌢*f*
by (*rule LeftSChopEqvSChop*)
have *3*: $\vdash$ #*True*⌢(((*init w*) ∧ *empty*)⌢*f*) = (#*True*⌢((*init w*) ∧ *empty*))⌢*f*
by (*rule SChopAssoc*)
have *4*: $\vdash$ #*True*⌢(((*init w*) ∧ *empty*)⌢*f*)= ◇ (((*init w*) ∧ *empty*)⌢*f*)
using *TrueSChopEqvDiamond* **by** *blast*
have *5*: $\vdash$ ((*init w*) ∧ *empty*)⌢*f* = ((*init w*) ∧ *f*)
using *StateAndEmptySChop* **by** *blast*
hence *6*: $\vdash$ ◇ (((*init w*) ∧ *empty*)⌢*f*) = ◇ ((*init w*) ∧ *f*)
by (*rule DiamondEqvDiamond*)
from *2 3 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSYields*:
$\vdash$ (*sfin* (*init w*)) *syields* (*init w*)
proof −
have *1*: $\vdash$ (*sfin* (*init w*))⌢ (¬(*init w*)) = ◇((*init w*) ∧ ¬(*init w*))
by (*rule SFinSChopEqvDiamond*)
have *2*: $\vdash$ ¬(◇((*init w*) ∧ ¬ (*init w*))) **by** (*rule NotDiamondAndNot*)
have *3*: $\vdash$ ¬ ((*sfin* (*init w*))⌢ (¬ (*init w*))) **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *AndFiniteImpAndSFinStateOrSFinNotState*:
$\vdash$ *f* ∧ *finite* ⟶ (*f* ∧ *sfin* (*init w*)) ∨ (*f* ∧ *sfin* (¬ (*init w*)))
by (*simp add*: *finite-defs sfin-defs Valid-def sum.case-eq-if*)

lemma *AndSFinSChopEqvStateAndSChop*:
$\vdash$ (*f* ∧ *sfin* (*init w*))⌢ *g* = *f*⌢ ((*init w*) ∧ *g*)
proof −
have *1*: $\vdash$ (*sfin* (*init w*)) *syields* (*init w*)
by (*rule SFinSYields*)
have *2*: $\vdash$ *f* ∧ *sfin* (*init w*) ⟶ *sfin* (*init w*)
by *auto*
hence *3*: $\vdash$ (*sfin* (*init w*)) *syields* (*init w*) ⟶
(*f* ∧ *sfin* (*init w*)) *syields* (*init w*)
using *LeftSYieldsImpSYields* **by** *metis*
have *4*: $\vdash$ (*f* ∧ *sfin* (*init w*)) *syields* (*init w*)
using *1 3 MP* **by** *fastforce*
have *5*: $\vdash$ (*f* ∧ *sfin* (*init w*))⌢ *g* ∧ (*f* ∧ *sfin* (*init w*)) *syields* (*init w*)
⟶ (*f* ∧ *sfin* (*init w*))⌢ (*g* ∧ (*init w*))
by (*rule SChopAndSYieldsImp*)
have *6*: $\vdash$ (*f* ∧ *sfin* (*init w*))⌢ *g* ⟶ (*f* ∧ *sfin* (*init w*))⌢ (*g* ∧ (*init w*))
using *4 5* **by** *fastforce*
have *7*: $\vdash$ (*f* ∧ *sfin* (*init w*))⌢ (*g* ∧ (*init w*)) ⟶ *f*⌢ (*g* ∧ (*init w*))
by (*rule AndSChopA*)
have *8*: $\vdash$ *g* ∧ (*init w*) ⟶ (*init w*) ∧ *g*
by *auto*

hence *9*: $\vdash f \frown (g \wedge (init\ w)) \longrightarrow f \frown ((init\ w) \wedge g)$
  by (*rule RightSChopImpSChop*)
have *10*: $\vdash (f \wedge\ sfin\ (init\ w)) \frown g \longrightarrow f \frown ((init\ w) \wedge g)$
  using *6 7 9* **by** *fastforce*
have *11*: $\vdash (f \wedge finite) \longrightarrow (f \wedge\ sfin\ (init\ w)) \vee\ (f \wedge sfin\ (\neg\ (init\ w)))$
  using *AndFiniteImpAndSFinStateOrSFinNotState* **by** *blast*
hence *12*: $\vdash f \frown\ ((init\ w) \wedge g) \longrightarrow$
    $((f \wedge\ sfin\ (init\ w)) \vee\ (f \wedge sfin\ (\neg\ (init\ w))\ )) \frown ((init\ w) \wedge g)$
  by (*metis FiniteImp LeftChopImpChop inteq-reflection schop-d-def*)
have *13*: $\vdash ((f \wedge\ sfin\ (init\ w)) \vee\ (f \wedge sfin\ (\neg\ (init\ w)))) \frown ((init\ w) \wedge g)$
    $=$
    $((f \wedge sfin\ (init\ w)) \frown ((init\ w) \wedge g) \vee\ (f \wedge sfin\ (\neg\ (init\ w))) \frown ((init\ w) \wedge g))$
  by (*rule OrSChopEqv*)
have *14*: $\vdash (\ f \wedge sfin\ \ (init\ (\neg\ w))) \frown ((init\ w) \wedge g) \longrightarrow \diamond(\ (init\ (\neg\ w)) \wedge ((init\ w) \wedge g))$
  using *SFinSChopEqvDiamond*
  by (*metis SChopImpSChop Prop12 int-iffD1 inteq-reflection lift-and-com*)
have *141*: $\vdash \neg(\ \diamond(\ \ (init\ (\neg\ w)) \wedge ((init\ w) \wedge g))) \longrightarrow$
    $\neg\ (\ (\ f \wedge sfin\ \ (init\ (\neg\ w))) \frown ((init\ w) \wedge g))$
  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 sum.case-eq-if*)
have *151*: $\vdash \neg\ (\ (\ f \wedge sfin\ \ (init\ (\neg\ w))) \frown ((init\ w) \wedge g))$
  using *15 141* **by** *fastforce*
have *1511*: $\vdash (\ f \wedge sfin\ (\neg\ \ (init\ w))) \frown ((init\ w) \wedge g) \longrightarrow \#False$
  using *151* **by** (*metis Initprop*(*2*) *int-simps*(*14*) *inteq-reflection*)
have *152*: $\vdash (f \wedge sfin(init\ w)) \frown ((init\ w) \wedge g) \vee\ (f \wedge sfin\ (\neg(init\ w))) \frown ((init\ w) \wedge g) \longrightarrow$
    $(f \wedge\ sfin\ (init\ w)) \frown ((init\ w) \wedge g)$
  using *1511* **by** *fastforce*
have *16*: $\vdash f \frown ((init\ w) \wedge g) \longrightarrow (f \wedge\ sfin\ (init\ w)) \frown ((init\ w) \wedge g)$
  using *12 13 152* **by** *fastforce*
have *17*: $\vdash (f \wedge\ sfin\ (init\ w)) \frown ((init\ w) \wedge g) \longrightarrow (f \wedge\ sfin\ (init\ w)) \frown g$
  by (*rule SChopAndB*)
have *18*: $\vdash f \frown ((init\ w) \wedge g) \longrightarrow (f \wedge\ sfin\ (init\ w)) \frown g$
  using *16 17* **by** *fastforce*
from *10 18* **show** *?thesis* **by** *fastforce*
qed

lemma *DfAndSFinEqvSChopState*:
$\vdash\ \ df\ (f \wedge\ sfin\ (init\ w)) = f \frown (init\ w)$
proof −
have *1*: $\vdash (f \wedge sfin(init\ w)) \frown \#True = f \frown ((init\ w) \wedge \#True)$
  by (*rule AndSFinSChopEqvStateAndSChop*)
have *2*: $\vdash ((init\ w) \wedge \#True) = (init\ w)$ **by** *auto*
hence *3*: $\vdash (f \frown ((init\ w) \wedge \#True)) = (f \frown (init\ w))$ **by** (*rule RightSChopEqvSChop*)
have *4*: $\vdash (f \wedge\ sfin\ (init\ w)) \frown \#True = f \frown (init\ w)$ **using** *1 3* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *SFinNotStateEqvNotSFinState*:

$\vdash$ *finite* $\longrightarrow$ ($\neg$(*sfin* (*init w*))) = (*sfin* (*init* ($\neg$ *w*)))
using *SFinEqvTrueSChopAndEmpty*
by (*metis InitAndEmptyEqvAndEmpty SFinprop*(*3*) *inteq-reflection*)

lemma *BfImpSFinEqvSYieldsState*:
$\vdash$ *bf* (*f* $\longrightarrow$ *sfin* (*init w*)) = *f syields* (*init w*)
proof –
have *1*: $\vdash$ *df* (*f* $\wedge$ *sfin* (*init* ($\neg$ *w*))) = *f* $\frown$ (*init* ($\neg$ *w*))
by (*rule DfAndSFinEqvSChopState*)
have *2*: $\vdash$ *finite* $\longrightarrow$ (*f* $\wedge$ *sfin*(*init* ($\neg$ *w*))) = (*f* $\wedge$ $\neg$(*sfin*(*init w*)))
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
have *3*: $\vdash$ (*f* $\wedge$ $\neg$ (*sfin*(*init w*))) = ($\neg$ (*f* $\longrightarrow$ *sfin* (*init w*)))
by *auto*
have *4*: $\vdash$ *finite* $\longrightarrow$ (*f* $\wedge$ *sfin*(*init* ($\neg$ *w*))) = ($\neg$ (*f* $\longrightarrow$ *sfin*(*init w*)))
using *2 3* **by** *fastforce*
hence *5*: $\vdash$ *df* (*f* $\wedge$ *sfin* (*init* ($\neg$ *w*))) = *df* ($\neg$ (*f* $\longrightarrow$ *sfin*(*init w*)))
by (*metis DfEqvNotBfNot FiniteImpAnd df-d-def inteq-reflection schop-d-def*)
have *6*: $\vdash$ *df* ($\neg$ (*f* $\longrightarrow$ *sfin* (*init w*))) = ($\neg$(*bf* (*f* $\longrightarrow$ *sfin*(*init w*))))
by (*rule DfNotEqvNotBf*)
have *7*: $\vdash$ $\neg$ (*bf* (*f* $\longrightarrow$ *sfin* (*init w*))) = *f* $\frown$(*init* ($\neg$ *w*))
using *1 5 6 Initprop* **by** *fastforce*
hence *8*: $\vdash$ *bf* (*f* $\longrightarrow$ *sfin* (*init w*)) = ($\neg$ (*f* $\frown$ ($\neg$ (*init w*))))
by (*metis Initprop*(*2*) *int-eq int-simps*(*7*))
from *8* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *StateImpSYields*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *sfin* (*init w1*)
shows $\vdash$ (*init w*) $\longrightarrow$ (*f syields* (*init w1*))
proof –
have *1*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *sfin* (*init w1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*init w*) $\longrightarrow$ (*f* $\longrightarrow$ *sfin* (*init w1*)) **by** *auto*
hence *3*: $\vdash$ (*init w*) $\longrightarrow$ *bf* (*f* $\longrightarrow$ *sfin* (*init w1*))
using *StateImpBfGen* **by** *auto*
have *4*: $\vdash$ *bf* (*f* $\longrightarrow$ *sfin* (*init w1*)) = *f syields* (*init w1*)
by (*rule BfImpSFinEqvSYieldsState*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndSYieldsImpSYields*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1*
shows $\vdash$ (*init w*) $\wedge$ (*f1 syields g*) $\longrightarrow$ (*f syields g*)
proof –
have *1*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*init w*) $\wedge$ (*f* $\frown$ ($\neg$ *g*)) $\longrightarrow$ *f1* $\frown$ ($\neg$ *g*) **by** (*rule StateAndSChopImpSChopRule*)
hence *3*: $\vdash$ (*init w*) $\wedge$ $\neg$ (*f1* $\frown$ ($\neg$ *g*)) $\longrightarrow$ $\neg$ (*f* $\frown$ ($\neg$ *g*)) **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *AndSYieldsA*:
$\vdash$ *f syields g* $\longrightarrow$ (*f* $\wedge$ *f1*) *syields g*
proof –
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftSYieldsImpSYields*)
qed

lemma *AndSYieldsB*:
$\vdash$ *f1 syields g* $\longrightarrow$ (*f* $\wedge$ *f1*) *syields g*
proof –
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f1* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftSYieldsImpSYields*)
qed

lemma *RightSYieldsImpSYields*:
assumes $\vdash$ *g* $\longrightarrow$ *g1*
shows $\vdash$ (*f syields g*) $\longrightarrow$ (*f syields g1*)
proof –
have *1*: $\vdash$ *g* $\longrightarrow$ *g1* **using** *assms* **by** *auto*
hence *2*: $\vdash \neg$ *g1* $\longrightarrow \neg$ *g* **by** *auto*
hence *3*: $\vdash$ *f* $\frown$ ($\neg$ *g1*) $\longrightarrow$ *f* $\frown$ ($\neg$ *g*) **by** (*rule RightSChopImpSChop*)
hence *4*: $\vdash \neg$ (*f* $\frown$ ($\neg$ *g*)) $\longrightarrow \neg$ (*f* $\frown$ ($\neg$ *g1*)) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *RightSYieldsEqvSYields*:
assumes $\vdash$ *g* = *g1*
shows $\vdash$ (*f syields g*) = (*f syields g1*)
proof –
have *1*: $\vdash$ *g* = *g1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ ($\neg$ *g*) = ($\neg$ *g1*) **by** *auto*
hence *3*: $\vdash$ *f* $\frown$ ($\neg$ *g*) = *f* $\frown$ ($\neg$ *g1*) **by** (*rule RightSChopEqvSChop*)
hence *4*: $\vdash$ ($\neg$ (*f* $\frown$ ($\neg$ *g*))) = ($\neg$ (*f* $\frown$ ($\neg$ *g1*))) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *BoxImpSYields*:
$\vdash$ $\Box$ *g* $\longrightarrow$ *f syields g*
proof –
have *1*: $\vdash$ *f* $\frown$ ($\neg$ *g*) $\longrightarrow$ $\Diamond$($\neg$ *g*) **by** (*rule SChopImpDiamond*)
hence *2*: $\vdash \neg$ ($\Diamond$($\neg$ *g*)) $\longrightarrow \neg$ (*f* $\frown$ ($\neg$ *g*)) **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def always-d-def*)
qed

lemma *BoxEqvTrueSYields*:
$\vdash$ $\Box$ *f* = #*True syields f*
proof –
have *1*: $\vdash$ #*True* $\frown$ ($\neg$ *f*) = $\Diamond$ ($\neg$ *f*) **by** (*rule TrueSChopEqvDiamond*)
hence *2*: $\vdash$ ($\neg$ (#*True* $\frown$ ($\neg$ *f*))) = ($\neg$($\Diamond$ ($\neg$ *f*))) **by** *auto*
have *3*: $\vdash$ $\Box$ *f* = ($\neg$ ($\Diamond$ ($\neg$ *f*))) **by** (*simp add*: *always-d-def*)

have *4*: $\vdash \Box f = (\neg (\#True \frown (\neg f)))$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SYieldsGen*:
assumes $\vdash g$
shows $\vdash$ *f syields g*
proof −
have *1*: $\vdash g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box g$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box g \longrightarrow$ *f syields g* **by** (*rule BoxImpSYields*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SYieldsAndSYieldsEqvSYieldsAnd*:
$\vdash$ ((*f syields g*) $\wedge$ (*f syields g1*)) = *f syields* ($g \wedge g1$)
proof −
have *1*: $\vdash f \frown (\neg g \vee \neg g1) = ((f \frown (\neg g)) \vee (f \frown (\neg g1)))$ **by** (*rule SChopOrEqv*)
hence *2*: $\vdash ((f \frown (\neg g)) \vee (f \frown (\neg g1))) = f \frown (\neg g \vee \neg g1)$ **by** *auto*
have *3*: $\vdash (\neg g \vee \neg g1) = (\neg (g \wedge g1))$ **by** *auto*
hence *4*: $\vdash f \frown (\neg g \vee \neg g1) = f \frown (\neg (g \wedge g1))$ **by** (*rule RightSChopEqvSChop*)
have *5*: $\vdash (f \frown (\neg g)) \vee (f \frown (\neg g1)) = f \frown (\neg (g \wedge g1))$ **using** *2 4* **by** *fastforce*
hence *6*: $\vdash (\neg (f \frown (\neg g)) \wedge \neg (f \frown (\neg g1))) = (\neg (f \frown (\neg (g \wedge g1))))$
by (*auto simp*: *schop-defs sum.case-eq-if*)
from *6* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SYieldsAndSYieldsImpAndSYieldsAnd*:
$\vdash$ (*f syields g*) $\wedge$ (*f1 syields g1*) $\longrightarrow$ ($f \wedge f1$) *syields* ($g \wedge g1$)
proof −
have *1*: $\vdash$ *f syields g* $\longrightarrow$ ($f \wedge f1$) *syields g*
by (*rule AndSYieldsA*)
have *2*: $\vdash$ *f1 syields g1* $\longrightarrow$ ($f \wedge f1$) *syields g1*
by (*rule AndSYieldsB*)
have *3*: $\vdash$ (($f \wedge f1$) *syields g* $\wedge$ ($f \wedge f1$) *syields g1*) = ($f \wedge f1$) *syields* ($g \wedge g1$)
by (*rule SYieldsAndSYieldsEqvSYieldsAnd*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SYieldsSYieldsEqvSChopSYields*:
$\vdash$ *f syields* (*g syields h*) = ($f \frown g$) *syields h*
proof −
have *1*: $\vdash f \frown (g \frown (\neg h)) = (f \frown g) \frown (\neg h)$ **by** (*rule SChopAssoc*)
hence *2*: $\vdash f \frown (g \frown (\neg h)) = (f \frown g) \frown (\neg h)$ **by** *auto*
have *3*: $\vdash g \frown (\neg h) = (\neg \neg (g \frown (\neg h)))$ **by** *auto*
hence *4*: $\vdash f \frown (g \frown (\neg h)) = f \frown (\neg \neg (g \frown (\neg h)))$ **by** (*rule RightSChopEqvSChop*)
have *5*: $\vdash f \frown (\neg \neg (g \frown (\neg h))) = (f \frown g) \frown (\neg h)$ **using** *2 4* **by** *auto*
hence *6*: $\vdash f \frown (\neg$ (*g syields h*)) = $(f \frown g) \frown (\neg h)$ **by** (*simp add*: *syields-d-def*)
hence *7*: $\vdash (\neg (f \frown (\neg$ (*g syields h*)))) = $(\neg ((f \frown g) \frown (\neg h)))$ **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *syields-d-def*)

qed

lemma *EmptyYields*:
$\vdash$ *empty* *syields* $f = f$
proof –
have *1*: $\vdash$ *empty* $\frown$ $(\neg\ f) = (\neg\ f)$ **by** (*rule EmptySChop*)
hence *2*: $\vdash$ $(\neg$ (*empty* $\frown$ $(\neg\ f))) = f$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *NextSYields*:
$\vdash$ $(\bigcirc f)$ *syields* $g =$ *wnext* (f *syields* g)
proof –
have *1*: $\vdash (\bigcirc f)\frown (\neg\ g) = \bigcirc(f \frown (\neg\ g))$ **by** (*rule NextSChop*)
hence *2*: $\vdash (\neg\ ((\bigcirc f)\frown (\neg\ g))) = (\neg\ (\bigcirc(f \frown (\neg\ g))))$ **by** *auto*
hence *3*: $\vdash (\bigcirc f)$ *syields* $g = (\neg\ (\bigcirc(f \frown (\neg\ g))))$ **by** (*simp add*: *syields-d-def*)
have *4*: $\vdash (\neg(\ \bigcirc(f \frown (\neg\ g)))) =$ *wnext* $(\neg\ (f \frown (\neg\ g)))$ **by** (*auto simp*: *wnext-d-def*)
have *5*: $\vdash (\bigcirc f)$ *syields* $g =$ *wnext* $(\neg\ (f \frown (\neg\ g)))$ **using** *3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SkipSChopEqvNext*:
$\vdash$ *skip* $\frown f = \bigcirc f$
by (*meson NextSChopdef Prop11*)

lemma *SkipSYieldsEqvWeakNext*:
$\vdash$ *skip* *syields* $f =$ *wnext* f
proof –
have *1*: $\vdash$ *skip* $\frown (\neg\ f) = \bigcirc(\neg\ f)$ **by** (*rule SkipSChopEqvNext*)
hence *2*: $\vdash (\neg$ (*skip* $\frown (\neg\ f))) = (\neg(\ \bigcirc(\neg\ f)))$ **by** *auto*
have *3*: $\vdash (\neg\ (\bigcirc(\neg\ f))) =$ *wnext* f **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash (\neg$ (*skip* $\frown (\neg\ f))) =$ *wnext* f **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

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

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

lemma *MoreSChopImpMore*:
$\vdash$ *more* $\frown$ *f* $\longrightarrow$ *more*
proof −
have *1*: $\vdash$ ($\bigcirc$#*True*)$\frown$ *f* = $\bigcirc$(#*True*$\frown$ *f*) **by** (*rule NextSChop*)
have *2*: $\vdash$ $\bigcirc$(#*True*$\frown$ *f*) $\longrightarrow$ *more* **by** (*auto simp*: *more-defs next-defs sum.case-eq-if*)
have *3*: $\vdash$ ($\bigcirc$#*True*$\frown$ *f*) $\longrightarrow$ *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*metis more-d-def*)
qed

lemma *MoreSChopImpFmore*:
$\vdash$ *more* $\frown$ (*f* $\wedge$ *finite*) $\longrightarrow$ *fmore*
proof −
have *1*: $\vdash$ *more* $\frown$ (*f* $\wedge$ *finite*)= $\bigcirc$(#*True*$\frown$ (*f* $\wedge$ *finite*))
by (*simp add*: *NextSChop more-d-def*)
have *2*: $\vdash$ $\bigcirc$(#*True*$\frown$ (*f* $\wedge$ *finite*)) $\longrightarrow$ *fmore*
by (*auto simp*: *fmore-defs schop-defs finite-defs more-defs next-defs sum.case-eq-if*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopMoreImpMore*:
$\vdash$ *f*$\frown$ *more* $\longrightarrow$ *more*
proof −
have *1*: $\vdash$ *f* $\frown$ *more* $\longrightarrow$ $\Diamond$ *more* **by** (*rule SChopImpDiamond*)
have *2*: $\vdash$ $\Diamond$ *more* $\longrightarrow$ *more* **by** (*auto simp*: *more-defs sometimes-defs sum.case-eq-if*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreSChopEqvNextDiamond*:
$\vdash$ *more* $\frown$ *f* = $\bigcirc$($\Diamond$ *f*)
proof −
have *1*: $\vdash$ *more* $\frown$ *f* = ($\bigcirc$ #*True*)$\frown$ *f*
by (*simp add*: *Valid-def schop-defs more-defs next-defs finite-defs sum.case-eq-if*)
have *2*: $\vdash$ ($\bigcirc$ #*True*)$\frown$ *f* = $\bigcirc$(#*True*$\frown$ *f*) **by** (*rule NextSChop*)
have *3*: $\vdash$ *more* $\frown$ *f* = $\bigcirc$(#*True*$\frown$ *f*) **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis*
by (*metis TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *WeakNextBoxImpMoreSYields*:
$\vdash$ *more syields f* = *wnext*($\Box$ *f*)
proof −
have *1*: $\vdash$ *more* $\frown$ ($\neg$ *f*) = $\bigcirc$($\Diamond$ ($\neg$*f*)) **by** (*rule MoreSChopEqvNextDiamond*)
have *2*: $\vdash$ $\bigcirc$($\Diamond$ ($\neg$*f*)) = $\bigcirc$($\neg$($\Box$*f*)) **by** (*auto simp*: *always-d-def*)
have *3*: $\vdash$ $\bigcirc$($\neg$($\Box$*f*)) = ($\neg$ (*wnext*($\Box$ *f*))) **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash$ *more* $\frown$ ($\neg$ *f*) = ($\neg$(*more syields f*)) **by** (*simp add*: *syields-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotEqvSYieldsMore*:
$\vdash$ *finite* $\longrightarrow$ $(\neg\ f) = f$ *syields* *more*
proof −
have *1*: $\vdash$ *finite* $\longrightarrow f \frown$ *empty* $= f$ **by** (*rule SChopEmpty*)
hence *2*: $\vdash$ *finite* $\longrightarrow (\neg\ (f \frown$ *empty* $)) = (\neg\ f)$ **by** *auto*
have *3*: $\vdash$ *empty* $= (\neg$ *more*$)$ **by** (*auto simp*: *empty-d-def*)
hence *4*: $\vdash f \frown$ *empty* $= f \frown (\neg$ *more*$)$ **by** (*rule RightSChopEqvSChop*)
hence *5*: $\vdash (\neg\ (f \frown$ *empty* $)) = (\neg\ (f \frown (\neg$ *more* $)))$ **by** *auto*
have *6*: $\vdash$ *finite* $\longrightarrow (\neg\ f) = (\neg\ (f \frown (\neg$ *more*$)\ ))$ **using** *2 5* **by** *fastforce*
from *6* **show** *?thesis* **by** (*metis syields-d-def*)
qed

lemma *LeftSChopImpMoreRule*:
assumes $\vdash f \longrightarrow$ *more*
shows $\vdash f \frown g \longrightarrow$ *more*
proof −
have *1*: $\vdash f \longrightarrow$ *more* **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow$ *more* $\frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash$ *more* $\frown g \longrightarrow$ *more* **by** (*rule MoreSChopImpMore*)
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *LeftSChopImpFMoreRule*:
assumes $\vdash f \longrightarrow$ *fmore*
shows $\vdash f \frown (g \wedge$ *finite*$) \longrightarrow$ *fmore*
proof −
have *1*: $\vdash f \longrightarrow$ *fmore* **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown (g \wedge$ *finite*$) \longrightarrow$ *more* $\frown (g \wedge$ *finite*$)$
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite LeftSChopImpSChop Prop12 inteq-reflection*)
have *3*: $\vdash$ *more* $\frown (g \wedge$ *finite*$) \longrightarrow$ *fmore* **using** *MoreSChopImpFmore* **by** *fastforce*
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *RightSChopImpMoreRule*:
assumes $\vdash g \longrightarrow$ *more*
shows $\vdash f \frown g \longrightarrow$ *more*
proof −
have *1*: $\vdash g \longrightarrow$ *more* **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown$ *more* **by** (*rule RightSChopImpSChop*)
have *3*: $\vdash f \frown$ *more* $\longrightarrow$ *more* **by** (*rule SChopMoreImpMore*)
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NotDfEqvBfNot*:
$\vdash$ $(\neg\ (\ df\ \ f)) = bf\ (\neg\ f)$
proof −
have *1*: $\vdash f = (\neg\ \neg\ f)$ **by** *auto*
hence *2*: $\vdash$ $df\ \ f =\ df\ (\neg\ \neg\ f)$ **by** (*rule DfEqvDf*)
hence *3*: $\vdash (\neg\ \ (\ df\ \ f)) = (\neg\ (\ df\ (\neg\ \neg\ f)))$ **by** *auto*

from *3* **show** *?thesis* **by** (*simp add*: *bf-d-def*)
qed

lemma *SChopImpDf*:
$\vdash$ $f \frown g \longrightarrow$ *df* f
proof −
have *1*: $\vdash g \longrightarrow \#True$ **by** *auto*
hence *2*: $\vdash f \frown g \longrightarrow f \frown \#True$ **by** (*rule RightSChopImpSChop*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *TrueEqvTrueSChopTrue*:
$\vdash$ $\#True = \#True \frown \#True$
proof −
have *1*: $\vdash \#True \frown \#True \longrightarrow \#True$ **by** *auto*
have *2*: $\vdash \#True \longrightarrow \#True \frown \#True$
by (*metis DfState Initprop*(*4*) *df-d-def int-eq-true int-iffD1 inteq-reflection*)
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *DfEqvDfDf*:
$\vdash$ *df* $f =$ *df* (*df* f)
proof −
have *1*: $\vdash \#True = \#True \frown \#True$ **by** (*rule TrueEqvTrueSChopTrue*)
hence *2*: $\vdash f \frown \#True = f \frown (\#True \frown \#True)$ **by** (*rule RightSChopEqvSChop*)
have *3*: $\vdash f \frown (\#True \frown \#True) = (f \frown \#True) \frown \#True$ **by** (*rule SChopAssoc*)
have *4*: $\vdash f \frown \#True = (f \frown \#True) \frown \#True$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis df-d-def*)
qed

lemma *BfEqvBfBf*:
$\vdash$ *bf* $f =$ *bf*(*bf* f)
proof −
have *1*: $\vdash$ *df* $(\neg\ f) =$ *df*(*df* $(\neg\ f))$ **by** (*rule DfEqvDfDf*)
have *2*: $\vdash$ *df* $(\neg\ f) = (\neg$ (*bf* $f))$ **by** (*rule DfNotEqvNotBf*)
hence *3*: $\vdash$ *df* (*df* $(\neg\ f)) =$ *df* $(\neg$ (*bf* $f))$ **by** (*rule DfEqvDf*)
have *4*: $\vdash$ *df* $(\neg\ f) =$ *df* $(\neg($ *bf* $f))$ **using** *1 3* **by** *fastforce*
hence *5*: $\vdash (\neg$ (*df* $(\neg\ f))) = (\neg$ (*df* $(\neg$ (*bf* $f))))$ **by** *fastforce*
from *5* **show** *?thesis* **by** (*metis bf-d-def*)
qed

lemma *BfImpBfBf*:
$\vdash$ *bf* $f \longrightarrow$ *bf*(*bf* f)
proof −
have *1*: $\vdash$ *bf*(*bf* f) $=$ *bf* f **using** *BfEqvBfBf* **by** *fastforce*
from *1* **show** *?thesis* **by** (*simp add*: *int-iffD2*)

qed

lemma *DfOrEqv*:
$\vdash$ *df* $(f \vee g) =$ $($*df* $f \vee$ *df* $g)$
proof −
have *1*: $\vdash (f \vee g)\frown \#True = (f\frown \#True \vee g\frown \#True)$ **by** (*rule OrSChopEqv*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfAndA*:
$\vdash$ *df* $(f \wedge g) \longrightarrow$ *df* f
proof −
have *1*: $\vdash (f \wedge g)\frown \#True \longrightarrow f\frown \#True$ **by** (*rule AndSChopA*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfAndB*:
$\vdash$ *df* $(f \wedge g) \longrightarrow$ *df* g
proof −
have *1*: $\vdash (f \wedge g)\frown \#True \longrightarrow g\frown \#True$ **by** (*rule AndSChopB*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfAndImpAnd*:
$\vdash$ *df* $(f \wedge g) \longrightarrow$ *df* $f \wedge$ *df* g
proof −
have *1*: $\vdash$ *df* $(f \wedge g) \longrightarrow$ *df* f **by** (*rule DfAndA*)
have *2*: $\vdash$ *df* $(f \wedge g) \longrightarrow$ *df* g **by** (*rule DfAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DfSkipEqvMore*:
$\vdash$ *df skip* $=$ *more*
proof −
have *1*: $\vdash$ *skip* $\frown \#True = \bigcirc\#True$ **by** (*rule SkipSChopEqvNext*)
have *2*: $\vdash \bigcirc\#True =$ *more* **by** (*auto simp*: *more-d-def*)
have *3*: $\vdash$ *skip* $\frown \#True =$ *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfMoreEqvMore*:
$\vdash$ *df more* $=$ *more*
proof −
have *1*: $\vdash$ *df* $(\bigcirc \#True\ ) = \bigcirc($ *df* $\#True)$ **by** (*rule DfNext*)
have *2*: $\vdash \bigcirc($ *df* $\#True) \longrightarrow$ *more* **by** (*auto simp*: *next-defs di-defs more-defs sum.case-eq-if*)
have *3*: $\vdash$ *df*$(\bigcirc \#True) \longrightarrow$ *more* **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *df more* $\longrightarrow$ *more* **by** (*simp add*: *more-d-def*)
have *5*: $\vdash$ *more* $\longrightarrow$ *df more*

by (*metis 1 4 TrueEqvTrueSChopTrue df-d-def inteq-reflection more-d-def*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *DfIfEqvRule*:
assumes $\vdash \ \ f = \ if_i \ (init\ w) \ then \ g \ else \ h$
shows $\vdash \ df \ f = \ if_i \ (init\ w) \ then \ (\ df \ g) \ else \ (df \ h)$
proof −
have *1*: $\vdash f = \ if_i \ (init\ w) \ then \ g \ else \ h$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \frown \#True = \ if_i \ (init\ w) \ then \ (g \frown \#True) \ else \ (h \frown \#True)$
by (*rule IfSChopEqvRule*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *SDaNotEqvNotSBa*:
$\vdash \ \ sda\ (\neg \ f) = (\neg \ (\ sba \ f))$
proof −
have *1*: $\vdash \ sba \ f = (\neg \ (\ sda\ (\neg \ f)))$ **by** (*simp add*: *sba-d-def*)
from *1* **show** *?thesis* **by** *fastforce*
qed

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

lemma *SDaEqvNotSBaNot*:
$\vdash \ \ sda \ f = (\neg \ (\ sba\ (\neg \ f)))$
proof −
have *1*: $\vdash \ sba\ (\neg \ f) = (\neg \ (\ sda\ (\neg \neg \ f)))$ **by** (*simp add*: *sba-d-def*)
hence *2*: $\vdash \ sda\ (\neg \neg \ f) = (\neg(\ sba\ (\neg \ f)))$ **by** *fastforce*
have *3*: $\vdash f = (\neg \neg \ f)$ **by** *simp*
hence *4*: $\vdash \ sda \ f = \ sda \ (\neg \neg \ f)$ **by** (*rule SDaEqvSDa*)
from *2 4* **show** *?thesis* **by** *simp*
qed

lemma *SBaElim*:
$\vdash \ \ sba\ f \wedge finite \longrightarrow f$
proof −
have *1*: $\vdash \ sba \ f = \Box(bf \ f)$ **by** (*rule SBaEqvBtBf*)
have *2*: $\vdash bf \ f \wedge finite \longrightarrow f$ **by** (*rule BfElim*)
hence *3*: $\vdash \Box(bf \ f \wedge finite \longrightarrow f)$ **by** (*rule BoxGen*)
have *4*: $\vdash \Box(bf \ f \wedge finite \longrightarrow f) \longrightarrow \Box(bf \ f \wedge finite) \longrightarrow \Box \ f$ **by** (*rule BoxImpDist*)
have *5*: $\vdash \Box(bf \ f \wedge finite) \longrightarrow \Box \ f$ **using** *3 4 MP* **by** *fastforce*
have *6*: $\vdash \Box(bf \ f \wedge finite) = (\Box(bf \ f) \wedge finite)$
by (*metis* (*no-types*, *lifting*) *BoxEqvFiniteYields FiniteChopInfEqvInf NotChopEqvYieldsNot YieldsAndYieldsEqvYieldsAnd finite-d-def inteq-reflection*)

have *7*: $\vdash \Box\ f \longrightarrow f$ **by** (*rule BoxElim*)
from *1 5 6 7* **show** *?thesis* **using** *SBaImpBt lift-imp-trans* **by** *metis*
qed

lemma *SDaIntro*:
$\vdash\ \ f \wedge finite \longrightarrow\ sda\ \ f$
proof −
have *1*: $\vdash\ sba\ \ (\neg\ \ f) \wedge finite \longrightarrow (\neg\ \ f)$ **by** (*rule SBaElim*)
hence *2*: $\vdash \neg\ \neg\ \ f \longrightarrow \neg\ (\ sba\ \ (\neg\ \ f) \wedge finite)$ **by** *fastforce*
have *3*: $\vdash f = (\neg\ \neg\ \ f)$ **by** *simp*
have *4*: $\vdash\ sda\ \ f = (\neg\ \ (\ sba\ \ (\neg\ \ f)))$ **by** (*rule SDaEqvNotSBaNot*)
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaGen*:
assumes $\vdash\ \ f$
shows $\vdash\ sba\ \ f$
proof −
have *1*: $\vdash\ \ f$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ f$ **by** (*rule BoxGen*)
hence *3*: $\vdash bf(\ \Box\ f)$ **by** (*rule BfGen*)
have *4*: $\vdash\ sba\ \ f = bf\ (\Box\ f)$ **by** (*rule SBaEqvBfBt*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpDist*:
$\vdash\ sba\ (f \longrightarrow g) \longrightarrow\ sba\ \ f \longrightarrow\ sba\ \ g$
proof −
have *1*: $\vdash bf\ (f \longrightarrow g) \longrightarrow (bf\ \ f \longrightarrow bf\ \ g)$ **by** (*rule BfImpDist*)
hence *2*: $\vdash \Box(bf\ (f \longrightarrow g) \longrightarrow (bf\ \ f \longrightarrow bf\ \ g))$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box(bf\ (f \longrightarrow g) \longrightarrow (bf\ \ f \longrightarrow bf\ \ g))$
$\longrightarrow$
$(\Box\ (bf\ (f \longrightarrow g)) \longrightarrow (\Box(bf\ \ f) \longrightarrow \Box(bf\ \ g)))$
by (*meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09*)
have *4*: $\vdash \Box(bf\ (f \longrightarrow g)) \longrightarrow (\Box(bf\ \ f) \longrightarrow \Box(bf\ \ g))$ **using** *2 3 MP* **by** *fastforce*
have *5*: $\vdash\ sba\ (f \longrightarrow g) = \Box(bf\ (f \longrightarrow g))$ **by** (*rule SBaEqvBtBf*)
have *6*: $\vdash\ sba\ \ f = \Box(bf\ \ f)$ **by** (*rule SBaEqvBtBf*)
have *7*: $\vdash\ sba\ \ g = \Box(bf\ \ g)$ **by** (*rule SBaEqvBtBf*)
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaAndEqv*:
$\vdash\ \ sba\ (f \wedge g) =\ (sba\ \ f \wedge\ sba\ \ g)$
proof −
have *1*: $\vdash\ \ sba\ (f \wedge g) =\ \Box(bf\ (f \wedge g))$
by (*rule SBaEqvBtBf*)
have *2*: $\vdash\ bf\ (f \wedge g) = (bf\, f \wedge bf\, g)$
by (*auto simp*: *bf-defs sum.case-eq-if*)
hence *3*: $\vdash \Box(bf\ (f \wedge g)) = \Box(bf\, f \wedge bf\, g)$
using *BoxEqvBox* **by** *blast*

have *4*: $\vdash \Box(bf\, f \wedge bf\, g) = (\Box(bf\, f) \wedge \Box(bf\, g))$
  by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
have *5*: $\vdash sba\ \ f = \Box(bf\ \ f)$
  by (*rule SBaEqvBtBf*)
have *6*: $\vdash\ sba\ \ g = \Box(bf\ \ g)$
  by (*rule SBaEqvBtBf*)
from *1 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpSBaEqvSBa*:
$\vdash\ sba\ (f = g) \longrightarrow (sba\ \ f = sba\ \ g)$
proof −
have *1*: $\vdash\ sba\ (f \longrightarrow g) \longrightarrow sba\ \ f \longrightarrow sba\ \ g$ **by** (*rule SBaImpDist*)
have *2*: $\vdash\ sba\ (g \longrightarrow f) \longrightarrow sba\ \ g \longrightarrow sba\ \ f$ **by** (*rule SBaImpDist*)
have *3*: $\vdash\ (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$
by *auto*
hence *31*: $\vdash sba(f = g) = sba\ ((f \longrightarrow g) \wedge (g \longrightarrow f))$
using *inteq-reflection* **by** *force*
have *4*: $\vdash sba\ ((f \longrightarrow g) \wedge (g \longrightarrow f)) = (sba((f \longrightarrow g)) \wedge sba((g \longrightarrow f)))$
by (*rule SBaAndEqv*)
have *5*: $\vdash ((sba\ \ f \longrightarrow sba\ \ g) \wedge (sba\ \ g \longrightarrow sba\ \ f)) = (sba\ \ f = sba\ \ g)$ **by** *auto*
from *1 2 31 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpSBa*:
assumes $\vdash\ f \longrightarrow g$
shows $\vdash sba\ \ f \longrightarrow sba\ \ g$
using *SBaGen SBaImpDist MP assms* **by** *metis*

lemma *SBaEqvSBa*:
assumes $\vdash\ f = g$
shows $\vdash sba\ \ f = sba\ \ g$
using *SBaGen SBaImpSBaEqvSBa MP assms* **by** *metis*

lemma *SDaImpSDa*:
assumes $\vdash\ f \longrightarrow g$
shows $\vdash sda\ \ f \longrightarrow sda\ \ g$
using *assms* **by** (*metis SDaEqvDtDf DfAndB DiamondImpDiamond inteq-reflection Prop10*)

lemma *SDaEqvSDaSDa*:
$\vdash\ sda\ \ f = sda\ (\ sda\ \ f)$
proof −
have *1*: $\vdash\ sda\ \ f = \Diamond(\ df\ \ f)$
  by (*rule SDaEqvDtDf*)
have *2*: $\vdash\ df\ \ f = (df\ (\ df\ \ f))$
  by (*rule DfEqvDfDf*)
hence *3*: $\vdash \Diamond\ (\ df\ \ f) = \Diamond\ (df\ \ (df\ \ f))$
  by (*rule DiamondEqvDiamond*)
have *4*: $\vdash \Diamond\ \ (df\ \ f) = \Diamond(\Diamond\ (df\ \ (df\ \ f)))$

using *DiamondEqvDiamondDiamond DfEqvDfDf* **using** *3* **by** *fastforce*
have *5*: $\vdash \Diamond\ (df\ (df\ f)) = df\ (\Diamond\ (df\ f))$
by (*rule DtDfEqvDfDt*)
hence *6*: $\vdash \Diamond(\Diamond\ (df\ (df\ f))) = \Diamond\ (df\ (\Diamond\ (df\ f)))$
by (*rule DiamondEqvDiamond*)
have *7*: $\vdash\ sda\ f = \Diamond\ (df(\ \Diamond\ (\ df\ f)))$
using *1 3 4 6* **by** *fastforce*
have *8*: $\vdash\ sda\ (\Diamond\ (\ df\ f)) = \Diamond(\ df\ (\Diamond\ (df\ f)))$
by (*rule SDaEqvDtDf*)
have *9*: $\vdash\ sda\ (\ sda\ f) = sda\ (\Diamond\ (df\ f))$
using *1* **by** (*rule SDaEqvSDa*)
from *7 8 9* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaEqvSBaSBa*:
$\vdash\ sba\ f = sba\ (sba\ f)$
proof −
have *1*: $\vdash\ sda\ (\neg\ f) = sda\ (sda\ (\neg\ f))$ **by** (*rule SDaEqvSDaSDa*)
have *2*: $\vdash\ sda\ (sda\ (\neg\ f)) = (\neg\ (sba\ (\neg\ (sda\ (\neg\ f)))))$ **by** (*rule SDaEqvNotSBaNot*)
have *3*: $\vdash (\neg\ (sda\ (sda\ (\neg\ f)))) = sba\ (\neg\ (sda\ (\neg\ f)))$ **by** (*auto simp*: *sba-d-def*)
have *4*: $\vdash (\neg\ (sda\ (\neg\ f))) = sba\ (\neg\ (sda\ (\neg\ f)))$ **using** *1 2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis sba-d-def*)
qed

lemma *SBaLeftSChopImpSChop*:
$\vdash\ sba\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$
proof −
have *1*: $\vdash\ sba\ (f \longrightarrow f1) \longrightarrow bf\ (f \longrightarrow f1)$ **by** (*rule SBaImpBf*)
have *2*: $\vdash bf\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaLeftSChopImpSChop*:
$\vdash\ ba\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$
proof −
have *1*: $\vdash\ ba\ (f \longrightarrow f1) \longrightarrow bf\ (f \longrightarrow f1)$ **by** (*rule BaImpBf*)
have *2*: $\vdash bf\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaRightSChopImpSChop*:
$\vdash\ sba\ (g \longrightarrow g1) \land finite \longrightarrow f \frown g \longrightarrow f \frown g1$
proof −
have *1*: $\vdash\ sba\ (g \longrightarrow g1) \land finite \longrightarrow \Box(g \longrightarrow g1)$ **by** (*rule SBaImpBt*)
have *2*: $\vdash \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BoxSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaRightSChopImpSChop*:

$\vdash$ *ba* $(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$
proof −
 have *1*: $\vdash$ *ba* $(g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **by** (*rule BaImpBt*)
 have *2*: $\vdash \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BoxSChopImpSChop*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndSBaImport*:
 $\vdash (f \frown f1) \wedge$ *sba* $g \wedge$ *finite* $\longrightarrow (f \wedge g) \frown (f1 \wedge g)$
proof −
 have *1*: $\vdash$ *sba* $g \wedge$ *finite* $\wedge (f \frown f1) \longrightarrow (g \wedge f) \frown (g \wedge f1)$ **by** (*rule SBaAndSChopImport*)
 have *2*: $\vdash (g \wedge f) \frown (g \wedge f1) = (f \wedge g) \frown (f1 \wedge g)$ **by** (*rule AndSChopAndCommute*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndBaImport*:
 $\vdash (f \frown f1) \wedge$ *ba* $g \longrightarrow (f \wedge g) \frown (f1 \wedge g)$
proof −
 have *1*: $\vdash$ *ba* $g \wedge (f \frown f1) \longrightarrow (g \wedge f) \frown (g \wedge f1)$ **by** (*rule BaAndSChopImport*)
 have *2*: $\vdash (g \wedge f) \frown (g \wedge f1) = (f \wedge g) \frown (f1 \wedge g)$ **by** (*rule AndSChopAndCommute*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndSChopImportA*:
$\vdash$ *ba* $f \wedge g \frown g1 \longrightarrow (f \wedge g) \frown g1$
by (*meson BaAndSChopImport SChopAndB lift-imp-trans*)

lemma *BaAndSChopImportB*:
$\vdash$ *ba* $f \wedge g \frown g1 \longrightarrow (f \wedge g) \frown ($*ba* $f \wedge g1)$
proof −
 have *1*: $\vdash$ *ba* $f =$ *ba* $($*ba* $f)$
  by (*simp add*: *BaEqvBaBa*)
 have *2*: $\vdash$ *ba* $($*ba* $f) \wedge g \frown g1 \longrightarrow g \frown ($*ba* $f \wedge g1)$
  by (*metis AndSChopB BaAndSChopImport lift-imp-trans*)
 have *3*: $\vdash$ *ba* $f \wedge g \frown ($*ba* $f \wedge g1) \longrightarrow (f \wedge g) \frown ($*ba* $f \wedge g1)$
  by (*simp add*: *BaAndSChopImportA*)
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpSBaImpSBaAnd*:
$\vdash$ *sba* $h \longrightarrow$ *sba*$(g \longrightarrow$ *sba* $h \wedge g)$
proof −
 have *1*: $\vdash$ *sba* $h \longrightarrow (g \longrightarrow$ *sba* $h \wedge g)$ **by** *fastforce*
 hence *2*: $\vdash$ *sba*$($*sba* $h) \longrightarrow$ *sba*$(g \longrightarrow$ *sba* $h \wedge g)$ **by** (*rule SBaImpSBa*)
 have *3*: $\vdash$ *sba* $h =$ *sba*$($*sba* $h)$ **by** (*rule SBaEqvSBaSBa*)
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaSChopImpSChopSBa*:
$\vdash$ *sba f* $\wedge$ *finite* $\longrightarrow$ *g* $\frown$ *g1* $\longrightarrow$ *g* $\frown$ ((*sba f*) $\wedge$ *g1*)
proof –
have *1*: $\vdash$ *sba f* $\longrightarrow$ *sba* (*g1* $\longrightarrow$ (*sba f*) $\wedge$ *g1*) **by** (*rule SBaImpSBaImpSBaAnd*)
have *2*: $\vdash$ *sba* (*g1* $\longrightarrow$ *sba f* $\wedge$ *g1*) $\wedge$ *finite* $\longrightarrow$ *g* $\frown$ *g1* $\longrightarrow$ *g* $\frown$ (*sba f* $\wedge$ *g1*)
by (*rule SBaRightSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaSChopImpSChopBa*:
$\vdash$ *ba f* $\longrightarrow$ *g* $\frown$ *g1* $\longrightarrow$ *g* $\frown$ ((*ba f*) $\wedge$ *g1*)
proof –
have *1*: $\vdash$ *ba f* $\longrightarrow$ *ba* (*g1* $\longrightarrow$ (*ba f*) $\wedge$ *g1*) **by** (*rule BaImpBaImpBaAnd*)
have *2*: $\vdash$ *ba* (*g1* $\longrightarrow$ *ba f* $\wedge$ *g1*) $\longrightarrow$ *g* $\frown$ *g1* $\longrightarrow$ *g* $\frown$ (*ba f* $\wedge$ *g1*)
by (*rule BaRightSChopImpSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DfNotSBaImpNotSBa*:
$\vdash$ *df* ($\neg$ (*sba f*)) $\longrightarrow$ $\neg$ (*sba f*)
proof –
have *1*: $\vdash$ *sba f* = *sba*(*sba f*) **by** (*rule SBaEqvSBaSBa*)
have *2*: $\vdash$ *sba* (*sba f*) $\longrightarrow$ *bf* (*sba f*) **by** (*rule SBaImpBf*)
have *3*: $\vdash$ *sba f* $\longrightarrow$ *bf* (*sba f*) **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *sba f* $\longrightarrow$ $\neg$ (*df* ($\neg$ (*sba f*))) **by** (*simp add*: *bf-d-def*)
from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *DfNotBaImpNotBa*:
$\vdash$ *df* ($\neg$ (*ba f*)) $\longrightarrow$ $\neg$ (*ba f*)
proof –
have *1*: $\vdash$ *ba f* = *ba*(*ba f*) **by** (*rule BaEqvBaBa*)
have *2*: $\vdash$ *ba* (*ba f*) $\longrightarrow$ *bf* (*ba f*) **by** (*rule BaImpBf*)
have *3*: $\vdash$ *ba f* $\longrightarrow$ *bf* (*ba f*) **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *ba f* $\longrightarrow$ $\neg$ (*df* ($\neg$ (*ba f*))) **by** (*simp add*: *bf-d-def*)
from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotSBaSChopImpNotSBa*:
$\vdash$ ($\neg$ (*sba f*)) $\frown$ *g* $\longrightarrow$ $\neg$ (*sba f*)
proof –
have *1*: $\vdash$ ($\neg$ (*sba f*)) $\frown$ *g* $\longrightarrow$ *df* ($\neg$ (*sba f*)) **by** (*rule SChopImpDf*)
have *2*: $\vdash$ *df* ($\neg$ (*sba f*)) $\longrightarrow$ $\neg$ (*sba f*) **by** (*rule DfNotSBaImpNotSBa*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NotBaSChopImpNotSBa*:
$\vdash$ ($\neg$ (*ba f*)) $\frown$ *g* $\longrightarrow$ $\neg$ (*ba f*)
proof –
have *1*: $\vdash$ ($\neg$ (*ba f*)) $\frown$ *g* $\longrightarrow$ *df* ($\neg$ (*ba f*)) **by** (*rule SChopImpDf*)

have *2*: $\vdash$ *df* ($\neg$ (*ba* *f*)) $\longrightarrow$ $\neg$ (*ba* *f*) **by** (*rule DfNotBaImpNotBa*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *DiamondSFinImpSFin*:
$\vdash$ $\diamond$ (*sfin f*) $\longrightarrow$ *sfin f*
proof −
have *1*: $\vdash$ *sfin f* = #*True*$\frown$(*f* $\wedge$ *empty*)
by (*rule SFinEqvTrueSChopAndEmpty*)
hence *2*: $\vdash$ $\diamond$ (*sfin f*) = #*True*$\frown$(#*True*$\frown$(*f* $\wedge$ *empty*))
using *DiamondSChopdef inteq-reflection* **by** *force*
have *3*: $\vdash$ #*True*$\frown$(#*True*$\frown$(*f* $\wedge$ *empty*)) = (#*True*$\frown$#*True*)$\frown$(*f* $\wedge$ *empty*)
by (*rule SChopAssoc*)
have *4*: $\vdash$ (#*True*$\frown$#*True*)$\frown$(*f* $\wedge$ *empty*) $\longrightarrow$ #*True*$\frown$(*f* $\wedge$ *empty*)
using *1 2 3*
by (*metis SChopImpDiamond TrueEqvTrueSChopTrue inteq-reflection*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopSFinImpSFin*:
$\vdash$ *f* $\frown$ *sfin* (*init w*) $\longrightarrow$ *sfin* (*init w*)
proof −
have *1*: $\vdash$ *f* $\frown$ *sfin* (*init w*) $\longrightarrow$ $\diamond$ (*sfin* (*init w*)) **by** (*rule SChopImpDiamond*)
have *2*: $\vdash$ $\diamond$ (*sfin* (*init w*)) $\longrightarrow$ *sfin* (*init w*) **by** (*rule DiamondSFinImpSFin*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *SFinImpSYieldsSFin*:
$\vdash$ *sfin* (*init w*) $\longrightarrow$ *f syields* (*sfin* (*init w*))
proof −
have *1*: $\vdash$ *f*$\frown$ (*sfin* (*init* ($\neg$ *w*))) $\longrightarrow$ (*sfin* (*init* ($\neg$ *w*)))
by (*simp add*: *SChopSFinImpSFin*)
have *2*: $\vdash$ *finite* $\longrightarrow$ ($\neg$ (*sfin* (*init w*))) = (*sfin* (*init* ($\neg$ *w*)))
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
hence *3*: $\vdash$ *finite* $\longrightarrow$ *f* $\frown$ ($\neg$ (*sfin* (*init w*))) = *f*$\frown$ (*sfin* (*init* ($\neg$ *w*)))
using *FiniteRightSChopEqvSChop* **by** *blast*
have *4*: $\vdash$ *f*$\frown$ ($\neg$ (*sfin* (*init w*))) $\wedge$ *finite* $\longrightarrow$ ($\neg$ (*sfin* (*init w*)))
using *1 2 3* **by** *fastforce*
hence *5*: $\vdash$ *sfin* (*init w*) $\longrightarrow$ $\neg$ (*f*$\frown$ ($\neg$ (*sfin* (*init w*))))
by (*metis SChopImpDiamond SFinImpBox always-d-def int-simps*(*32*) *inteq-reflection lift-imp-trans*)
from *5* **show** *?thesis*
by (*simp add*: *syields-d-def*)
qed

lemma *SChopAndSFin*:
$\vdash$ ((*f*$\frown$ *g*) $\wedge$ (*sfin* (*init w*))) = *f*$\frown$ (*g* $\wedge$ (*sfin* (*init w*)))
proof −
have *1*: $\vdash$ *sfin* (*init w*) $\longrightarrow$ *f syields* (*sfin* (*init w*))

by (*rule SFinImpSYieldsSFin*)
have *2*: ⊢ (*f* ⌢ *g*) ∧ (*sfin* (*init w*)) ⟶ (*f* ⌢ *g*) ∧ *f syields* (*sfin* (*init w*))
using *1* **by** *fastforce*
have *3*: ⊢ *f* ⌢ *g* ∧ *f syields* (*sfin* (*init w*)) ⟶
f ⌢ (*g* ∧ (*sfin* (*init w*)))
using *SChopAndSYieldsImp* **by** *blast*
have *4*: ⊢ (*f* ⌢ *g*) ∧ (*sfin* (*init w*))⟶ *f* ⌢ (*g* ∧ *sfin* (*init w*))
using *2 3* **by** (*metis* (*mono-tags*, *lifting*) *lift-imp-trans*)
from *4* **show** *?thesis*
by (*simp add*: *Prop12 SChopAndA SChopSFinExportA int-iffI*)
qed

lemma *SChopAndNotSFin*:
⊢ (*f* ⌢ *g* ∧ ¬ (*sfin* (*init w*)) ∧ *finite*) = *f* ⌢ (*g* ∧ ¬ (*sfin* (*init w*)) ∧ *finite*)
proof −
have *1*: ⊢ (*f* ⌢ *g* ∧ *sfin* (*init* (¬ *w*))) = *f* ⌢ (*g* ∧ *sfin* (*init* (¬ *w*)))
by (*rule SChopAndSFin*)
have *2*: ⊢ (*sfin* (*init* (¬ *w*)) ∧ *finite*)= ((¬ (*sfin* (*init w*))) ∧ *finite*)
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
hence *3*: ⊢ (*g* ∧ *sfin* (*init* (¬ *w*))) = (*g* ∧ ¬(*sfin* (*init w*)) ∧ *finite*)
using *DiamondEmptyEqvFinite SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond*
by *fastforce*
hence *4*: ⊢ *f* ⌢ (*g* ∧ *sfin* (*init* (¬ *w*))) = *f* ⌢ (*g* ∧ ¬ (*sfin* (*init w*)) ∧ *finite*)
using *RightSChopEqvSChop* **by** *blast*
from *1 2 4* **show** *?thesis*
using *DiamondEmptyEqvFinite SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond* **by** *fastforce*
qed

lemma *SFinSChopChain*:
⊢ (((*init w*) ∧ *finite* ⟶ *sfin* (*init w1*)))⌢
(((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*)))
∧ *finite*
⟶ (((*init w*) ∧ *finite* ⟶ *sfin* (*init w2*)))
proof −
have *1*: ⊢ (*init w*) ∧ *finite* ∧
((*init w*) ∧ *finite* ⟶ *sfin* (*init w1*))⌢
((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))
⟶
((*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *sfin* (*init w1*)))⌢
(((*init w1*) ∧ *finite*⟶ *sfin* (*init w2*)) ∧ *finite*)
by (*metis* (*no-types*, *lifting*) *ChopAndFiniteDist StateAndSChop int-iffD2*
inteq-reflection lift-and-com schop-d-def)
have *2*: ⊢ (*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite*⟶ *sfin* (*init w1*)) ⟶
sfin (*init w1*)
by *auto*
have *3*: ⊢ ((*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *sfin* (*init w1*)))⌢
(((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*)) ∧ *finite*)
⟶
(*sfin* (*init w1*))⌢ (((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*)) ∧ *finite*)
using *2 LeftSChopImpSChop* **by** *blast*

have *4*: ⊢ (*sfin* (*init w1*))⌢ (((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))) =
 ◇((*init w1*) ∧ ((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*)))
 using *SFinSChopEqvDiamond* **by** *blast*
have *41*: ⊢ ((*init w1*) ∧ *finite* ∧ ((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))) ⟶ *sfin* (*init w2*)
 by *auto*
have *42*: ⊢ ◇((*init w1*) ∧ *finite* ∧((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))) ⟶ ◇(*sfin*(*init w2*))
 using *41 DiamondImpDiamond* **by** *blast*
have *5*: ⊢ ◇(*sfin*(*init w2*)) ⟶ *sfin* (*init w2*)
 using *DiamondSFinImpSFin* **by** *blast*
have *6*: ⊢ (*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *sfin* (*init w1*))⌢
 ((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))
 ⟶ *sfin* (*init w2*)
 using *1 3 4 5 42*
 by (*metis* (*no-types*, *lifting*) *SFinSChopEqvDiamond inteq-reflection lift-and-com lift-imp-trans*)
from *6* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopRule*:
assumes ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ *sfin* (*init w1*)
 ⊢ (*init w1*)∧ *f1* ∧ *finite* ⟶ *sfin* (*init w2*)
shows ⊢ (*init w*) ∧ (*f* ⌢ *f1*) ∧ *finite* ⟶ *sfin* (*init w2*)
proof −
have *1*: ⊢ (*init w*) ∧ (*f* ⌢ *f1*) ∧ *finite* ⟶ ((*init w*) ∧ *f*)⌢ (*f1* ∧ *finite*)
 using *StateAndSChopImport*
 by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SChopAssoc SFinAndSChop SFinEqvTrueSChopAndEmpty StateAndEmptySChop TrueSChopEqvDiamond inteq-reflection*)
have *2*: ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ *sfin* (*init w1*) **using** *assms* **by** *auto*
hence *3*: ⊢ ((*init w*) ∧ *f*)⌢ (*f1* ∧ *finite*) ⟶ (*sfin* (*init w1*))⌢ (*f1* ∧ *finite*)
 by (*simp add*: *schop-d-def*)
 (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty LeftChopImpChop Prop10 Prop12 SFinAndSChopImport SFinEqvTrueSChopAndEmpty StateAndEmptySChop TrueSChopEqvDiamond inteq-reflection lift-and-com*)
have *4*: ⊢ (*sfin* (*init w1*))⌢ (*f1* ∧ *finite*) = ◇((*init w1*) ∧ *f1* ∧ *finite*)
 by (*rule SFinSChopEqvDiamond*)
have *5*: ⊢ (*init w1*) ∧ *f1* ∧ *finite* ⟶ *sfin* (*init w2*) **using** *assms* **by** *auto*
hence *6*: ⊢ ◇((*init w1*) ∧ *f1* ∧ *finite*) ⟶ ◇ (*sfin* (*init w2*)) **by** (*rule DiamondImpDiamond*)
have *7*: ⊢ ◇(*sfin* (*init w2*)) ⟶ *sfin* (*init w2*) **using** *DiamondSFinImpSFin* **by** *blast*
from *1 3 4 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopRep*:
assumes ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ *f1* ∧ *sfin* (*init w1*)
 ⊢ (*init w1*) ∧ *g* ∧ *finite* ⟶ *g1*
shows ⊢ (*init w*) ∧ (*f* ⌢ *g*) ∧ *finite* ⟶ (*f1*⌢ *g1*)
proof −
have *1*: ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ (*f1* ∧ *sfin* (*init w1*)) **using** *assms* **by** *auto*
hence *2*: ⊢ (*init w*) ∧ (*f* ⌢ (*g* ∧ *finite*)) ⟶ (*f1* ∧ *sfin* (*init w1*))⌢ (*g* ∧ *finite*)
 by (*metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop12 SChopSFinExportA SFinEqvTrueSChopAndEmpty StateAndChopImpChopRule StateAndEmptySChop*

TrueSChopEqvDiamond inteq-reflection schop-d-def)
have *3*: $\vdash$ (*f1* $\wedge$ *sfin* (*init w1*))$\frown$ (*g* $\wedge$ *finite*) = *f1*$\frown$ ((*init w1*) $\wedge$ (*g* $\wedge$ *finite*))
using *AndSFinSChopEqvStateAndSChop* **by** *blast*
have *4*: $\vdash$ (*init w1*)$\wedge$ *g* $\wedge$ *finite* $\longrightarrow$ *g1* **using** *assms* **by** *auto*
hence *5*: $\vdash$ *f1*$\frown$ ((*init w1*) $\wedge$ *g* $\wedge$ *finite*) $\longrightarrow$ *f1*$\frown$ *g1*
using *RightSChopImpSChop* **by** *blast*
from *2 3 5* **show** *?thesis*
by (*metis* (*no-types*, *lifting*) *ChopAndFiniteDist Prop10 Prop12 int-eq int-iffD2 lift-and-com schop-d-def*)
qed

lemma *SChopRepAndSFin*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\wedge$ *finite* $\longrightarrow$ *f1* $\wedge$ *sfin* (*init w1*)
$\vdash$ (*init w1*) $\wedge$ *g* $\wedge$ *finite* $\longrightarrow$ *g1* $\wedge$ *sfin* (*init w2*)
shows $\vdash$ (*init w*) $\wedge$ (*f*$\frown$ *g*) $\wedge$ *finite* $\longrightarrow$ (*f1*$\frown$ *g1*) $\wedge$ *sfin* (*init w2*)
proof −
have *1*: $\vdash$ (*init w*) $\wedge$ *f* $\wedge$ *finite* $\longrightarrow$ *f1* $\wedge$ *sfin* (*init w1*) **using** *assms* **by** *auto*
have *2*: $\vdash$ (*init w1*) $\wedge$ *g* $\wedge$ *finite* $\longrightarrow$ *g1* $\wedge$ *sfin* (*init w2*) **using** *assms* **by** *auto*
have *3*: $\vdash$ (*init w*) $\wedge$ (*f*$\frown$ *g*) $\wedge$ *finite* $\longrightarrow$ *f1*$\frown$ (*g1* $\wedge$ *sfin* (*init w2*))
using *1 2* **by** (*rule SChopRep*)
have *4*: $\vdash$ *f1*$\frown$ (*g1* $\wedge$ *sfin* (*init w2*)) $\longrightarrow$ *f1*$\frown$ *g1* **by** (*rule SChopAndA*)
have *5*: $\vdash$ *f1*$\frown$ (*g1* $\wedge$ *sfin* (*init w2*)) $\longrightarrow$ *f1*$\frown$ *sfin* (*init w2*) **by** (*rule SChopAndB*)
have *6*: $\vdash$ *f1*$\frown$ *sfin* (*init w2*) $\longrightarrow$ *sfin* (*init w2*)
by (*rule SChopSFinImpSFin*)
from *1 2 3 4 5 6* **show** *?thesis* **using** *SChopRep SChopRule* **by** *fastforce*
qed

lemma *TrueSChopMoreEqvMore*:
$\vdash$ #*True* $\frown$ *more* = *more*
by (*metis ChopAssoc TrueChopMoreEqvMore TrueEqvTrueSChopTrue inteq-reflection schop-d-def*)

lemma *SChopFmoreEqvFmore*:
$\vdash$ #*True*$\frown$*fmore* = *fmore*
by (*metis ChopAndFiniteDist TrueChopMoreEqvMore fmore-d-def inteq-reflection schop-d-def*)

lemma *MoreSChopLoop*:
assumes $\vdash$ *f* $\longrightarrow$ *more* $\frown$ *f*
shows $\vdash$ *finite* $\longrightarrow$ $\neg$ *f*
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *more* $\frown$ *f*
using *assms* **by** *auto*
hence *11*: $\vdash$ $\Diamond$ (*f*) $\longrightarrow$ $\Diamond$ (*more*$\frown$*f*)
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash$ $\Diamond$ (*more*$\frown$*f*) = #*True*$\frown$(*more*$\frown$*f*)
by (*simp add*: *DiamondSChopdef*)
have *13*: $\vdash$ #*True*$\frown$(*more*$\frown$*f*) = (#*True*$\frown$*more*)$\frown$*f*
by (*rule SChopAssoc*)
have *14*: $\vdash$ $\Diamond$ (*more*$\frown$*f*) = *more*$\frown$*f*
using *12 13* **by** (*metis TrueSChopMoreEqvMore inteq-reflection*)
have *2*: $\vdash$ *more* $\frown$ *f* = $\bigcirc$($\Diamond$ *f*)

using *MoreSChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash \ \Diamond\ (f) \longrightarrow \bigcirc(\Diamond\ f)$
using *11 14 2* **by** *fastforce*
hence *4*: $\vdash$ *finite* $\longrightarrow \neg\ (\Diamond\ f)$
using *NextLoop* **by** *blast*
have *5*: $\vdash \neg\ (\Diamond\ f) \longrightarrow \neg\ f$
using *NowImpDiamond* **by** *fastforce*
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *MoreSChopContra*:
assumes $\vdash \ f \wedge \neg\ g \longrightarrow (\ more \frown (f \wedge \neg\ g))$
shows $\vdash f \wedge finite \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg\ g \longrightarrow (\ more \frown (f \wedge \neg\ g))$ **using** *assms* **by** *auto*
hence *2*: $\vdash finite \longrightarrow \neg\ (f \wedge \neg\ g)$ **by** (*rule MoreSChopLoop*)
from *2* **show** *?thesis*
by (*simp add*: *Valid-def finite-defs infinite-defs sum.case-eq-if*)
qed

lemma *MoreSChopLoopFinite*:
assumes $\vdash \ f \wedge finite \longrightarrow \ more \frown f$
shows $\vdash finite \longrightarrow \neg\ f$
proof −
have *1*: $\vdash f \wedge finite \longrightarrow \ more \frown f$
using *assms* **by** *auto*
hence *11*: $\vdash \Diamond\ (f \wedge finite) \longrightarrow \Diamond\ (more \frown f)$
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash \Diamond\ (more \frown f) = \#True \frown (more \frown f)$
by (*simp add*: *DiamondSChopdef*)
have *13*: $\vdash \#True \frown (more \frown f) = (\#True \frown more) \frown f$
by (*rule SChopAssoc*)
have *14*: $\vdash \ \Diamond\ (more \frown f) = more \frown f$
using *12 13* **by** (*metis TrueSChopMoreEqvMore inteq-reflection*)
have *2*: $\vdash \ more \frown f = \bigcirc(\Diamond\ f)$
using *MoreSChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash \ \Diamond\ (f \wedge finite) \ \longrightarrow \bigcirc(\Diamond\ f)$
using *11 14 2* **by** *fastforce*
have *31*: $\vdash \Diamond\ (f \wedge finite) = ((\Diamond\ f) \wedge finite)$
by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SFinAndSChop SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection lift-and-com*)
have *32*: $\vdash (\Diamond\ f) \wedge finite \longrightarrow \bigcirc(\Diamond\ f)$
using *3 31* **by** *fastforce*
hence *4*: $\vdash finite \longrightarrow \neg\ (\Diamond\ f)$
by (*metis* (*no-types*, *lifting*) *DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09 finite-d-def int-simps*(*15*) *int-simps*(*32*) *inteq-reflection sometimes-d-def*)
have *5*: $\vdash \neg\ (\Diamond\ f) \longrightarrow \neg\ f$
by (*simp add*: *NowImpDiamond*)
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *MoreSChopContraFinite*:
 assumes $\vdash$ $(f \wedge \neg\ g) \wedge finite \longrightarrow (\ more \frown (f \wedge \neg\ g))$
 shows $\vdash f \wedge finite \longrightarrow g$
proof –
 have *1*: $\vdash (f \wedge \neg\ g) \wedge finite \longrightarrow (\ more \frown (f \wedge \neg\ g))$ **using** *assms* **by** *auto*
 hence *2*: $\vdash finite \longrightarrow \neg\ (f \wedge \neg\ g)$ **by** (*simp add*: *MoreSChopLoopFinite*)
 from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *SChopLoop*:
 assumes $\vdash$ $f \longrightarrow g \frown f$
 $\vdash g \longrightarrow\ fmore$
 shows $\vdash finite \longrightarrow \neg\ f$
proof –
have *1*: $\vdash f \longrightarrow g \frown f$ **using** *assms* **by** *auto*
have *2*: $\vdash g \longrightarrow\ more$ **using** *assms* **by** (*simp add*: *Prop12 fmore-d-def*)
hence *3*: $\vdash g \frown f \longrightarrow\ more \frown f$ **by** (*rule LeftSChopImpSChop*)
have *4*: $\vdash f \longrightarrow\ more \frown f$ **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **using** *MoreSChopLoop* **by** *auto*
qed

lemma *SChopLoopB*:
 assumes $\vdash$ $f \longrightarrow g \frown f$
 $\vdash g \longrightarrow\ more$
 shows $\vdash finite \longrightarrow \neg\ f$
proof –
have *1*: $\vdash f \longrightarrow g \frown f$ **using** *assms* **by** *auto*
have *2*: $\vdash g \longrightarrow\ more$ **using** *assms* **by** *auto*
hence *3*: $\vdash g \frown f \longrightarrow\ more \frown f$ **by** (*rule LeftSChopImpSChop*)
have *4*: $\vdash f \longrightarrow\ more \frown f$ **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **using** *MoreSChopLoop* **by** *blast*
qed

lemma *SChopContra*:
 assumes $\vdash$ $f \wedge \neg\ g \longrightarrow h \frown f \wedge \neg\ (h \frown g)$
 $\vdash h \longrightarrow\ fmore$
 shows $\vdash f \wedge finite \longrightarrow g$
proof –
have *1*: $\vdash f \wedge \neg\ g \longrightarrow h \frown f \wedge \neg\ (h \frown g)$ **using** *assms* **by** *auto*
have *2*: $\vdash h \longrightarrow\ more$ **using** *assms* **by** (*simp add*: *Prop12 fmore-d-def*)
have *3*: $\vdash h \frown f \wedge \neg\ (h \frown g) \longrightarrow h \frown (f \wedge \neg\ g)$ **by** (*rule SChopAndNotSChopImp*)
have *4*: $\vdash h \frown (f \wedge \neg\ g) \longrightarrow\ more \frown (f \wedge \neg\ g)$ **using** *2* **by** (*rule LeftSChopImpSChop*)
have *5*: $\vdash f \wedge \neg\ g \longrightarrow\ more \frown (f \wedge \neg\ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreSChopContra* **by** *auto*
qed

lemma *SChopContraB*:
 assumes $\vdash f \wedge \neg\ g \longrightarrow h \frown f \wedge \neg\ (h \frown g)$
 $\vdash h \longrightarrow\ more$

shows $\vdash f \wedge finite \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg\ g \longrightarrow h \frown f \wedge \neg (h \frown g)$ **using** *assms* **by** *auto*
have *2*: $\vdash h \longrightarrow\ more$ **using** *assms* **by** *auto*
have *3*: $\vdash h \frown f \wedge \neg (h \frown g) \longrightarrow h \frown (f \wedge \neg\ g)$ **by** (*rule SChopAndNotSChopImp*)
have *4*: $\vdash h \frown (f \wedge \neg\ g) \longrightarrow\ more \frown (f \wedge \neg\ g)$ **using** *2* **by** (*rule LeftSChopImpSChop*)
have *5*: $\vdash f \wedge \neg\ g \longrightarrow\ more \frown (f \wedge \neg\ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreSChopContra* **by** *blast*
qed

14.7 Properties of SChopstar and SChopplus

lemma *AndEmptySChopAndEmptyEqvAndEmpty*:
$\vdash (f \wedge empty) \frown (f \wedge empty) = (f \wedge empty)$
by (*auto simp add*: *Valid-def empty-defs schop-defs sum.case-eq-if*)
(*metis interval-st-intlen sum.collapse*(*1*))

lemma *SPowerImpFinite*:
$\vdash spower\ f\ n \longrightarrow finite$
proof
(*induct n*)
case *0*
then show *?case*
using *EmptyImpFinite* **by** *auto*
next
case (*Suc n*)
then show *?case*
by (*metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop10 SChopSFinExportA SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection spow-Suc*)
qed

lemma *SPowerCommute*:
$\vdash f \frown spower\ f\ n = spower\ f\ n \frown (f \wedge finite)$
proof
(*induct n*)
case *0*
then show *?case*
by (*metis ChopEmptySem EmptySChop intI inteq-reflection schop-d-def spow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis SChopAssoc inteq-reflection spow-Suc*)
qed

lemma *SChopInductL*:
assumes $\vdash g \vee f \frown h \longrightarrow h$
shows $\vdash (spower\ f\ n) \frown g \longrightarrow h$
using *assms*

by (*metis ChopInductFiniteL PowerSpowerdef Prop10 SPowerImpFinite inteq-reflection schop-d-def*)

lemma *SChopInductMoreL*:
assumes $\vdash g \vee (f \wedge more) \frown h \longrightarrow h$
shows $\vdash (spower\ f\ n) \frown g \longrightarrow h$
proof
(*induct n*)
case *0*
then show *?case* **using** *assms* **by** (*metis SChopInductL spow-0*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: $\vdash\ spower\ f\ (Suc\ n) \frown g\ = (f \frown spower\ f\ n) \frown g$
by *simp*
have *2*: $\vdash (f \frown spower\ f\ n) \frown g = f \frown ((spower\ f\ n) \frown g)$
by (*meson SChopAssoc Prop11*)
have *3*: $\vdash f \frown ((spower\ f\ n) \frown g) \longrightarrow f \frown h$
by (*simp add*: *RightSChopImpSChop Suc.hyps*)
have *4*: $\vdash f \frown h = ((\ f \wedge more) \frown\ h \vee ((f \wedge empty)) \frown h)$
using *neq0-conv*
by (*auto simp add*: *Valid-def finite-defs schop-defs more-defs empty-defs sum.case-eq-if*)
blast
have *5*: $\vdash (\ (f \wedge more)) \frown\ h \longrightarrow h$ **using** *assms* **by** *auto*
have *6*: $\vdash (\ (f \wedge empty)) \frown\ h \longrightarrow h$
by (*metis AndSChopB EmptySChop inteq-reflection*)
from *5 6 4 3 2 1* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *SChopImpFinite*:
assumes $\vdash f \longrightarrow finite$
shows $\vdash g \frown f \longrightarrow finite$
using *assms*
by (*metis DiamondImpDiamond FiniteChopEqvDiamond FiniteChopFiniteEqvFinite SChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *SChopInductR*:
assumes $\vdash g \vee h \frown f \longrightarrow h$
shows $\vdash g \frown (spower\ f\ n) \longrightarrow h$
proof
(*induct n*)
case *0*
then show *?case* **using** *assms*
by (*metis ChopEmpty MP Prop11 Prop12 int-simps*(*33*) *lift-imp-trans schop-d-def spow-0*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: $\vdash g \frown (spower\ f\ (Suc\ n)) = g \frown\ (f \frown (spower\ f\ n))$

by *simp*
have *2*: $\vdash g\frown (f\frown(spower\ f\ n)) = g\frown((spower\ f\ n)\frown (f \wedge finite))$
using *SPowerCommute* **by** (*simp add*: *SPowerCommute RightSChopEqvSChop*)
have *3*: $\vdash g\frown((spower\ f\ n)\frown (f \wedge finite)) =$
$(g\frown(spower\ f\ n))\frown (f \wedge finite)$
using *SChopAssoc* **by** *blast*
have *4*: $\vdash (g\frown(spower\ f\ n))\frown (f \wedge finite) \longrightarrow h\frown(f \wedge finite)$
using *LeftSChopImpSChop Suc.hyps* **by** *blast*
have *5*: $\vdash h\frown(f \wedge finite) \longrightarrow h$
using *assms*
by (*metis Prop03 Prop10 SChopAndA inteq-reflection lift-imp-trans*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *SChopExistSPower*:
$\vdash (g\frown(\exists\, n.\ spower\ f\ n)) = (\exists\ n.\ g\frown spower\ f\ n)$
using *SChopExist* **by** *fastforce*

lemma *ExistSChopSPower*:
$\vdash (\exists\, n.\ (spower\ f\ n)\frown g) = (\exists\, n.\ spower\ f\ n)\frown g$
using *ExistSChop* **by** *fastforce*

lemma *SPowerStarCommute*:
$\vdash f\frown(\exists\ n.\ spower\ f\ n) = (\exists\ n.\ spower\ f\ n)\frown(f \wedge finite)$
proof −
have *1*: $\vdash f\ \frown(\exists\ n.\ spower\ f\ n) =$
$(\exists\ n.\ f\ \frown\ spower\ f\ n)$
using *SChopExistSPower* **by** *blast*
have *2*: $\vdash (\exists\ n.\ f\ \frown\ spower\ f\ n) =$
$(\exists\ n.\ (spower\ f\ n)\frown(f \wedge finite)\ )$
using *SPowerCommute* **by** *fastforce*
have *3*: $\vdash (\exists\ n.\ (spower\ f\ n)\frown(f \wedge finite)) =$
$(\exists\ n.\ (spower\ f\ n))\frown(f \wedge finite)$
using *ExistSChopSPower* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SPowerSucAndEmptyEqvAndEmpty*:
$\vdash (spower\ (f \wedge empty)\ (Suc\ n)) = (f \wedge empty)$
proof
(*induct n*)
case *0*
then show *?case*
by (*metis PowerSpowerdef PowerSucAndEmptyEqvAndEmpty inteq-reflection*)
next
case (*Suc n*)
then show *?case*
by (*metis AndEmptySChopAndEmptyEqvAndEmpty inteq-reflection spow-Suc*)

qed

lemma *SPowerOr*:
$\vdash$ (*spower* ($f \vee g$) (*Suc* n)) = (($f \frown$ *spower* ($f \vee g$) n) $\vee$
($g \frown$ *spower* ($f \vee g$) n))
by (*simp add*: *FiniteOr OrSChopEqvRule*)

lemma *PowerEmptyOrMore*:
$\vdash$ (*spower* (($f \wedge$ *empty*) $\vee$ ($f \wedge$ *more*)) (*Suc* n)) =
(($f \wedge$ *empty*)$\frown$(*spower* (($f \wedge$ *empty*) $\vee$ ($f \wedge$ *more*)) n) $\vee$
($f \wedge$ *more*)$\frown$(*power* (($f \wedge$ *empty*) $\vee$ ($f \wedge$ *more*)) n))
using *SPowerOr*
by (*metis PowerSpowerdef inteq-reflection*)

lemma *SPSEqvEmptyOrSChopSPS*:
$\vdash$ *spowerstar* f = (*empty* $\vee$ $f \frown$*spowerstar* f)
by (*simp add*: *spowerstar-d-def spowersem*)

lemma *EmptyImpSCS*:
$\vdash$ *empty* $\longrightarrow$ *schopstar* f
proof –
have *1*: $\vdash$ *schopstar* f = (*empty* $\vee$ ($f \wedge$ *more*)$\frown$*schopstar* f)
by (*rule SChopstarEqv*)
have *2*: $\vdash$ *empty* $\longrightarrow$ *empty* $\vee$ ($f \wedge$ *more*)$\frown$*schopstar* f **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SCSEqvOrSChopSCS*:
$\vdash$ *schopstar* f = (*empty* $\vee$ ($f \frown$ *schopstar* f))
proof –
have *1*: $\vdash$ *schopstar* f = (*empty* $\vee$ ($f \wedge$ *more*)$\frown$*schopstar* f)
by (*rule SChopstarEqv*)
have *2*: $\vdash$ ($f \wedge$ *more*)$\frown$*schopstar* f $\longrightarrow$ $f \frown$*schopstar* f
by (*rule AndSChopA*)
have *3*: $\vdash$ *schopstar* f $\longrightarrow$ *empty* $\vee$ $f \frown$ *schopstar* f
using *1 2* **by** (*metis int-iffD1 Prop08*)
have *4*: $\vdash$ *empty* $\longrightarrow$ *schopstar* f **by** (*rule EmptyImpSCS*)
have *5*: $\vdash$ f $\longrightarrow$ *empty* $\vee$ ($f \wedge$ *more*) **by** (*auto simp*: *empty-d-def*)
have *6*: $\vdash$ $f \frown$ *schopstar* f $\longrightarrow$ *schopstar* f $\vee$ ($f \wedge$ *more*)$\frown$ *schopstar* f
using *5* **by** (*rule EmptyOrSChopImpRule*)
have *7*: $\vdash$ *schopstar* f $\longrightarrow$ *empty* $\vee$ ($f \wedge$ *more*)$\frown$*schopstar* f
using *1* **by** *fastforce*
have *8*: $\vdash$ $f \frown$ *schopstar* f $\longrightarrow$ *empty* $\vee$ ($f \wedge$ *more*)$\frown$ *schopstar* f
using *6 7* **by** *fastforce*
hence *9*: $\vdash$ $f \frown$ *schopstar* f $\longrightarrow$ *schopstar* f **using** *1* **by** *fastforce*
have *10*: $\vdash$ *empty* $\vee$ $f \frown$ *schopstar* f $\longrightarrow$ *schopstar* f **using** *9 4* **by** *fastforce*
from *3 10* **show** *?thesis* **by** *fastforce*
qed

lemma *SPowerSChopCommute*:
$\vdash ((f \wedge more))\frown spower\ (f \wedge more)\ n = spower\ (f \wedge more)\ n\frown((f \wedge more) \wedge finite)$
using *SPowerCommute* **by** *auto*

lemma *SChopExist*:
$\vdash (g\frown(\exists\, n.\ spower\ (f \wedge more)\ n)) = (\exists\ n.\ g\frown\ spower\ (f \wedge more)\ n)$
using *SChopExistSPower* **by** *auto*

lemma *ExistSChop*:
$\vdash (\exists\, n.\ (spower\ (f \wedge more)\ n)\frown g) = (\exists\, n.\ spower\ (f \wedge more)\ n)\frown g$
using *ExistSChopSPower* **by** *auto*

lemma *SPowerstarInductL*:
assumes $\vdash g \vee f\frown h \longrightarrow h$
shows $\vdash (spowerstar\ f)\frown g \longrightarrow h$
proof −
have *1*: $\vdash (spowerstar\ f)\frown g = ((\exists\ n.\ spower\ f\ n))\frown g$
by (*simp add*: *spowerstar-d-def LeftChopEqvChop*)
have *2*: $\vdash (\exists\ n.\ spower\ f\ n)\frown g =$
$(\exists\ n.\ (spower\ f\ n)\frown g)$
using *ExistSChopSPower* **by** *fastforce*
have *3*: $\bigwedge n. \vdash (spower\ f\ n)\frown g \longrightarrow h$
using *SChopInductL assms* **by** *blast*
have *4*: $\vdash (\exists\ n.\ (spower\ f\ n)\frown g) \longrightarrow h$
using *3* **by** (*simp add*: *Valid-def*) *fastforce*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *SChopstarInductL*:
assumes $\vdash g \vee f\frown h \longrightarrow h$
shows $\vdash (schopstar\ f)\frown\ g \longrightarrow h$
proof −
have *1*: $\vdash (schopstar\ f)\frown g = (\exists\ n.\ spower\ (f \wedge more)\ n)\frown g$
by (*simp add*: *schopstar-d-def spowerstar-d-def LeftChopEqvChop*)
have *2*: $\vdash (\exists\ n.\ spower\ (f \wedge more)\ n)\frown g =$
$(\exists\ n.\ (spower\ (f \wedge more)\ n)\frown g)$
using *ExistSChopSPower* **by** *fastforce*
have *21*: $\vdash g \vee (f \wedge more)\ \frown\ h \longrightarrow h$
using *AndSChopA assms* **by** *fastforce*
have *3*: $\bigwedge n. \vdash (spower\ (f \wedge more)\ n)\frown g \longrightarrow h$
using *21 SChopInductL*[*of g LIFT*$(f \wedge more)$ *h*] **by** *auto*
have *4*: $\vdash (\exists\ n.\ (spower\ (f \wedge more)\ n)\frown g) \longrightarrow h$
using *3* **by** (*simp add*: *Valid-def*) *fastforce*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *SChopstarInductMoreL*:
assumes $\vdash g \vee (f \wedge more)\frown h \longrightarrow h$

shows $\vdash$ (*schopstar f*)$\frown$*g* $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ (*schopstar f*)$\frown$*g* = ($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*)$\frown$*g*
by (*simp add*: *schopstar-d-def spowerstar-d-def LeftChopEqvChop*)
have *2*: $\vdash$ ($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*)$\frown$*g* =
($\exists$ *n*. (*spower* (*f* $\wedge$ *more*) *n*)$\frown$*g*)
using *ExistSChopSPower* **by** *fastforce*
have *3*: $\bigwedge$ *n*. $\vdash$ (*spower* (*f* $\wedge$ *more*) *n*)$\frown$*g* $\longrightarrow$ *h*
using *SChopInductL assms* **by** (*metis*)
have *4*: $\vdash$ ($\exists$ *n*. (*spower* (*f* $\wedge$ *more*) *n*)$\frown$*g*) $\longrightarrow$ *h*
using *3* **by** *fastforce*
from *1 2 4* **show** *?thesis*
by (*metis inteq-reflection*)
qed

lemma *SPowerstarInductR*:
assumes $\vdash$ *g* $\vee$ *h*$\frown$*f* $\longrightarrow$ *h*
shows $\vdash$ *g*$\frown$(*spowerstar f*) $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ *g*$\frown$(*spowerstar f*) = *g*$\frown$(($\exists$ *n*. *spower f n*))
by (*simp add*: *spowerstar-d-def*)
have *2*: $\vdash$ (*g*$\frown$($\exists$ *n*. *spower f n*)) = ($\exists$ *n*. *g*$\frown$(*spower f n*))
using *SChopExistSPower* **by** *blast*
have *3*: $\bigwedge$ *n*. $\vdash$ *g*$\frown$(*spower f n*) $\longrightarrow$ *h*
using *SChopInductR assms* **by** *blast*
have *4*: $\vdash$ ($\exists$ *n*. *g*$\frown$(*spower f n*)) $\longrightarrow$ *h*
using *3* **by** (*simp add*: *Valid-def*) *fastforce*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *SChopstarInductR*:
assumes $\vdash$ *g* $\vee$ *h*$\frown$*f* $\longrightarrow$ *h*
shows $\vdash$ *g*$\frown$(*schopstar f*) $\longrightarrow$ *h*
proof −
have *1*: $\vdash$ *g*$\frown$(*schopstar f*) =
g$\frown$(($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*))
by (*simp add*: *schopstar-d-def spowerstar-d-def*)
have *2*: $\vdash$ (*g*$\frown$($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*)) =
(($\exists$ *n*. *g*$\frown$*spower* (*f* $\wedge$ *more*) *n*))
using *SChopExistSPower* **by** *fastforce*
have *21*: $\vdash$ *h* $\frown$ (*f* $\wedge$ *more*) $\longrightarrow$ *h*
using *assms*
by (*metis Prop03 Prop10 SChopAndA inteq-reflection lift-imp-trans*)
have *22*: $\vdash$ *g* $\longrightarrow$ *h*
using *assms* **by** *auto*
have *23*: $\vdash$ *g* $\vee$ *h* $\frown$ (*f* $\wedge$ *more*) $\longrightarrow$ *h*
using *21 22 Prop02* **by** *blast*
have *3*: $\bigwedge$ *n*. $\vdash$ *g*$\frown$(*spower* (*f* $\wedge$ *more*) *n*) $\longrightarrow$ *h*

using *23 SChopInductR*[*of g h LIFT*(*f* ∧ *more*)] **by** *auto*
have *4*: ⊢ (∃ *n*. *g*⌢(*spower* (*f* ∧ *more*) *n*)) ⟶ *h*
using *3* **by** (*simp add*: *Valid-def*) *fastforce*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *SChopstarInductMoreR*:
assumes ⊢ *g* ∨ *h*⌢(*f* ∧ *more*) ⟶ *h*
shows ⊢ *g*⌢(*schopstar f*) ⟶ *h*
proof −
have *1*: ⊢ *g*⌢(*schopstar f*) = *g*⌢((∃ *n*. *spower* (*f* ∧ *more*) *n*))
by (*simp add*: *schopstar-d-def spowerstar-d-def*)
have *2*: ⊢ (*g*⌢(∃ *n*. *spower* (*f* ∧ *more*) *n*)) =
((∃ *n*. *g*⌢*spower* (*f* ∧ *more*) *n*))
using *SChopExistSPower* **by** *fastforce*
have *3*: ⋀ *n*. ⊢ *g*⌢(*spower* (*f* ∧ *more*) *n*) ⟶ *h*
using *SChopInductR assms* **by** (*metis*)
have *4*: ⊢ (∃ *n*. *g*⌢(*spower* (*f* ∧ *more*) *n*)) ⟶ *h*
using *3* **by** (*simp add*: *Valid-def*) *fastforce*
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *SChopstarImpSPowerstar*:
⊢ *schopstar f* ⟶ *spowerstar f*
by (*metis* (*mono-tags*, *lifting*) *PowerSpowerdef SChopstarFPowerstardef fpowerstar-d-def intI intensional-rews*(*3*) *intensional-rews*(*6*) *inteq-reflection spowerstar-d-def*)

lemma *SPowerstarImpSChopstar*:
⊢ *spowerstar f* ⟶ *schopstar f*
by (*metis* (*mono-tags*, *lifting*) *PowerSpowerdef SChopstarFPowerstardef fpowerstar-d-def intI intensional-rews*(*3*) *intensional-rews*(*6*) *inteq-reflection spowerstar-d-def*)

lemma *SChopstarEqvSPowerstar*:
⊢ *schopstar f* = *spowerstar f*
using *SChopstarImpSPowerstar SPowerstarImpSChopstar* **by** *fastforce*

lemma *SCSAndMoreEqvAndMoreSChop*:
⊢ (*schopstar f* ∧ *more*) = (*f* ∧ *more*)⌢ *schopstar f*
proof −
have *1*: ⊢ (*empty* ∨ (*f* ∧ *more*)⌢ *schopstar f*) ∧ *more* ⟶ (*f* ∧ *more*)⌢ *schopstar f*
by (*auto simp*: *empty-d-def*)
have *2*: ⊢ *schopstar f* = (*empty* ∨ (*f* ∧ *more*)⌢ *schopstar f*)
by (*rule SChopstarEqv*)
have *3*: ⊢ *schopstar f* ∧ *more* ⟶ (*f* ∧ *more*)⌢ *schopstar f*
using *1 2* **by** *fastforce*
have *4*: ⊢ (*f* ∧ *more*)⌢ *schopstar f* ⟶ *schopstar f*
using *2* **by** *fastforce*
have *5*: ⊢ (*f* ∧ *more*) ⟶ *more*
by *auto*

hence *6*: $\vdash$ (*f* $\wedge$ *more*)$\frown$ *schopstar f* $\longrightarrow$ *more*
by (*rule LeftSChopImpMoreRule*)
have *7*: $\vdash$ (*f* $\wedge$ *more*)$\frown$ *schopstar f* $\longrightarrow$ *schopstar f* $\wedge$ *more*
using *4 6* **by** *fastforce*
from *3 7* **show** *?thesis* **by** *fastforce*
qed

lemma *SPowerAndMoreAndFinite*:
$\vdash$ ((*spower* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*) = (*spower* (*f* $\wedge$ *more*) *n*)
by (*meson Prop10 Prop11 SPowerImpFinite*)

lemma *SCSAndFinite*:
$\vdash$ (*schopstar f* $\wedge$ *finite*) = *schopstar f*
proof –
have *1*: $\vdash$ (*schopstar f* $\wedge$ *finite*) = (($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*)
by (*simp add*: *schopstar-d-def spowerstar-d-def intI*)
have *2*: $\vdash$ (($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*) $\wedge$ *finite*) =
($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n* $\wedge$ *finite*)
by (*simp add*: *Valid-def*)
have *3*: $\vdash$ ($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n* $\wedge$ *finite*) =
($\exists$ *n*. (*spower* (*f* $\wedge$ *more*) *n*))
using *SPowerAndMoreAndFinite* **by** *fastforce*
have *4*: $\vdash$ ($\exists$ *n*. (*spower* (*f* $\wedge$ *more*) *n*)) = *schopstar f*
by (*simp add*: *schopstar-d-def spowerstar-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SPowerchopAndFmore*:
$\vdash$ ((*spower* (*f* $\wedge$ *more*) (*Suc n*)) $\wedge$ *fmore*) = (*spower* (*f* $\wedge$ *more*) (*Suc n*))
by (*metis* (*no-types*, *lifting*) *FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite LeftSChopImpMoreRule Prop10 Prop12 SPowerImpFinite int-iffD1 inteq-reflection lift-and-com spower-d.simps*(*2*))

lemma *ExistSPowerAndMoreExpand*:
$\vdash$ ($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*) = (*empty* $\vee$ ($\exists$ *n*. (*spower* (*f* $\wedge$ *more*) (*Suc n*))))
using *spowersem1*[*of LIFT*(*f* $\wedge$ *more*)] **by** *auto*

lemma *SCSAndMoreEqvAndFMoreSChop*:
$\vdash$ (*schopstar f* $\wedge$ *fmore*) = (*f* $\wedge$ *more*)$\frown$ *schopstar f*
proof –
have *1*: $\vdash$ (*schopstar f* $\wedge$ *fmore*) = (($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*) $\wedge$ *fmore*)
by (*simp add*: *schopstar-d-def spowerstar-d-def intI*)
have *2*: $\vdash$ (($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n*) $\wedge$ *fmore*) =
($\exists$ *n*. *spower* (*f* $\wedge$ *more*) *n* $\wedge$ *fmore*)
by (*simp add*: *Valid-def*)

have *3*: ⊢ (∃ *n*. *spower* (*f* ∧ *more*) *n* ∧ *fmore*) =
((*spower* (*f* ∧ *more*) *0* ∨ (∃ *n*. (*spower* (*f* ∧ *more*) (*Suc n*)))) ∧ *fmore*)
using *ExistSPowerAndMoreExpand* **by** *fastforce*
have *4*: ⊢ ((*spower* (*f* ∧ *more*) *0* ∨ (∃ *n*. (*spower* (*f* ∧ *more*) (*Suc n*)))) ∧ *fmore*) =
(((*spower* (*f* ∧ *more*) *0* ∧ *fmore*) ∨ ((∃ *n*. (*spower* (*f* ∧ *more*) (*Suc n*))) ∧ *fmore*)))
by *auto*
have *5*: ⊢ (((*spower* (*f* ∧ *more*) *0* ∧ *fmore*) ∨ ((∃ *n*. (*spower* (*f* ∧ *more*) (*Suc n*))) ∧ *fmore*))) =
((∃ *n*. (*spower* (*f* ∧ *more*) (*Suc n*))) ∧ *fmore*)
using *NotFmoreAndEmpty* **by** *fastforce*
have *6*: ⊢ ((∃ *n*. (*spower* (*f* ∧ *more*) (*Suc n*))) ∧ *fmore*) =
(∃ *n*. (*spower* (*f* ∧ *more*) (*Suc n*)))
using *SPowerchopAndFmore* **by** *fastforce*
have *7*: ⊢ (∃ *n*. (*spower* (*f* ∧ *more*) (*Suc n*))) =
(∃ *n*. ((*f* ∧ *more*)⁀(*spower* (*f* ∧ *more*) *n*)))
by (*simp*)
have *8*: ⊢ (∃ *n*. ((*f* ∧ *more*)⁀(*spower* (*f* ∧ *more*) *n*))) =
(*f* ∧ *more*)⁀(∃ *n*. (*spower* (*f* ∧ *more*) *n*))
using *SChopExist* **by** *fastforce*
have *9*: ⊢ (∃ *n*. (*spower* (*f* ∧ *more*) *n*)) =
schopstar f
by (*simp add*: *schopstar-d-def spowerstar-d-def intI*)
hence *10*: ⊢ (*f* ∧ *more*)⁀(∃ *n*. (*spower* (*f* ∧ *more*) *n*)) =
(*f* ∧ *more*)⁀*schopstar f*
by (*simp add*: *RightSChopEqvSChop*)
from *1 2 3 4 5 6 7 8 10* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *SCSAndMoreImpSChopSCS*:
⊢ *schopstar f* ∧ *more* ⟶ *f*⁀ *schopstar f*
proof −
have *1*: ⊢ (*schopstar f* ∧ *more*) = (*f* ∧ *more*)⁀ *schopstar f* **by** (*rule SCSAndMoreEqvAndMoreSChop*)
have *2*: ⊢ (*f* ∧ *more*)⁀ *schopstar f* ⟶ *f*⁀ *schopstar f* **by** (*rule AndSChopA*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SCSMoreNotImpSChopSCSAndMore*:
⊢ *schopstar f* ∧ *more* ∧ ¬ *f* ⟶ (*f* ∧ *more*)⁀ (*schopstar f* ∧ *more*)
proof −
have *1*: ⊢ (*schopstar f* ∧ *more*) = (*f* ∧ *more*)⁀ *schopstar f*
by (*rule SCSAndMoreEqvAndMoreSChop*)
have *2*: ⊢ *empty* ∨ *more*
by (*auto simp*: *empty-d-def*)
hence *3*: ⊢ *schopstar f* ⟶ *empty* ∨ (*schopstar f* ∧ *more*)
by *auto*
hence *4*: ⊢ (*f* ∧ *more*)⁀ *schopstar f* ⟶ (*f* ∧ *more*) ∨ ((*f* ∧ *more*)⁀ (*schopstar f* ∧ *more*))
using *SChopEmptyOrImpRule*
by (*metis 1 AndMoreAndFiniteEqvAndFmore SCSAndMoreEqvAndFMoreSChop inteq-reflection*)
hence *5*: ⊢ (*f* ∧ *more*)⁀ *schopstar f* ∧ ¬(*f* ∧ *more*) ⟶ ((*f* ∧ *more*)⁀ (*schopstar f* ∧ *more*))
by *fastforce*
have *6*: ⊢ (*f* ∧ *more*)⁀ *schopstar f* = ((*f* ∧ *more*)⁀ *schopstar f* ∧ *more*) **using** *1*

by *auto*
have *7*: ⊢ ((*f* ∧ *more*)⌢ *schopstar f* ∧ ¬(*f* ∧ *more*)) =
((*f* ∧ *more*)⌢ *schopstar f* ∧ *more* ∧ ¬(*f* ∧ *more*))
using *6* **by** *auto*
have *8*: ⊢ (*f* ∧ *more*)⌢ *schopstar f* ∧ *more* ∧ ¬ *f* ⟶ (*f* ∧ *more*)⌢ (*schopstar f* ∧ *more*)
using *5 7* **by** *auto*
have *9*: ⊢ (*schopstar f* ∧ *more* ∧ ¬ *f*) = ((*schopstar f* ∧ *more*) ∧ (*more* ∧ ¬ *f*))
by *auto*
have *10*: ⊢ ((*schopstar f* ∧ *more*) ∧ (*more* ∧ ¬ *f*)) =
((*f* ∧ *more*)⌢ *schopstar f* ∧ (*more* ∧ ¬ *f*))
using *1* **by** *fastforce*
from *1 8 9 10* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopplusCommuteImpA*:
⊢ *schopstar f*⌢(*f* ∧ *finite*) ⟶ *f*⌢*schopstar f*
by (*metis SChopstarEqvSPowerstar SPowerStarCommute int-iffD1 inteq-reflection spowerstar-d-def*)

lemma *SChopplusCommuteImpB*:
⊢ *f*⌢*schopstar f* ⟶ *schopstar f*⌢(*f* ∧ *finite*)
by (*metis SChopstarEqvSPowerstar SPowerStarCommute int-iffD2 inteq-reflection spowerstar-d-def*)

lemma *SChopplusCommute*:
⊢ *f*⌢*schopstar f* = *schopstar f*⌢(*f* ∧ *finite*)
using *SChopplusCommuteImpA SChopplusCommuteImpB* **by** *fastforce*

lemma *SCSEqvOrChopSCSB*:
⊢ *schopstar f* = (*empty* ∨ (*schopstar f*⌢ (*f* ∧ *finite*)))
by (*meson SCSEqvOrSChopSCS SChopplusCommute Prop06*)

lemma *SCSAndMoreImpSCSSChop*:
⊢ *schopstar f* ∧ *more* ⟶ *schopstar f*⌢ (*f* ∧ *finite*)
using *SCSAndMoreEqvAndMoreSChop SChopplusCommute SCSAndMoreImpSChopSCS* **by** *fastforce*

lemma *SPowerSChopSPower*:
⊢ (*spower* (*f* ∧ *more*) *n*)⌢ (*spower* (*f* ∧ *more*) *k*) = (*spower* (*f* ∧ *more*) (*n*+*k*))
proof
(*induct n arbitrary*: *k*)
case *0*
then show *?case* **by** (*metis EmptySChop add.left-neutral spow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis PowerChopPower PowerSpowerdef SPowerAndMoreAndFinite inteq-reflection schop-d-def*)
qed

lemma *SCSSChopSCS*:
⊢ *schopstar f* ⌢ *schopstar f* = *schopstar f*
proof −

have *1*: $\vdash$ *schopstar f* $\frown$ *schopstar f* $\longrightarrow$ *schopstar f*
by (*metis Prop02 Prop03 SChopstarEqv SChopstarEqvSPowerstar SChopstarInductMoreL SPowerstarImpSChopstar inteq-reflection*)
have *2*: $\vdash$ *schopstar f* $\longrightarrow$ *schopstar f* $\frown$ *schopstar f*
by (*metis* (*no-types*, *lifting*) *AndSFinEqvSChopAndEmpty DiamondEmptyEqvFinite EmptyImpSCS FiniteAndEmptyEqvEmpty SCSAndFinite SChopImpSChop SChopstarEqvSPowerstar SFinEqvTrueSChopAndEmpty SPowerstarImpSChopstar TrueSChopEqvDiamond inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotSCSImpMore*:
$\vdash$ $\neg$ (*schopstar f*) $\longrightarrow$ *more*
proof –
have *1*: $\vdash$ *empty* $\longrightarrow$ *schopstar f* **using** *EmptyImpSCS* **by** *blast*
hence *2*: $\vdash$ $\neg$ *empty* $\vee$ *schopstar f* **by** *fastforce*
from *2* **show** *?thesis* **using** *1 NotEmptyEqvMore* **by** *fastforce*
qed

lemma *NotSCSAndInf*:
$\vdash$ $\neg$(*schopstar f* $\wedge$ *inf*)
using *InfEqvNotFinite SCSAndFinite* **by** *fastforce*

lemma *SCSSChopSCSImpSCS*:
$\vdash$ (*schopstar f* $\frown$ *schopstar f*) $\longrightarrow$ *schopstar f*
by (*simp add*: *SCSSChopSCS int-iffD1*)

lemma *ImpSChopPlus*:
$\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *f*$\frown$*schopstar f*
proof –
have *1*: $\vdash$ *schopstar f* = (*empty* $\vee$ *f*$\frown$ *schopstar f*) **by** (*rule SCSEqvOrSChopSCS*)
hence *2*: $\vdash$ *f*$\frown$*schopstar f* = (*f*$\frown$*empty* $\vee$ *f*$\frown$ (*f*$\frown$*schopstar f*)) **using** *SChopOrEqvRule* **by** *blast*
have *3*: $\vdash$ *finite* $\longrightarrow$ *f*$\frown$*empty* = *f* **using** *SChopEmpty* **by** *blast*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *ImpSCS*:
$\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *schopstar f*
proof –
have *1*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *f*$\frown$ *schopstar f* **by** (*rule ImpSChopPlus*)
hence *2*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *empty* $\vee$ *f*$\frown$*schopstar f* **by** *auto*
from *2* **show** *?thesis* **using** *SCSEqvOrSChopSCS* **by** *fastforce*
qed

lemma *SCSSChopImpSCS*:
$\vdash$ *schopstar f*$\frown$ (*f* $\wedge$ *finite*) $\longrightarrow$ *schopstar f*
proof –
have *1*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *schopstar f* **by** (*rule ImpSCS*)
hence *2*: $\vdash$ *schopstar f*$\frown$ (*f* $\wedge$ *finite*) $\longrightarrow$ *schopstar f*$\frown$ *schopstar f* **by** (*rule RightSChopImpSChop*)
hence *3*: $\vdash$ *schopstar f*$\frown$ (*f* $\wedge$ *finite*) $\longrightarrow$ *schopstar f*$\frown$ *schopstar f* **by** *auto*

have *4*: $\vdash$ *schopstar f* $\frown$ *schopstar f* $\longrightarrow$ *schopstar f* **by** (*rule SCSSChopSCSImpSCS*)
from *2 3 4* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *SChopPlusImpSCS*:
$\vdash$ *f* $\frown$*schopstar f* $\longrightarrow$ *schopstar f*
proof −
have *1*: $\vdash$ *f* $\frown$*schopstar f* $\longrightarrow$ *empty* $\vee$ *f* $\frown$*schopstar f* **by** *auto*
from *1* **show** *?thesis* **using** *SCSEqvOrSChopSCS* **by** *fastforce*
qed

lemma *SCSSChopEqvOrSChopPlusSChop*:
$\vdash$ *schopstar f* $\frown$ *g* = (*g* $\vee$ (*f* $\frown$*schopstar f*) $\frown$ *g*)
proof −
have *1*: $\vdash$ *schopstar f* = (*empty* $\vee$ *f* $\frown$*schopstar f*) **by** (*rule SCSEqvOrSChopSCS*)
from *1* **show** *?thesis* **using** *EmptyOrSChopEqvRule* **by** *blast*
qed

lemma *SCSElim*:
assumes $\vdash$ *empty* $\longrightarrow$ *g*
$\vdash$ (*f* $\wedge$ *more*) $\frown$ *g* $\longrightarrow$ *g*
shows $\vdash$ *schopstar f* $\longrightarrow$ *g*
proof −
have *1*: $\vdash$ *empty* $\vee$ (*f* $\wedge$ *more*)$\frown$*g* $\longrightarrow$ *g*
using *assms* **using** *Prop02* **by** *blast*
have *2*: $\vdash$ (*schopstar f*)$\frown$*empty* $\longrightarrow$ *g*
using *SChopstarInductMoreL 1* **by** *blast*
from *2* **show** *?thesis*
by (*metis AndSFinEqvSChopAndEmpty DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SCSAndFinite SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *SChopstarImp*:
assumes $\vdash$ *f* $\frown$(*schopstar g*) $\vee$ *empty* $\longrightarrow$ (*schopstar g*)
shows $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar g*)
using *assms SChopstarInductL*[*of LIFT*(*empty*) *f LIFT*(*schopstar g*)]
SChopEmpty[*of LIFT*(*schopstar f*)]
by (*metis ChopEmpty SCSAndFinite int-eq int-simps*(*33*) *lift-and-com schop-d-def*)

lemma *SCSSCSImpSCS*:
$\vdash$ *schopstar* (*schopstar f*) $\longrightarrow$ *schopstar f*
proof −
have *1*: $\vdash$ ((*schopstar f*)$\frown$(*schopstar f*)) $\vee$ *empty* $\longrightarrow$ (*schopstar f*)
by (*meson SCSSChopSCSImpSCS EmptyImpSCS Prop02*)
from *1* **show** *?thesis* **using** *SChopstarImp* **by** *blast*
qed

lemma *SCSImpSCSSCS*:

$\vdash$ *schopstar f* $\longrightarrow$ *schopstar* (*schopstar f*)
using *ImpSCS* **by** (*metis SCSAndFinite inteq-reflection*)

lemma *SCSSCSEqvSCS*:
$\vdash$ *schopstar* (*schopstar f*) = *schopstar f*
by (*simp add*: *SCSSCSImpSCS SCSImpSCSSCS int-iffI*)

lemma *RightEmptyOrSChopEqv*:
$\vdash$ $g\frown$(*empty* $\vee$ *f*) = ((*g* $\wedge$ *finite*) $\vee$ ($g\frown$ *f*))
proof −
have *1*: $\vdash$ $g\frown$(*empty* $\vee$ *f*) = ($g\frown$*empty* $\vee$ $g\frown f$) **by** (*rule SChopOrEqv*)
have *2*: $\vdash$ *finite* $\longrightarrow$ $g\frown$*empty* = *g* **by** (*rule SChopEmpty*)
from *1 2* **show** *?thesis* **by** (*simp add*: *RightEmptyOrChopEqv schop-d-def*)
qed

lemma *RightEmptyOrSChopEqvRule*:
assumes $\vdash$ *f* = (*empty* $\vee$ *f1*)
shows $\vdash$ $g\frown f$ = ((*g* $\wedge$ *finite*) $\vee$ ($g\frown$*f1*))
proof −
have *1*: $\vdash$ *f* = (*empty* $\vee$ *f1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ $g\frown f$ = $g\frown$(*empty* $\vee$ *f1*) **by** (*rule RightSChopEqvSChop*)
have *3*: $\vdash$ $g\frown$(*empty* $\vee$ *f1*) = ((*g* $\wedge$ *finite*) $\vee$ ($g\frown$*f1*)) **by** (*rule RightEmptyOrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopPlusEqvOrSChopSChopPlus*:
$\vdash$ ($f\frown$*schopstar f*) = ((*f* $\wedge$ *finite*) $\vee$ $f\frown$ ($f\frown$*schopstar f*))
proof −
have *1*: $\vdash$ *schopstar f* = (*empty* $\vee$ $f\frown$*schopstar f*) **by** (*rule SCSEqvOrSChopSCS*)
from *1* **show** *?thesis* **by** (*rule RightEmptyOrSChopEqvRule*)
qed

lemma *SCSAndEmptyEqvEmpty*:
$\vdash$ (*schopstar f* $\wedge$ *empty*) = *empty*
using *EmptyImpSCS* **by** *fastforce*

lemma *NotAndMoreSChopAndEmpty*:
$\vdash$ $\neg$(((*f* $\wedge$ *more*)$\frown$*g*) $\wedge$ *empty*)
by (*metis LeftSChopImpMoreRule Prop05 Prop12 Prop13 empty-d-def int-iffD1 int-simps*(*15*)
inteq-reflection lift-and-com)

lemma *NotSChopAndMoreAndEmpty*:
$\vdash$ $\neg$(($f\frown$(*g* $\wedge$ *more*)) $\wedge$ *empty*)
by (*simp add*: *NotChopAndMoreAndEmpty schop-d-def*)

lemma *SChopSCSAndEmptyEqvAndEmpty*:
$\vdash$ (($f\frown$*schopstar f*) $\wedge$ *empty*) = (*f* $\wedge$ *empty*)
proof −
have *1*: $\vdash$ (($f\frown$*schopstar f*) $\wedge$ *empty*) = (*f* $\wedge$ *empty*)$\frown$(*schopstar f* $\wedge$ *empty*)
using *SChopAndEmptyEqvEmptySChopEmpty* **by** *blast*

have *2*: $\vdash$ $(f \wedge\ empty) \frown (schopstar\ f\ \wedge\ empty) = (f \wedge\ empty) \frown empty$
using *SCSAndEmptyEqvEmpty* **using** *RightSChopEqvSChop* **by** *blast*
have *3*: $\vdash$ $(f\ \wedge\ empty) \frown empty = (f\ \wedge\ empty)$
by (*metis AndChopA AndEmptySChopAndEmptyEqvAndEmpty ChopEmpty Prop11 SChopAndB inteq-reflection schop-d-def*)
show *?thesis*
using *2 3 SChopAndEmptyEqvEmptySChopEmpty* **by** *fastforce*
qed

lemma *AndMoreSChopAndMoreEqvAndMoreSChop*:
$\vdash ((f\ \wedge\ more) \frown g\ \wedge\ more) = (f\ \wedge\ more) \frown g$
by (*meson AndSChopB MoreSChopImpMore Prop10 Prop11 lift-imp-trans*)

lemma *AndFmoreOrAndEmptyEqvAndFinite*:
$\vdash ((f\ \wedge\ more \wedge\ finite) \vee (f\ \wedge\ empty)) = (f\ \wedge\ finite)$
by (*auto simp add*: *Valid-def empty-defs more-defs finite-defs sum.case-eq-if*)

lemma *SChopPlusEqv*:
$\vdash (f \frown schopstar\ f) = ((f\ \wedge\ finite) \vee\ (f\ \wedge\ more) \frown (f \frown schopstar\ f))$
proof −
have *1*: $\vdash$ $schopstar\ f = (empty\ \vee\ (f\ \wedge\ more) \frown schopstar\ f)$
by (*rule SChopstarEqv*)
have *2*: $\vdash$ $schopstar\ f = (empty\ \vee f \frown schopstar\ f)$
by (*rule SCSEqvOrSChopSCS*)
hence *3*: $\vdash$ $(empty\ \vee\ f \frown schopstar\ f) = (empty\ \vee\ (f\ \wedge\ more) \frown schopstar\ f)$
using *1 2* **by** *fastforce*
have *4*: $\vdash$ $(f\ \wedge\ more) \frown schopstar\ f = (f\ \wedge\ more) \frown (empty\ \vee\ f \frown schopstar\ f)$
using *2* **using** *RightSChopEqvSChop* **by** *blast*
hence *5*: $\vdash$ $empty\ \vee\ f \frown schopstar\ f = empty\ \vee\ (f\ \wedge\ more) \frown (empty\ \vee\ f \frown schopstar\ f)$
using *3 4* **by** *fastforce*
have *6*: $\vdash$ $(f\ \wedge\ more) \frown (\ empty\ \vee\ f \frown schopstar\ f\ ) =$
$((f\ \wedge\ more) \frown\ empty\ \vee\ (f\ \wedge\ more) \frown (f \frown schopstar\ f))$
using *SChopOrEqv* **by** *blast*
have *7*: $\vdash$ $(f\ \wedge\ more) \frown\ empty\ = (f\ \wedge\ more \wedge finite)$
by (*metis AndMoreAndFiniteEqvAndFmore ChopEmpty fmore-d-def inteq-reflection schop-d-def*)
have *8*: $\vdash$ $(empty\ \vee\ f \frown schopstar\ f) =$
$(empty\ \vee\ (f\ \wedge\ more \wedge finite) \vee\ (f\ \wedge\ more) \frown (f \frown schopstar\ f))$
using *5 6 7* **by** (*metis 2 3 inteq-reflection*)
have *9*: $\vdash$ $((empty\ \vee\ f \frown schopstar\ f) \wedge more) = (f \frown schopstar\ f \wedge more)$
by (*auto simp*: *empty-d-def*)
have *10*: $\vdash$ $((empty\ \vee\ (f\ \wedge\ more \wedge finite) \vee\ (f\ \wedge\ more) \frown (f \frown schopstar\ f)) \wedge more) =$
$(((f\ \wedge\ more \wedge finite) \vee\ (f\ \wedge\ more) \frown (f \frown schopstar\ f)) \wedge more)$
by (*auto simp*: *empty-d-def*)
have *11*: $\vdash$ $(((f\ \wedge\ more \wedge finite) \vee\ (f\ \wedge\ more) \frown (f \frown schopstar\ f)) \wedge more) =$
$((f\ \wedge\ more \wedge finite) \vee\ (f\ \wedge\ more) \frown (f \frown schopstar\ f))$
using *10 6 7 int-eq*
using *AndMoreSChopAndMoreEqvAndMoreSChop* **by** *fastforce*
have *12*: $\vdash$ $(f \frown schopstar\ f \wedge more) = ((f\ \wedge\ more \wedge finite) \vee\ (f\ \wedge\ more) \frown (f \frown schopstar\ f))$
using *8 9 10 11* **by** *fastforce*

have *13*: ⊢ ($f \frown$*schopstar f* ∧ *empty*) = (*f* ∧ *empty*)
  by (*rule SChopSCSAndEmptyEqvAndEmpty*)
have *14*: ⊢ ((*f* ∧ *more* ∧ *finite*) ∨ (*f* ∧ *more*)⌢ (*f*⌢*schopstar f*) ∨ (*f* ∧ *empty*)) =
    ((*f* ∧ *finite*) ∨ (*f* ∧ *more*)⌢(*f*⌢*schopstar f*))
  using *AndFmoreOrAndEmptyEqvAndFinite*[*of f*]
  by *auto*
have *15*: ⊢ *f*⌢*schopstar f* = ((*f*⌢*schopstar f* ∧ *empty*) ∨ (*f*⌢*schopstar f* ∧ *more*))
  by (*auto simp*: *empty-d-def*)
from *12 13 14 15* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopSChopPlusImpSChopPlus*:
⊢ *f*⌢ (*f*⌢*schopstar f*) ⟶ *f*⌢*schopstar f*
proof −
have *1*: ⊢ *empty* ∨ *more* **by** (*auto simp*: *empty-d-def*)
hence *2*: ⊢ *f* ⟶ *empty* ∨ (*f* ∧ *more*) **by** *auto*
hence *3*: ⊢ *f*⌢ (*f*⌢*schopstar f*) ⟶ (*f*⌢*schopstar f*) ∨ (*f* ∧ *more*)⌢(*f*⌢*schopstar f*)
 by (*rule EmptyOrSChopImpRule*)
have *4*: ⊢ *f*⌢*schopstar f* = ((*f* ∧ *finite*) ∨ (*f* ∧ *more*)⌢ (*f*⌢*schopstar f*))
 by (*rule SChopPlusEqv*)
hence *5*: ⊢ (*f* ∧ *more*)⌢ (*f*⌢*schopstar f*) ⟶ *f*⌢*schopstar f* **by** *auto*
from *3 5* **show** *?thesis* **using** *SChopPlusImpSCS RightSChopImpSChop* **by** *blast*
qed

lemma *SCSImpSCS*:
assumes ⊢ *f* ⟶ *g*
shows ⊢ *schopstar f* ⟶ *schopstar g*
using *assms*
by (*metis AndSChopB EmptyImpSCS Prop02 Prop10 SChopPlusImpSCS SChopstarImp*
  inteq-reflection lift-imp-trans)

lemma *SChopPlusImpSChopPlus*:
assumes ⊢ *f* ⟶ *g*
shows ⊢ *f*⌢*schopstar f* ⟶ *g*⌢*schopstar g*
using *assms* **by** (*simp add*: *SCSImpSCS SChopImpSChop*)

lemma *SChopPlusIntro*:
assumes ⊢ *f* ⟶ (*g* ∧ *finite*) ∨ (*g* ∧ *more*)⌢ *f*
shows ⊢ *f* ∧ *finite* ⟶ *g*⌢*schopstar g*
proof −
have *1*: ⊢ *f* ∧ ¬ (*g* ∧ *finite*) ⟶ (*g* ∧ *more*)⌢ *f* **using** *assms* **by** *auto*
have *2*: ⊢ *g*⌢*schopstar g* = ((*g* ∧ *finite*) ∨ (*g* ∧ *more*)⌢ (*g*⌢*schopstar g*))
 by (*rule SChopPlusEqv*)
have *3*: ⊢ *f* ∧ ¬ (*g*⌢*schopstar g*) ⟶
   (*g* ∧ *more*)⌢ *f* ∧ ¬ ((*g* ∧ *more*)⌢ (*g*⌢*schopstar g*)) **using** *1 2*
 by *fastforce*
have *4*: ⊢ *g* ∧ *more* ⟶ *more* **by** *auto*
from *3 4* **show** *?thesis* **using** *SChopContraB* **by** *blast*

qed

lemma *SChopPlusElim*:
assumes $\vdash\ \ f \longrightarrow g$
$\vdash (f \wedge\ \ more\ )\frown g \longrightarrow g$
shows $\vdash f\frown schopstar\ f\ \longrightarrow g$
proof −
have *1*: $\vdash f \vee (f \wedge more)\frown g \longrightarrow g$
using *assms Prop02* **by** *blast*
have *2*: $\vdash schopstar\ f\ \frown\ f \longrightarrow g$
using *SChopstarInductMoreL 1* **by** *blast*
from *2* **show** *?thesis*
using *SChopplusCommute* **by** (*metis Prop10 Prop12 SChopAndA inteq-reflection*)
qed

lemma *SChopPlusElimWithoutMore*:
assumes $\vdash\ \ f \longrightarrow g$
$\vdash f\frown g \longrightarrow g$
shows $\vdash f\frown schopstar\ f\ \longrightarrow g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *blast*
have *2*: $\vdash (f\frown g) \longrightarrow g$ **using** *assms* **by** *blast*
have *3*: $\vdash (f \wedge\ \ more\ )\frown g \longrightarrow f\frown g$ **by** (*rule AndSChopA*)
have *4*: $\vdash (f \wedge\ \ more\ )\frown g \longrightarrow g$ **using** *2 3 lift-imp-trans* **by** *blast*
from *1 4* **show** *?thesis* **using** *SChopPlusElim* **by** *blast*
qed

lemma *SChopPlusEqvSChopPlus*:
assumes $\vdash\ \ f = g$
shows $\vdash f\frown schopstar\ f\ = g\frown schopstar\ g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow g$ **by** *auto*
hence *3*: $\vdash f\frown schopstar\ f\ \longrightarrow g\frown schopstar\ g$ **by** (*rule SChopPlusImpSChopPlus*)
have *4*: $\vdash g \longrightarrow f$ **using** *1* **by** *auto*
hence *5*: $\vdash\ g\frown schopstar\ g \longrightarrow f\frown schopstar\ f$ **by** (*rule SChopPlusImpSChopPlus*)
from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *SCSEqvSCS*:
assumes $\vdash\ \ f = g$
shows $\vdash schopstar\ f = schopstar\ g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f\frown schopstar\ f\ = g\frown schopstar\ g$ **by** (*rule SChopPlusEqvSChopPlus*)
hence *3*: $\vdash\ (empty\ \vee f\frown schopstar\ f)\ = (empty\ \vee\ g\frown schopstar\ g)$ **by** *auto*
from *3* **show** *?thesis* **using** *SCSEqvOrSChopSCS* **by** (*metis int-eq*)
qed

lemma *AndSCSA*:
$\vdash$ *schopstar* $(f \wedge g) \longrightarrow$ *schopstar* f
proof $-$
have *1*: $\vdash f \wedge g \longrightarrow f$ **by** *auto*
from *1* **show** *?thesis* **using** *SCSImpSCS* **by** *blast*
qed

lemma *AndSCSB*:
$\vdash$ *schopstar* $(f \wedge g) \longrightarrow$ *schopstar* g
proof $-$
have *1*: $\vdash f \wedge g \longrightarrow g$ **by** *auto*
from *1* **show** *?thesis* **using** *SCSImpSCS* **by** *blast*
qed

lemma *SCSIntro*:
assumes $\vdash f \wedge \mathit{more} \longrightarrow (g \wedge \mathit{more}) \frown f$
shows $\vdash f \wedge \mathit{finite} \longrightarrow \mathit{schopstar}\ g$
proof $-$
have *1*: $\vdash f \wedge \mathit{more} \longrightarrow (g \wedge \mathit{more}) \frown f$
using *assms* **by** *auto*
have *2*: $\vdash \mathit{more} = (\neg \mathit{empty})$
by (*auto simp*: *empty-d-def*)
have *3*: $\vdash f \wedge \neg \mathit{empty} \longrightarrow (g \wedge \mathit{more}) \frown f$
using *1 2* **by** *fastforce*
have *4*: $\vdash \mathit{schopstar}\ g = (\mathit{empty} \vee (g \wedge \mathit{more}) \frown \mathit{schopstar}\ g)$
by (*rule SChopstarEqv*)
hence *41*: $\vdash (\neg(\mathit{empty} \vee (g \wedge \mathit{more}) \frown \mathit{schopstar}\ g)) = (\neg \mathit{empty} \wedge \neg((g \wedge \mathit{more}) \frown \mathit{schopstar}\ g))$
by *fastforce*
have *411*: $\vdash (\neg \mathit{empty} \wedge \neg((g \wedge \mathit{more}) \frown \mathit{schopstar}\ g)) = (\mathit{more} \wedge \neg((g \wedge \mathit{more}) \frown \mathit{schopstar}\ g))$
using *NotEmptyEqvMore* **by** *fastforce*
have *42*: $\vdash \neg(\mathit{schopstar}\ g) = (\mathit{more} \wedge \neg((g \wedge \mathit{more}) \frown \mathit{schopstar}\ g))$
using *4 41 411* **by** *fastforce*
have *43*: $\vdash f \wedge \neg(\mathit{schopstar}\ g) \longrightarrow f \wedge \mathit{more} \wedge \neg((g \wedge \mathit{more}) \frown \mathit{schopstar}\ g)$
using *42* **by** *fastforce*
have *44*: $\vdash f \wedge \mathit{more} \wedge \neg((g \wedge \mathit{more}) \frown \mathit{schopstar}\ g) \longrightarrow$
$(g \wedge \mathit{more}) \frown f \wedge \neg((g \wedge \mathit{more}) \frown \mathit{schopstar}\ g)$
using *3 43 1* **by** *auto*
have *5*: $\vdash f \wedge \neg(\mathit{schopstar}\ g) \longrightarrow$
$(g \wedge \mathit{more}) \frown f \wedge \neg((g \wedge \mathit{more}) \frown \mathit{schopstar}\ g)$
using *43 44 lift-imp-trans* **by** *fastforce*
have *6*: $\vdash g \wedge \mathit{more} \longrightarrow \mathit{more}$
by *auto*
from *5 6* **show** *?thesis* **using** *SChopContraB* **by** *blast*
qed

lemma *SCSElimWithoutMore*:
assumes $\vdash \mathit{empty} \longrightarrow g$
$\vdash f \frown g \longrightarrow g$
shows $\vdash \mathit{schopstar}\ f \longrightarrow g$

proof −
have *1*: ⊢ *empty* ⟶ *g* **using** *assms* **by** *blast*
have *2*: ⊢ *f* ⌢ *g* ⟶ *g* **using** *assms* **by** *blast*
have *3*: ⊢ (*f* ∧ *more*)⌢ *g* ⟶ *f* ⌢ *g* **by** (*rule AndSChopA*)
have *4*: ⊢ (*f* ∧ *more*)⌢ *g* ⟶ *g* **using** *2 3 lift-imp-trans* **by** *blast*
from *1 4* **show** *?thesis* **using** *SCSElim* **by** *blast*
qed

lemma *SChopAssocB*:
⊢ (*f*⌢*g*)⌢*h* = *f*⌢(*g*⌢*h*)
using *SChopAssoc* **by** *fastforce*

lemma *SCSChopEqvSChopOrRule*:
assumes ⊢ *f*= (*schopstar g*⌢ *h*)
shows ⊢ *f* = ((*g*⌢ *f*) ∨ *h*)
proof −
have *1*: ⊢ *f* = (*schopstar g*⌢ *h*) **using** *assms* **by** *auto*
have *2*: ⊢ *schopstar g* = (*empty* ∨ (*g*⌢ *schopstar g*)) **by** (*rule SCSEqvOrSChopSCS*)
hence *3*: ⊢ *schopstar g*⌢ *h* = (*h* ∨ ((*g*⌢ *schopstar g*)⌢ *h*)) **by** (*rule EmptyOrSChopEqvRule*)
have *4*: ⊢ (*g*⌢ *schopstar g*)⌢ *h* = *g*⌢ (*schopstar g*⌢ *h*) **by** (*rule SChopAssocB*)
hence *41*: ⊢ *schopstar g*⌢ *h* = (*h* ∨ *g*⌢ (*schopstar g*⌢ *h*)) **using** *3* **by** *fastforce*
have *5*: ⊢ *g*⌢ *f* = *g*⌢ (*schopstar g*⌢ *h*) **using** *1* **by** (*rule RightSChopEqvSChop*)
hence *6*: ⊢ (*schopstar g*⌢ *h*) = (*h* ∨ *g*⌢ *f*) **using** *41* **by** *fastforce*
hence *61*: ⊢ (*schopstar g*⌢ *h*) = ((*g*⌢ *f*) ∨ *h*) **by** *auto*
from *1 61* **show** *?thesis* **by** *fastforce*
qed

lemma *SCSChopIntroRule*:
assumes ⊢ *f* ∧ ¬ *h* ⟶ *g*⌢ *f*
⊢ *g* ⟶ *more*
shows ⊢ *f* ∧ *finite* ⟶ *schopstar g*⌢ *h*
proof −
have *1*: ⊢ *f* ∧ ¬ *h* ⟶ *g*⌢ *f*
using *assms* **by** *blast*
have *2*: ⊢ *g* ⟶ *more*
using *assms* **by** *blast*
hence *3*: ⊢ *g* ⟶ *g* ∧ *more*
by *auto*
hence *4*: ⊢ *g*⌢ *f* ⟶ (*g* ∧ *more*)⌢ *f*
by (*rule LeftSChopImpSChop*)
have *5*: ⊢ *f* ⟶ (*g* ∧ *more*)⌢ *f* ∨ *h*
using *1 4* **by** *fastforce*
have *6*: ⊢ *schopstar g* = (*empty* ∨ (*g* ∧ *more*)⌢ *schopstar g*)
by (*rule SChopstarEqv*)
hence *7*: ⊢ (*schopstar g*)⌢ *h* = (*h* ∨ ((*g* ∧ *more*)⌢ *schopstar g*)⌢ *h*)
by (*rule EmptyOrSChopEqvRule*)
have *8*: ⊢ ((*g* ∧ *more*)⌢ *schopstar g*)⌢ *h* = (*g* ∧ *more*)⌢ (*schopstar g*⌢ *h*)
by (*rule SChopAssocB*)
have *9*: ⊢ (*schopstar g*)⌢ *h* = (*h* ∨ (*g* ∧ *more*)⌢ (*schopstar g*⌢ *h*))
using *7 8* **by** *fastforce*

have *10*: $\vdash f \wedge \neg$ (*schopstar g*$\frown$ *h*) $\longrightarrow$ (*g* $\wedge$ *more*)$\frown$ *f* $\wedge$ $\neg$ ((*g* $\wedge$ *more*)$\frown$ (*schopstar g*$\frown$ *h*))
  using *5 9* **by** *fastforce*
have *11*: $\vdash$ *g* $\wedge$ *more* $\longrightarrow$ *more*
  by *fastforce*
from *10 11* **show** *?thesis* **using** *SChopContraB* **by** *blast*
qed

lemma *BoxImpTrueSChopAndEmpty*:
$\vdash \Box$ *f* $\wedge$ *finite*$\longrightarrow$ #*True*$\frown$(*f* $\wedge$ *empty*)
by (*metis BoxAndSChopImport DiamondEmptyEqvFinite TrueSChopEqvDiamond inteq-reflection*)

lemma *BoxInitAndMoreImpBoxInitAndMoreAndSFinInit*:
$\vdash \Box$(*init w*) $\wedge$ *more* $\wedge$ *finite* $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *sfin* (*init w*)
proof −
have *1*: $\vdash$ *sfin* (*init w*) = #*True* $\frown$ (*init w* $\wedge$ *empty*) **using** *SFinEqvTrueSChopAndEmpty* **by** *blast*
have *2*: $\vdash \Box$(*init w*) $\wedge$ *finite* $\longrightarrow$ #*True*$\frown$(*init w* $\wedge$ *empty*) **by** (*rule BoxImpTrueSChopAndEmpty*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SCSImpBox*:
assumes $\vdash$ *f* $\longrightarrow$ *empty* $\vee$ (($\Box$ (*init w*) $\wedge$ *more*))$\frown$ *f*
shows $\vdash$ (*init w* $\wedge$ *f*) $\wedge$ *finite* $\longrightarrow$ $\Box$ (*init w*)
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *empty* $\vee$ (($\Box$(*init w*) $\wedge$ *more*))$\frown$ *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*))$\frown$ *f*
  using *1 2* **by** *fastforce*
have *4*: $\vdash$ $\Box$ (*init w*) $\wedge$ *more* $\wedge$ *finite* $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *sfin* (*init w*)
  by (*rule BoxInitAndMoreImpBoxInitAndMoreAndSFinInit*)
have *41*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite* $\longrightarrow$ (($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin* (*init w*)
using *4* **by** *auto*
hence *5*: $\vdash$ (($\Box$(*init w*) $\wedge$ *more*))$\frown$*f* $\longrightarrow$ ((($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin*(*init w*))$\frown$ *f*
 by (*metis FiniteImp LeftChopImpChop inteq-reflection schop-d-def*)
have *6*: $\vdash$ ((($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin* (*init w*))$\frown$ *f* =
   (($\Box$(*init w*) $\wedge$ *more*))$\frown$ (*init w* $\wedge$ *f*)
  using *AndSFinSChopEqvStateAndSChop*
  by (*metis* (*no-types*, *lifting*) *41 Prop10 Prop12 int-eq schop-d-def*)
have *7*: $\vdash$ $\neg$($\Box$(*init w*)) $\longrightarrow$ ($\Box$ (*init w*)) *syields* ($\neg$($\Box$ (*init w*)))
  by (*rule NotBoxStateImpBoxSYieldsNotBox*)
have *8*: $\vdash$ ($\Box$(*init w*)) *syields* ($\neg$ ($\Box$ (*init w*))) $\longrightarrow$
   (($\Box$ (*init w*) $\wedge$ *more*)) *syields* ($\neg$($\Box$(*init w*)))
  using *AndSYieldsA* **by** (*metis*)
have *9*: $\vdash$ (($\Box$(*init w*) $\wedge$ *more*))$\frown$(*init w* $\wedge$ *f*) $\wedge$ (($\Box$(*init w*) $\wedge$ *more*)) *syields* ($\neg$($\Box$(*init w*)))
   $\longrightarrow$
   (($\Box$ (*init w*) $\wedge$ *more*))$\frown$ ((*init w* $\wedge$ *f*) $\wedge$ $\neg$ ($\Box$ (*init w*)))
  by (*rule SChopAndSYieldsImp*)
have *10*: $\vdash$ (*init w* $\wedge$ *f*) $\wedge$ $\neg$($\Box$ (*init w*)) $\longrightarrow$
   (($\Box$(*init w*) $\wedge$ *more*))$\frown$ ((*init w* $\wedge$ *f*) $\wedge$ $\neg$($\Box$ (*init w*)))

using *3 5 6 7 8 9* **by** *fastforce*
have *11*: ⊢ ((□(*init w*) ∧ *more*))⌢ ((*init w* ∧ *f*) ∧ ¬(□ (*init w*))) ⟶
more ⌢ ((*init w* ∧ *f*) ∧ ¬(□ (*init w*)))
using *AndSChopB* **by** *blast*
have *12*: ⊢ (*init w* ∧ *f*) ∧ ¬(□ (*init w*)) ⟶
more ⌢ ((*init w* ∧ *f*) ∧ ¬(□ (*init w*)))
using *10 11* **by** *fastforce*
from *12* **show** *?thesis* **using** *MoreSChopContra* **by** *blast*
qed

lemma *BoxSCSEqvBox*:
⊢ (*init w* ∧ *schopstar* (□(*init w*))) = (□ (*init w*) ∧ *finite*)
proof −
have *1*: ⊢ □(*init w*)⌢(□(*init w*) ∧ *finite*)⟶ □(*init w*) ∧ *finite*
by (*metis BoxStateAndChopEqvChop FiniteChopFiniteEqvFinite int-iffD2 inteq-reflection schop-d-def*)
have *2*: ⊢ (*init w* ∧ *empty*) ⟶ □(*init w*) ∧ *finite*
using *EmptyImpFinite StateAndEmptyImpBoxState* **by** *fastforce*
have *3*: ⊢ (*init w* ∧ *empty*) ∨ □(*init w*)⌢(□(*init w*) ∧ *finite*) ⟶ □(*init w*) ∧ *finite*
using *1 2* **by** *fastforce*
have *4*: ⊢ (*init w* ∧ *empty*)⌢*schopstar* (□(*init w*)) ⟶ □ (*init w*) ∧ *finite*
using *SChopstarInductR 3*
by (*metis* (*no-types*, *lifting*) *BoxBoxImpBox BoxEqvBoxBox BoxStateSChopBoxEqvBox Prop02 Prop12 SCSAndFinite SCSImpSCSSCS SChopImpFinite StateAndEmptyImpBoxState int-eq*)
have *5*: ⊢ *init w* ∧ *schopstar* (□(*init w*)) ⟶ □ (*init w*) ∧ *finite*
using *4 StateAndEmptySChop* **by** *fastforce*
have *11*: ⊢ □ (*init w*) ⟶ (*init w*)
using *BoxElim* **by** *blast*
have *12*: ⊢ □(*init w*) ∧ *finite* ⟶ *schopstar* (□ (*init w*))
by (*rule ImpSCS*)
have *13*: ⊢ □ (*init w*) ∧ *finite* ⟶ *init w* ∧ *schopstar* (□ (*init w*))
using *11 12* **by** *fastforce*
from *5 13* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateAndSCSEqvSCS*:
⊢ (□(*init w*) ∧ *schopstar f*) = (*init w* ∧ *schopstar* (□(*init w*) ∧ *f*))
proof −
have *1*: ⊢ □ (*init w*) ⟶ *init w*
using *BoxElim* **by** *blast*
have *2*: ⊢ (*schopstar f* ∧ *more*) = (*f* ∧ *more*)⌢ *schopstar f*
by (*rule SCSAndMoreEqvAndMoreSChop*)
have *21*: ⊢ (*f* ∧ *more*)⌢ *schopstar f* = (*finite* ∧ (*f* ∧ *more*)⌢ *schopstar f*)
by (*metis 2 AndMoreAndFiniteEqvAndFmore SCSAndMoreEqvAndFMoreSChop inteq-reflection lift-and-com*)
have *22*: ⊢ (□ (*init w*) ∧ (*f* ∧ *more*) ⌢ *schopstar f*) =
(□ (*init w*) ∧ *finite* ∧ (*f* ∧ *more*) ⌢ *schopstar f*)

using *21* **by** *auto*
have *23*: ⊢((□ (*init w*) ∧ *schopstar f*) ∧ *finite*) = (□ (*init w*) ∧ *schopstar f*)

using *SCSAndFinite* **by** *fastforce*
have *3*: $\vdash (\Box(\ init\ w) \wedge ((f \wedge\ \ more\ )\frown schopstar\ f)) =$
$((\Box\ (init\ w) \wedge f \wedge\ \ more\ )\frown (\Box\ (init\ w) \wedge schopstar\ f))$
using *22 23 BoxStateAndSChopEqvSChop*[*of w LIFT*$(f \wedge\ \ more\ )$ *LIFT*$(schopstar\ f)$]
by (*metis Prop10 Prop12 SChopImpFinite int-eq int-iffD2*)
have *4*: $\vdash \Box\ (init\ w) \wedge f \wedge\ \ more\ \longrightarrow (\Box\ (init\ w) \wedge f) \wedge\ \ more$
by *auto*
hence *5*: $\vdash (\Box\ (init\ w) \wedge f \wedge\ \ more\ )\frown (\Box\ (init\ w) \wedge schopstar\ f) \longrightarrow$
$((\Box\ (init\ w) \wedge f) \wedge\ \ more\ )\frown (\Box\ (init\ w) \wedge schopstar\ f)$
by (*rule LeftSChopImpSChop*)
have *6*: $\vdash (\Box(\ init\ w) \wedge schopstar\ f) \wedge\ \ more\ \ \longrightarrow$
$((\Box\ (init\ w) \wedge f) \wedge\ \ more\ )\frown (\Box\ (init\ w) \wedge schopstar\ f)$
using *2 3 5* **by** *fastforce*
hence *7*: $\vdash(\ \Box\ (init\ w) \wedge schopstar\ f) \wedge finite \longrightarrow schopstar\ (\Box\ (init\ w) \wedge f)$
using *SCSIntro* **by** *blast*
have *70*: $\vdash \Box\ (init\ w) \wedge schopstar\ f \longrightarrow schopstar\ (\Box\ (init\ w) \wedge f)$
using *SCSAndFinite 7* **using** *Valid-def* **by** *fastforce*
have *71*: $\vdash init\ w \wedge \Box\ (init\ w) \wedge schopstar\ f \longrightarrow init\ w \wedge schopstar\ (\Box\ (init\ w) \wedge f)$
using *70 SCSAndFinite* **by** *fastforce*
have *8*: $\vdash \Box(\ init\ w) \wedge schopstar\ f \longrightarrow init\ w \wedge schopstar\ (\Box\ (init\ w) \wedge f)$
using *1 71* **by** *fastforce*
have *11*: $\vdash schopstar\ (\Box\ (init\ w) \wedge f) \longrightarrow schopstar\ (\Box\ (init\ w))$
by (*rule AndSCSA*)
have *12*: $\vdash (init\ w \wedge schopstar\ (\Box\ (init\ w))) = (\Box\ (init\ w) \wedge finite)$
by (*rule BoxSCSEqvBox*)
have *13*: $\vdash schopstar\ (\Box\ (init\ w) \wedge f) \longrightarrow schopstar\ f$
by (*rule AndSCSB*)
have *14*: $\vdash init\ w \wedge schopstar\ (\Box\ (init\ w) \wedge f){\longrightarrow}init\ w \wedge schopstar\ (\Box\ (init\ w)) \wedge schopstar\ f$
using *11 13* **by** *fastforce*
have *15*: $\vdash init\ w \wedge schopstar\ (\Box\ (init\ w)) \wedge schopstar\ f \longrightarrow \Box\ (init\ w) \wedge schopstar\ f$
using *12* **by** *auto*
have *16*: $\vdash init\ w \wedge schopstar\ (\Box\ (init\ w) \wedge f) \longrightarrow \Box\ (init\ w) \wedge schopstar\ f$
using *14 15 lift-imp-trans* **by** *blast*
from *8 16* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaSCSImpSCS*:
$\vdash\ \ \ sba\ (f \longrightarrow g) \longrightarrow schopstar\ f \longrightarrow schopstar\ g$
proof −
have *1*: $\vdash schopstar\ f =\ (empty\ \vee\ (f \wedge\ \ more\ )\frown schopstar\ f)$
by (*rule SChopstarEqv*)
have *2*: $\vdash schopstar\ g =\ (empty\ \vee\ (g \wedge\ \ more\ )\frown schopstar\ g)$
by (*rule SChopstarEqv*)
have *21*: $\vdash (\neg(schopstar\ g)) =\ (\neg empty\ \ \wedge \neg(\ (g \wedge\ \ more\ )\frown schopstar\ g))$
using *2* **by** *fastforce*
hence *22*: $\vdash (\neg(schopstar\ g)) =\ (more\ \ \wedge \neg(\ (g \wedge\ \ more\ )\frown schopstar\ g))$
using *NotEmptyEqvMore* **by** *fastforce*
have *3*: $\vdash schopstar\ f \wedge \neg\ (schopstar\ g) \longrightarrow$
$(empty\ \ \vee (f \wedge\ \ more\ )\frown schopstar\ f) \wedge more \wedge \neg\ ((g \wedge\ \ more\ )\frown schopstar\ g)$
using *1 22* **by** *fastforce*

have *31*: $\vdash$ ((*empty* $\lor$ (*f* $\land$ *more*)$\frown$ *schopstar f*) $\land$ *more*) = ((*f* $\land$ *more*)$\frown$ *schopstar f* $\land$ *more*)
by (*auto simp*: *empty-d-def*)
have *32*: $\vdash$ *schopstar f* $\land$ $\neg$ (*schopstar g*) $\longrightarrow$
(*f* $\land$ *more*)$\frown$ *schopstar f* $\land$ $\neg$ ((*g* $\land$ *more*)$\frown$ *schopstar g*)
using *3 31* **by** *fastforce*
have *4*: $\vdash$ (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*f* $\land$ *more* $\longrightarrow$ *g* $\land$ *more*)
by *auto*
hence *5*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ *sba* (*f* $\land$ *more* $\longrightarrow$ *g* $\land$ *more*)
by (*rule SBaImpSBa*)
have *6*: $\vdash$ *sba* (*f* $\land$ *more* $\longrightarrow$ *g* $\land$ *more*) $\longrightarrow$
(*f* $\land$ *more*)$\frown$ *schopstar f* $\longrightarrow$ (*g* $\land$ *more*)$\frown$ *schopstar f*
by (*rule SBaLeftSChopImpSChop*)
have *7*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\land$ (*f* $\land$ *more*)$\frown$ *schopstar f* $\longrightarrow$ (*g* $\land$ *more*)$\frown$ *schopstar f*
using *5 6* **by** *fastforce*
have *8*: $\vdash$ (*g* $\land$ *more*)$\frown$ *schopstar f* $\land$ $\neg$ ((*g* $\land$ *more*)$\frown$ *schopstar g*)
$\longrightarrow$ (*g* $\land$ *more*)$\frown$ (*schopstar f* $\land$ $\neg$ (*schopstar g*))
by (*rule SChopAndNotSChopImp*)
have *9*: $\vdash$ (*g* $\land$ *more*)$\frown$ (*schopstar f* $\land$ $\neg$ (*schopstar g*)) $\longrightarrow$
more $\frown$ (*schopstar f* $\land$ $\neg$ (*schopstar g*))
by (*rule AndSChopB*)
have *10*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\land$ *finite* $\longrightarrow$ *more* $\frown$ (*schopstar f* $\land$ $\neg$ (*schopstar g*)) $\longrightarrow$
more $\frown$ (*sba* (*f* $\longrightarrow$ *g*) $\land$ *schopstar f* $\land$ $\neg$ (*schopstar g*))
using *SBaSChopImpSChopSBa* **by** *fastforce*
have *11*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\land$ *finite* $\land$ *schopstar f* $\land$ $\neg$ (*schopstar g*) $\longrightarrow$
more $\frown$ (*sba* (*f* $\longrightarrow$ *g*) $\land$ *schopstar f* $\land$ $\neg$ (*schopstar g*))
using *32 7 8 9 10* **by** *fastforce*
have *12*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\land$ *schopstar f* $\land$ $\neg$ (*schopstar g*) $\longrightarrow$
more $\frown$ (*sba* (*f* $\longrightarrow$ *g*) $\land$ *schopstar f* $\land$ $\neg$ (*schopstar g*))
using *11 SCSAndFinite* **by** *fastforce*
hence *12*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ((*sba* (*f* $\longrightarrow$ *g*)) $\land$ (*schopstar f*) $\land$ ($\neg$ (*schopstar g*)))
using *MoreSChopLoop* **by** *blast*
have *13*: $\vdash$ (*sba* (*f* $\longrightarrow$ *g*)) $\land$ *finite* $\land$ (*schopstar f*) $\longrightarrow$ (*schopstar g*)
using *12* **by** *fastforce*
have *14*: $\vdash$ (*sba* (*f* $\longrightarrow$ *g*)) $\land$ (*schopstar f*) $\longrightarrow$ (*schopstar g*)
using *SCSAndFinite 13* **by** *fastforce*
from *14* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaSCSEqvSCS*:
$\vdash$ *sba* (*f* = *g*) $\longrightarrow$ (*schopstar f* = *schopstar g*)
proof −
have *1*: $\vdash$ *sba* (*f* = *g*) = (*sba* (*f* $\longrightarrow$ *g*) $\land$ *sba* (*g* $\longrightarrow$ *f*))
by (*auto simp*: *sba-defs sum.case-eq-if*)
have *2*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*schopstar f* $\longrightarrow$ *schopstar g*) **by** (*rule SBaSCSImpSCS*)
have *3*: $\vdash$ *sba* (*g* $\longrightarrow$ *f*) $\longrightarrow$ (*schopstar g* $\longrightarrow$ *schopstar f*) **by** (*rule SBaSCSImpSCS*)
have *4*: $\vdash$ *sba* (*f* = *g*) $\longrightarrow$ (*schopstar f* $\longrightarrow$ *schopstar g*) $\land$ (*schopstar g* $\longrightarrow$ *schopstar f*)
using *1 2 3* **by** *fastforce*
have *5*: $\vdash$ ((*schopstar f* $\longrightarrow$ *schopstar g*) $\land$ (*schopstar g* $\longrightarrow$ *schopstar f*)) =
(*schopstar f* = *schopstar g*) **by** *auto*

from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *SBaAndSCSImport*:
$\vdash$ *sba* $f \wedge$ *schopstar* $g \longrightarrow$ *schopstar* $(f \wedge g)$
proof –
have *1*: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
hence *2*: $\vdash$ *sba* $f \longrightarrow$ *sba* $(g \longrightarrow f \wedge g)$ **by** (*rule SBaImpSBa*)
have *3*: $\vdash$ *sba* $(g \longrightarrow f \wedge g) \longrightarrow$ *schopstar* $g \longrightarrow$ *schopstar* $(f \wedge g)$ **by** (*rule SBaSCSImpSCS*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SCSSkipImpFinite*:
$\vdash$ *schopstar skip* $\longrightarrow$ *finite*
by (*simp add*: *EmptyImpFinite SCSElim SChopImpFinite*)

lemma *FiniteImpSCSSkip*:
$\vdash$ *finite* $\longrightarrow$ *schopstar skip*
by (*metis* (*no-types*, *hide-lams*) *AndMoreAndFiniteEqvAndFmore ChopAndB ChopEmpty EmptyImpFinite FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite Prop10 SCSIntro inteq-reflection lift-and-com schop-d-def*)

lemma *SCSSkipEqvFinite*:
$\vdash$ *schopstar skip* = *finite*
using *SCSSkipImpFinite FiniteImpSCSSkip* **by** *fastforce*

14.8 Properties of Omega

lemma *SOmegaIntro*:
assumes $\vdash h \longrightarrow (f \wedge more)\frown h$
shows $\vdash h \wedge inf \longrightarrow f^{\omega}$
proof –
have *1*: $\vdash h \longrightarrow (f \wedge more)\frown h$ **using** *assms* **by** *auto*
have *2*: $\vdash \Box\ (h \longrightarrow (f \wedge more)\frown h)$ **by** (*simp add*: *BoxGen assms*)
from *1 2* **show** *?thesis* **using** *SOmegaInduct* **by** *fastforce*
qed

14.9 Properties of SWhile

lemma *SWhileEqvIf*:
$\vdash$ *swhile* (*init w*) *do* f = *if*$_i$ (*init w*) *then* ($f\frown$ (*swhile* (*init w*) *do* f)) *else* *empty*
proof –
have *1*: $\vdash$ *swhile* (*init w*) *do* f = (*schopstar* ((*init w*) $\wedge f$) $\wedge$ *sfin* ($\neg$ (*init w*)))
by (*simp add*: *swhile-d-def*)
have *2*: $\vdash$ *schopstar* (*init w* $\wedge f$) = (*empty* $\vee$ ((*init w* $\wedge f$)$\frown$ *schopstar* (*init w* $\wedge f$)))
by (*rule SCSEqvOrSChopSCS*)
have *21*: $\vdash$ (*schopstar* ((*init w*) $\wedge f$) $\wedge$ *sfin* ($\neg$ (*init w*))) =
((*empty* $\vee$ ((*init w* $\wedge f$)$\frown$ *schopstar* (*init w* $\wedge f$))) $\wedge$ *sfin* ($\neg$ (*init w*)))

using *2* **by** *fastforce*
have *22*: $\vdash ((empty\ \lor\ ((init\ w \land f) \frown schopstar\ (init\ w \land f))) \land\ sfin\ (\neg\ (init\ w))) =$
$((\ empty \land sfin\ (\neg(init\ w))) \lor$
$(\ ((init\ w \land f) \frown schopstar\ (init\ w \land f)) \land\ sfin\ (\neg\ (init\ w))))$
by *auto*
have *3*: $\vdash\ (empty\ \land\ sfin\ (\neg\ (\ init\ w))) = (\neg\ (\ init\ w) \land\ empty)$
by (*metis Prop04 SFinAndEmpty lift-and-com*)
have *4*: $\vdash (init\ w \land f) \frown schopstar\ (init\ w \land f) = (init\ w \land (f \frown schopstar\ (init\ w \land f)))$
by (*rule StateAndSChop*)
have *41*: $\vdash (((init\ w \land f) \frown schopstar\ (init\ w \land f)) \land\ sfin\ (\neg\ (init\ w))) =$
$(init\ w \land (f \frown schopstar\ (init\ w \land f)) \land\ sfin\ (\neg\ (init\ w)))$
using *4* **by** *auto*
have *42*: $\vdash (init\ w \land (f \frown schopstar\ (init\ w \land f)) \land\ sfin\ (\neg\ (init\ w))) =$
$(init\ w \land (f \frown schopstar\ (init\ w \land f)) \land\ sfin\ \ (init\ (\neg\ w)))$
using *Initprop*(*2*) **by** (*metis StateAndEmptySChop int-eq*)
have *5*: $\vdash ((f \frown (schopstar\ (init\ w \land f))) \land\ (sfin\ (\ init\ (\neg\ w))))$
$= (f \frown (schopstar\ (init\ w \land f) \land\ (sfin\ (\ init\ (\neg\ w)))))$
by (*rule SChopAndSFin*)
have *51*: $\vdash (f \frown (schopstar\ (init\ w \land f) \land\ (sfin\ (\ init\ (\neg\ w))))) =$
$(f \frown (schopstar\ (init\ w \land f) \land\ (sfin\ (\neg\ (\ init\ \ w)))))$
using *Initprop*(*2*)
by (*metis 21 RightSChopEqvSChop inteq-reflection*)
have *52*: $\vdash (init\ w \land (f \frown schopstar\ (init\ w \land f)) \land\ sfin\ (\neg\ (init\ w))) =$
$(init\ w \land (f \frown (schopstar\ (init\ w \land f) \land\ sfin\ (\neg\ (\ init\ \ w)))))$
using *42 5 51* **by** *fastforce*
have *6*: $\vdash (f \frown (schopstar\ (init\ w \land f) \land\ sfin\ (\neg\ (\ init\ w)))) = f \frown\ swhile\ \ (init\ w)\ \ do\ \ f$
by (*simp add*: *swhile-d-def*)
have *61*: $\vdash (init\ w \land (f \frown (schopstar\ (init\ w \land f) \land\ sfin\ (\neg\ (\ init\ \ w))))) =$
$(init\ w \land\ \ (f \frown\ \ swhile\ \ (init\ w)\ \ do\ \ f))$ **using** *6*
by *auto*
have *62*: $\vdash (\ empty \land\ sfin\ (\neg\ \ (init\ w))) \lor$
$(\ ((init\ w \land f) \frown schopstar\ (init\ w \land f)) \land\ sfin\ (\neg\ \ (init\ w)))$
$= (\neg\ (\ init\ w) \land\ empty\ ) \lor\ (init\ w \land (f \frown\ swhile\ \ (init\ w)\ \ do\ \ f))$
using *21 22 3 4 52 61* **by** *fastforce*
have *7*: $\vdash\ swhile\ \ (init\ w)\ \ do\ \ f$
$= ((\neg\ (\ init\ w) \land\ empty\ ) \lor\ (init\ w \land (f \frown\ swhile\ \ (init\ w)\ \ do\ \ f)))$
using *1 21 22 62*
by (*metis 3 41 42 5 51 inteq-reflection*)
have *71*: $\vdash if_i\ \ (init\ w)\ \ then\ \ (f \frown (\ swhile\ \ (init\ w)\ \ do\ \ f))\ \ else\ \ \ empty =$
$((\neg\ (\ init\ w) \land\ empty\ ) \lor\ (init\ w \land (f \frown\ swhile\ \ (init\ w)\ \ do\ \ f)))$
by (*auto simp*: *ifthenelse-d-def*)
from *7 71* **show** *?thesis* **by** *fastforce*
qed

lemma *SWhileSChopEqvIf*:
$\vdash\ \ (\ swhile\ (\ init\ w)\ do\ f) \frown g =\ if_i\ (init\ w)\ then\ (f \frown ((swhile\ (\ init\ w)\ do\ f) \frown\ g))\ \ else\ \ g$
proof −
have *1*: $\vdash\ \ swhile\ \ (init\ w)\ \ do\ \ f\ =$
$if_i\ \ (init\ w)\ \ then\ \ (f \frown (\ swhile\ \ (init\ w)\ \ do\ \ f))\ \ else\ \ \ empty$

by (*rule SWhileEqvIf*)
hence *2*: $\vdash$ (*swhile* (*init w*) *do* *f*)⌢ *g* =
if$_i$ (*init w*) *then* ((*f*⌢ *swhile* (*init w*) *do* *f*)⌢ *g*) *else* (*empty* ⌢ *g*)
by (*rule IfSChopEqvRule*)
have *3*: $\vdash$ *empty* ⌢ *g* = *g*
by (*rule EmptySChop*)
have *4*: $\vdash$ *if*$_i$ (*init w*) *then* ((*f*⌢ *swhile* (*init w*) *do* *f*)⌢ *g*) *else* (*empty* ⌢ *g*) =
if$_i$ (*init w*) *then* ((*f*⌢ *swhile* (*init w*) *do* *f*)⌢ *g*) *else* *g*
using *3* **using** *inteq-reflection* **by** *fastforce*
have *5*: $\vdash$ ((*f*⌢ *swhile* (*init w*) *do* *f*)⌢ *g*) = (*f*⌢ (*swhile* (*init w*) *do* *f*⌢ *g*))
by (*rule SChopAssocB*)
have *6*: $\vdash$ *if*$_i$ (*init w*) *then* ((*f*⌢ *swhile* (*init w*) *do* *f*)⌢ *g*) *else* *g* =
if$_i$ (*init w*) *then* (*f*⌢ ((*swhile* (*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 *SWhileSChopEqvIfRule*:
assumes $\vdash$ *f* = (*swhile* (*init w*) *do* *g*)⌢ *h*
shows $\vdash$ *f* = *if*$_i$ (*init w*) *then* (*g*⌢ *f*) *else* *h*
proof −
have *1*: $\vdash$ *f* = (*swhile* (*init w*) *do* *g*)⌢ *h*
using *assms* **by** *auto*
have *2*: $\vdash$ (*swhile* (*init w*) *do* *g*)⌢ *h* =
if$_i$ (*init w*) *then* (*g*⌢ ((*swhile* (*init w*) *do* *g*)⌢ *h*)) *else* *h*
by (*rule SWhileSChopEqvIf*)
have *3*: $\vdash$ (*g*⌢ *f*) = (*g*⌢ ((*swhile* (*init w*) *do* *g*)⌢ *h*))
using *1* **by** (*rule RightSChopEqvSChop*)
have *4*: $\vdash$ (*g*⌢ ((*swhile* (*init w*) *do* *g*)⌢ *h*)) = (*g*⌢ *f*)
using *3* **by** *auto*
have *5*: $\vdash$ *if*$_i$ (*init w*) *then* (*g*⌢ ((*swhile* (*init w*) *do* *g*)⌢ *h*)) *else* *h* =
if$_i$ (*init w*) *then* (*g*⌢ *f*) *else* *h*
using *4* **using** *inteq-reflection* **by** *fastforce*
from *1 2 5* **show** *?thesis* **by** *fastforce*
qed

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

lemma *SWhileEqvEmptyOrSChopSWhile*:
$\vdash$ *swhile* (*init w*) *do f* = (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$ (*init w* $\wedge$ (*f* $\wedge$ *more*)⌢*swhile* (*init w*) *do f*))
proof −
have *1*: $\vdash$ *schopstar* (*init w* $\wedge$ *f*) = (*empty* $\vee$ ((*init w* $\wedge$ *f*)$\wedge$ *more*)⌢ *schopstar* (*init w* $\wedge$ *f*))
by (*rule SChopstarEqv*)
have *2*: $\vdash$ ((*init w* $\wedge$ *f*) $\wedge$ *more*) = (*init w* $\wedge$ (*f* $\wedge$ *more*))
by *auto*

hence *3*: ⊢ ((*init w* ∧ *f*)∧ *more*)⁀ *schopstar* (*init w* ∧ *f*) =
(*init w* ∧ *f* ∧ *more*)⁀ *schopstar* (*init w* ∧ *f*)
by (*rule LeftSChopEqvSChop*)
have *4*: ⊢ *schopstar* (*init w* ∧ *f*) = (*empty* ∨ (*init w* ∧ *f* ∧ *more*)⁀ *schopstar* (*init w* ∧ *f*))
using *1 3* **by** *fastforce*
have *5*: ⊢ (*schopstar* (*init w* ∧ *f*) ∧ *sfin* (¬ (*init w*))) =
((*empty* ∧ *sfin* (¬ (*init w*))) ∨
((*init w* ∧ *f* ∧ *more*)⁀ *schopstar* (*init w* ∧ *f*)∧ *sfin* (¬ (*init w*))))
using *1 4* **by** *fastforce*
have *6*: ⊢ (*empty* ∧ *sfin* (¬ (*init w*))) = (¬ (*init w*) ∧ *empty*)
by (*meson Prop04 SFinAndEmpty lift-and-com*)
have *7*: ⊢ (*init w* ∧ *f* ∧ *more*)⁀ *schopstar* (*init w* ∧ *f*) =
(*init w* ∧ (*f* ∧ *more*)⁀ *schopstar* (*init w* ∧ *f*))
by (*rule StateAndSChop*)
have *8*: ⊢ (((*f* ∧ *more*)⁀ *schopstar* (*init w* ∧ *f*)) ∧ *sfin* (*init* (¬ *w*))) =
((*f* ∧ *more*)⁀ (*schopstar* (*init w* ∧ *f*) ∧ *sfin* (*init* (¬ *w*))))
by (*rule SChopAndSFin*)
have *81*: ⊢ *sfin* (*init* (¬ *w*)) = *sfin* (¬ (*init w*))
by (*metis Initprop*(*2*) *SFinStateEqvStateAndEmptyOrNextSFinState inteq-reflection*)
have *82*: ⊢ ((*f* ∧ *more*)⁀ *schopstar* (*init w* ∧ *f*) ∧ *sfin* (¬ (*init w*))) =
((*f* ∧ *more*)⁀ (*schopstar* (*init w* ∧ *f*) ∧ *sfin* (¬ (*init w*))))
using *8 81*
by (*metis inteq-reflection*)
have *9*: ⊢ (*schopstar* (*init w* ∧ *f*) ∧ *sfin* (¬ (*init w*))) =
((¬ (*init w*) ∧ *empty*) ∨
(*init w* ∧ (*f* ∧ *more*)⁀ (*schopstar* (*init w* ∧ *f*) ∧ *sfin* (¬ (*init w*)))))
using *5 6 7 82* **by** *fastforce*
from *9* **show** *?thesis* **by** (*simp add*: *swhile-d-def*)
qed

lemma *SWhileIntro*:
assumes ⊢ ¬ (*init w*) ∧ *f* ⟶ *empty*
⊢ *init w* ∧ *f* ⟶ (*g* ∧ *more*)⁀ *f*
shows ⊢ *f* ∧ *finite* ⟶ *swhile* (*init w*) *do* *g*
proof −
have *1*: ⊢ ¬ (*init w*) ∧ *f* ⟶ *empty*
using *assms* **by** *blast*
have *2*: ⊢ *init w* ∧ *f* ⟶ (*g* ∧ *more*)⁀ *f*
using *assms* **by** *blast*
have *3*: ⊢ *swhile* (*init w*) *do* *g* =
((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*)⁀ *swhile* (*init w*) *do* *g*))
by (*rule SWhileEqvEmptyOrSChopSWhile*)
hence *31*: ⊢ ¬ (*swhile* (*init w*) *do* *g*) =
(¬((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*)⁀ *swhile* (*init w*) *do* *g*)))
by *fastforce*
hence *32*: ⊢ (*f* ∧ ¬ (*swhile* (*init w*) *do* *g*)) =
(*f* ∧ ¬((¬(*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*)⁀ *swhile* (*init w*) *do* *g*)))
by *fastforce*
have *33*: ⊢ (*f* ∧ ¬((¬(*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*)⁀ *swhile* (*init w*) *do* *g*))) =
(*f* ∧ ¬(¬(*init w*) ∧ *empty*) ∧ ¬(*init w* ∧ (*g* ∧ *more*)⁀ *swhile* (*init w*) *do* *g*))

by *auto*
have *34*: $\vdash (f \wedge \neg(\neg(init\ w) \wedge empty) \wedge \neg((init\ w) \wedge ((g \wedge\ more) \frown swhile\ (init\ w)\ do\ g))) =$
$(f \wedge ((init\ w) \vee more) \wedge (\neg(init\ w) \vee \neg((g \wedge more) \frown swhile\ (init\ w)\ do\ g)))$
by (*auto simp*: *empty-d-def*)
have *35*: $\vdash (f \wedge ((init\ w) \vee more) \wedge (\neg(init\ w) \vee \neg((g \wedge more) \frown swhile\ (init\ w)\ do\ g))) =$
$((f \wedge (init\ w) \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)) \vee$
$(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
$(f \wedge more \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)) \vee$
$(f \wedge more \wedge \neg(init\ w)))$
by *auto*
have *36*: $\vdash (f \wedge \neg (swhile\ (init\ w)\ do\ g)) =$
$((f \wedge (init\ w) \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)) \vee$
$(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
$(f \wedge more \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)) \vee$
$(f \wedge more \wedge \neg(init\ w)))$ **using** *32 33 34 35* **by** *fastforce*
have *37*: $\vdash \neg(f \wedge more \wedge \neg(init\ w))$
using *1* **by** (*auto simp*: *empty-d-def*)
have *38*: $\vdash (f \wedge more \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)) \longrightarrow$
$((g \wedge\ more) \frown f \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g))$
using *1 2* **by** (*auto simp*: *empty-d-def Valid-def*)
have *39*: $\vdash (f \wedge (init\ w) \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)) \longrightarrow$
$((g \wedge\ more) \frown f \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g))$
using *2* **by** *auto*
have *40*: $\vdash ((f \wedge (init\ w) \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)) \vee$
$(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
$(f \wedge more \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)) \vee$
$(f \wedge more \wedge \neg(init\ w))) \longrightarrow$
$(g \wedge\ more) \frown f \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)$
using *39 38 37 38* **by** *fastforce*
have *4*: $\vdash f \wedge \neg (swhile\ (init\ w)\ do\ g) \longrightarrow$
$(g \wedge\ more) \frown f \wedge \neg ((g \wedge\ more) \frown\ swhile\ (init\ w)\ do\ g)$
using *36 40* **by** *fastforce*
have *5*: $\vdash g \wedge\ more \longrightarrow more$
by *auto*
from *4 5* **show** *?thesis* **using** *SChopContraB* **by** *blast*
qed

lemma *SWhileElim*:
assumes $\vdash \neg (init\ w) \wedge\ empty \longrightarrow g$
$\vdash init\ w \wedge (f \wedge\ more) \frown g \longrightarrow g$
shows $\vdash swhile\ (init\ w)\ do\ f \longrightarrow g$
proof –
have *1*: $\vdash swhile\ (init\ w)\ do\ f =$
$((\neg (init\ w) \wedge\ empty) \vee (init\ w \wedge (f \wedge\ more) \frown\ swhile\ (init\ w)\ do\ f))$
by (*rule SWhileEqvEmptyOrSChopSWhile*)
hence *11*: $\vdash ((swhile\ (init\ w)\ do\ f) \wedge \neg g) =$
$(((\neg(init\ w) \wedge empty) \vee (init\ w \wedge (f \wedge more) \frown swhile\ (init\ w)\ do\ f)) \wedge \neg g)$
by *auto*
have *2*: $\vdash \neg (init\ w) \wedge\ empty \longrightarrow g$
using *assms* **by** *blast*

hence *21*: ⊢ ¬ *g* ⟶ ¬(¬ (*init w*) ∧ *empty*)
by *auto*
have *22*: ⊢ ((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*f* ∧ *more*)⌢*swhile* (*init w*) *do f*)) ∧ ¬ *g* ⟶
(*init w* ∧ (*f* ∧ *more*)⌢ *swhile* (*init w*) *do f*)
using *21* **by** *auto*
have *23*: ⊢ (*swhile* (*init w*) *do f*) ∧ ¬ *g* ⟶
(*init w* ∧ (*f* ∧ *more*)⌢ *swhile* (*init w*) *do f*) ∧ ¬ *g*
using *11 21* **by** *fastforce*
have *3*: ⊢ (*init w*) ∧ ((*f* ∧ *more*)⌢ *g*) ⟶ *g*
using *assms* **by** *blast*
hence *31*: ⊢ ¬ *g* ⟶ ¬((*init w*) ∧ ((*f* ∧ *more*)⌢ *g*))
by *fastforce*
have *32*: ⊢ (*init w* ∧ (*f* ∧ *more*)⌢ *swhile* (*init w*) *do f*) ∧ ¬ *g* ⟶
(((*f* ∧ *more*)⌢ (*swhile* (*init w*) *do f*)) ∧ ¬ ((*f* ∧ *more*)⌢ *g*)) ∧ ¬*g*
using *31* **by** *auto*
have *4*: ⊢ (*swhile* (*init w*) *do f*) ∧ ¬ *g* ⟶
((*f* ∧ *more*)⌢ (*swhile* (*init w*) *do f*)) ∧ ¬ ((*f* ∧ *more*)⌢ *g*)
using *23 32* **by** *fastforce*
have *5*: ⊢ *f* ∧ *more* ⟶ *more*
by *auto*
have *6*: ⊢ (*swhile* (*init w*) *do f*) ∧ *finite* ⟶ *g*
using *ChopContraB 4 5* **using** *SChopContraB* **by** *blast*
have *7*: ⊢ ((*swhile* (*init w*) *do f*) ∧ *finite*) = (*swhile* (*init w*) *do f*)
using *swhile-d-def*
by (*metis 1 DiamondEmptyEqvFinite Prop10 Prop11 Prop12 SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond lift-imp-trans*)
from *6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaSWhileImpSWhile*:
⊢ *sba* (*f* ⟶ *g*) ⟶ (*swhile* (*init w*) *do f*) ⟶ (*swhile* (*init w*) *do g*)
proof −
have *1*: ⊢ (*f* ⟶ *g*) ⟶ ((*init w* ∧ *f*) ⟶ (*init w* ∧ *g*))
by *auto*
hence *2*: ⊢ *sba* (*f* ⟶ *g*) ⟶ *sba* ((*init w* ∧ *f*) ⟶ (*init w*∧ *g*))
by (*rule SBaImpSBa*)
have *3*: ⊢ *sba* ((*init w* ∧ *f*) ⟶ (*init w*∧ *g*)) ⟶
(*schopstar* (*init w*∧ *f*) ⟶ *schopstar* (*init w*∧ *g*))
by (*rule SBaSCSImpSCS*)
have *4*: ⊢ *sba* (*f* ⟶ *g*) ⟶ (*schopstar* (*init w* ∧ *f*)∧ *sfin* (¬ (*init w*)) ⟶
schopstar (*init w*∧ *g*) ∧ *sfin* (¬ (*init w*)))
using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *swhile-d-def*)
qed

lemma *SWhileImpSWhile*:
assumes ⊢ *f* ⟶ *g*
shows ⊢ (*swhile* (*init w*) *do f*) ⟶ (*swhile* (*init w*) *do g*)
proof −
have *1*: ⊢ *f* ⟶ *g*

using *assms* **by** *auto*
hence *2*: $\vdash$ *sba* ($f \longrightarrow g$)
by (*rule SBaGen*)
have *3*: $\vdash$ *sba* ($f \longrightarrow g$) $\longrightarrow$ (*swhile* (*init w*) *do* *f*) $\longrightarrow$ (*swhile* (*init w*) *do* *g*)
by (*rule SBaSWhileImpSWhile*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

14.10 Properties of Halt

lemma *HaltSChopEqv*:
$\vdash$ ((*halt* (*init w*)) $\frown$ *f*) = (*if*$_i$ (*init w*) *then* (*f*) *else* ($\bigcirc$((*halt* (*init w*))$\frown$ *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*))$\frown$*f*) =
(*if*$_i$ (*init w*) *then* (*empty*$\frown$*f*) *else* ($\bigcirc$(*halt* (*init w*))$\frown$*f*))
by (*rule IfSChopEqvRule*)
have *3*: $\vdash$ *empty* $\frown$ *f* = *f*
by (*rule EmptySChop*)
have *4*: $\vdash$ ($\bigcirc$ (*halt* (*init w*)))$\frown$ *f* = $\bigcirc$(*halt* (*init w*)$\frown$ *f*)
by (*rule NextSChop*)
from *2 3 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *AndHaltSChopImp*:
$\vdash$ *init w* $\wedge$ (*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ *f*
proof –
have *1*: $\vdash$ *halt* (*init w*)$\frown$ *f* = *if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*))
by (*rule HaltSChopEqv*)
have *2*: $\vdash$ *init w* $\wedge$ *if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$ *f*
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotAndHaltSChopImpNext*:
$\vdash$ $\neg$ (*init w*) $\wedge$ (*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ $\bigcirc$(*halt* (*init w*)$\frown$ *f*)
proof –
have *1*: $\vdash$ *halt* (*init w*)$\frown$ *f* = *if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*))
by (*rule HaltSChopEqv*)
have *2*: $\vdash$ $\neg$ (*init w*) $\wedge$ *if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$
$\bigcirc$(*halt* (*init w*)$\frown$ *f*)
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotAndHaltSChopImpSkipSYields*:
$\vdash$ $\neg$ (*init w*) $\wedge$ (*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ *skip* *syields* (*halt* (*init w*)$\frown$ *f*)
proof –

have *1*: ⊢ ¬ (*init w*) ∧ (*halt* (*init w*)⌢ *f*) ⟶ ○(*halt* (*init w*)⌢ *f*)
 by (*rule NotAndHaltSChopImpNext*)
have *2*: ⊢ ○(*halt* (*init w*)⌢ *f*) ⟶ *skip* *syields* (*halt* (*init w*)⌢ *f*)
 by (*rule NextImpSkipSYields*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndEmptyEqvSChopAndEmpty*:
⊢ ((#*True*⌢(*f* ∧ *empty*)) ∧ *g*) = (*g*⌢(*f*∧ *empty*))
proof −
have *1*: ⊢ (#*True*⌢(*f* ∧ *empty*)) ∧ *g* ⟶ *g* ⌢(*f* ∧ *empty*)
 by (*metis* (*no-types*, *lifting*) *AndSFinEqvSChopAndEmpty Prop12 int-iffD2 inteq-reflection lift-and-com*)
have *2*: ⊢ *g*⌢(*f*∧ *empty*) ⟶ (#*True*⌢(*f* ∧ *empty*)) ∧ *g*
 by (*metis AndSFinEqvSChopAndEmpty Prop12 SFinEqvTrueSChopAndEmpty int-iffD1 inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotSChopSkipEqvFmoreAndNotSChopSkip*:
⊢ (¬ *f*)⌢*skip* = (*fmore* ∧ ¬(*f*⌢*skip*))
proof −
have *1*: ⊢ (¬ *f*)⌢*skip* = ((¬*f* ∧ *finite*);*skip*)
by (*simp add*: *schop-d-def*)
have *2*: ⊢ (¬*f* ∧ *finite*);*skip* = (¬(*f* ∨ *inf*));*skip*
by (*metis* (*no-types*, *lifting*) *LeftChopEqvChop finite-d-def int-simps*(*14*) *int-simps*(*33*) *inteq-reflection*)
have *3*: ⊢ (¬(*f* ∨ *inf*));*skip* = (*more* ∧ ¬((*f* ∨ *inf*);*skip*))
 using *NotChopSkipEqvMoreAndNotChopSkip* **by** *blast*
have *4*: ⊢ (*f* ∨ *inf*);*skip* = (*f*;*skip* ∨ *inf*)
 by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv inteq-reflection*)
have *5*: ⊢ (*more* ∧ ¬((*f* ∨ *inf*);*skip*)) = (*more* ∧ ¬(*f*;*skip* ∨ *inf*))
 using *4* **by** *auto*
have *6*: ⊢ (*more* ∧ ¬(*f*;*skip* ∨ *inf*)) = (*more* ∧ ¬(*f*;*skip*) ∧ *finite*)
 using *finite-d-def*
 by (*metis 3 4 int-simps*(*14*) *int-simps*(*33*) *inteq-reflection*)
have *7*: ⊢ (*more* ∧ ¬(*f*;*skip*) ∧ *finite*) = (*more* ∧ ¬(*f*⌢*skip* ∨ (*f* ∧ *inf*)) ∧ *finite*)
 using *ChopSChopdef* **by** *fastforce*
have *8*: ⊢ (*more* ∧ ¬(*f*⌢*skip* ∨ (*f* ∧ *inf*)) ∧ *finite*) =
 (*more* ∧ ¬(*f*⌢*skip*) ∧ ¬(*f* ∧ *inf*) ∧ *finite*)
by *auto*
have *9*: ⊢ (¬(*f* ∧ *inf*) ∧ *finite*) = *finite*
using *finite-d-def*
by (*metis* (*no-types*, *lifting*) *AndInfChopAndInfEqvAndInf AndInfEqvChopFalse ChopAndB ChopLoopB FiniteChopMoreEqvMore NotEmptyEqvMore Prop10 RightChopImpMoreRule int-simps*(*21*) *inteq-reflection lift-and-com*)
have *10*: ⊢ (*more* ∧ ¬(*f*⌢*skip*) ∧ ¬(*f* ∧ *inf*) ∧ *finite*) =
 (*more* ∧ ¬(*f*⌢*skip*) ∧ *finite*)
using *9* **by** *fastforce*
have *11*: ⊢ (*more* ∧ ¬(*f*⌢*skip*) ∧ *finite*) = (*fmore* ∧ ¬(*f*⌢*skip*))
using *fmore-d-def*

by (*metis Prop11 Prop12 lift-and-com*)
from *1 2 3 5 6 7 8 10 11* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *HaltSChopImpNotHaltSChopNot*:
$\vdash\ \ halt\ (\ init\ w) \frown f \wedge finite \longrightarrow \neg\ (\ halt\ (\ init\ w) \frown (\neg\ f))$
proof −
have 1: $\vdash\ halt\ (init\ w) \frown f = if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(\ halt\ (\ init\ w) \frown f))$
by (*rule HaltSChopEqv*)
have 2: $\vdash if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(\ halt\ (\ init\ w) \frown f)) \longrightarrow$
$(\ ((init\ w) \longrightarrow f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown f))))$
by (*rule IfThenElseImp*)
have 3: $\vdash\ halt\ (init\ w) \frown (\neg f) =$
$if_i\ (init\ w)\ then\ (\neg f)\ else\ (\bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))$
by (*rule HaltSChopEqv*)
have 4: $\vdash if_i\ (init\ w)\ then\ (\neg f)\ else\ (\bigcirc(\ halt\ (\ init\ w) \frown (\neg f))) \longrightarrow$
$(\ ((init\ w) \longrightarrow \neg f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))))$
by (*rule IfThenElseImp*)
have 5: $\vdash halt\ (init\ w) \frown f \wedge halt\ (init\ w) \frown (\neg f) \longrightarrow$
$(\ ((init\ w) \longrightarrow f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown f)))) \wedge$
$(\ ((init\ w) \longrightarrow \neg f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))))$
using *1 2 3 4* **by** *fastforce*
have 6: $\vdash (\ ((init\ w) \longrightarrow f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown f)))) \wedge$
$(\ ((init\ w) \longrightarrow \neg f) \wedge (\ \neg(init\ w) \longrightarrow (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f))))) \longrightarrow$
$(\ \bigcirc(\ halt\ (\ init\ w) \frown f)) \wedge (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))$
by *auto*
have 7: $\vdash halt\ (init\ w) \frown f \wedge halt\ (init\ w) \frown (\neg f) \longrightarrow$
$(\ \bigcirc(\ halt\ (\ init\ w) \frown f)) \wedge (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))$
using *5 6 lift-imp-trans* **by** *blast*
have 8: $\vdash ((\ \bigcirc(\ halt\ (\ init\ w) \frown f)) \wedge (\ \bigcirc(\ halt\ (\ init\ w) \frown (\neg f)))) =$
$\bigcirc\ (halt\ (\ init\ w) \frown f \wedge halt\ (\ init\ w) \frown (\neg f))$
using *NextAndEqvNextAndNext* **by** *fastforce*
have 9: $\vdash halt\ (init\ w) \frown f \wedge halt\ (init\ w) \frown (\neg f) \longrightarrow$
$\bigcirc\ (halt\ (\ init\ w) \frown f \wedge halt\ (\ init\ w) \frown (\neg f))$
using *7 8* **by** *fastforce*
hence 10: $\vdash finite \longrightarrow \neg(halt\ (init\ w) \frown f \wedge halt\ (init\ w) \frown (\neg f))$
using *NextLoop* **by** *blast*
from *10* **show** *?thesis* **by** *auto*
qed

lemma *HaltSChopImpHaltSYields*:
$\vdash\ \ halt\ (\ init\ w) \frown f \wedge finite \longrightarrow (\ halt\ (\ init\ w))\ syields\ f$
proof −
have 1: $\vdash\ halt\ (\ init\ w) \frown f \wedge finite \longrightarrow \neg\ (\ halt\ (\ init\ w) \frown (\neg\ f))$
by (*rule HaltSChopImpNotHaltSChopNot*)
from *1* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *HaltSChopAnd*:
$\vdash (\ halt\ (init\ w)) \frown f \wedge (\ halt\ (init\ w)) \frown g \wedge finite \longrightarrow (\ halt\ (\ init\ w)) \frown (f \wedge g)$

proof −
have *1*: $\vdash$ (*halt* (*init w*))$\frown$ *g* $\wedge$ *finite* $\longrightarrow$ (*halt* (*init w*)) *syields* *g*
by (*rule HaltSChopImpHaltSYields*)
hence *2*: $\vdash$ (*halt* (*init w*))$\frown$ *f* $\wedge$ (*halt* (*init w*))$\frown$ *g* $\wedge$ *finite* $\longrightarrow$
(*halt* (*init w*))$\frown$ *f* $\wedge$ (*halt* (*init w*)) *syields* *g* **by** *auto*
have *3*: $\vdash$ (*halt* (*init w*))$\frown$ *f* $\wedge$ (*halt* (*init w*)) *syields* *g* $\longrightarrow$
(*halt* (*init w*))$\frown$ (*f* $\wedge$ *g*) **by** (*rule SChopAndSYieldsImp*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltAndSChopAndHaltSChopImpHaltAndSChopAnd*:
$\vdash$ (*halt* (*init w*) $\wedge$ *f*)$\frown$ *f1* $\wedge$ (*halt* (*init w*)$\frown$ *g*) $\wedge$ *finite*
$\longrightarrow$ (*halt* (*init w*) $\wedge$ *f*)$\frown$ (*f1* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *f1* $\longrightarrow$ $\neg$ *g* $\vee$ (*f1* $\wedge$ *g*)
by *auto*
hence *2*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*)$\frown$ *f1* $\longrightarrow$
(*halt* (*init w*) $\wedge$ *f*)$\frown$ ($\neg$ *g*) $\vee$ ((*halt* (*init w*) $\wedge$ *f*)$\frown$ (*f1* $\wedge$ *g*))
by (*rule SChopOrImpRule*)
have *3*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*)$\frown$ ($\neg$ *g*) $\longrightarrow$ *halt* (*init w*)$\frown$ ($\neg$ *g*)
by (*rule AndSChopA*)
have *31*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*)$\frown$ *f1* $\longrightarrow$
halt (*init w*)$\frown$ ($\neg$ *g*) $\vee$ ((*halt* (*init w*) $\wedge$ *f*)$\frown$ (*f1* $\wedge$ *g*))
using *23* **by** *fastforce*
have *4*: $\vdash$ *halt* (*init w*)$\frown$ *g* $\wedge$ *finite* $\longrightarrow$ $\neg$ (*halt* (*init w*)$\frown$ ($\neg$ *g*))
by (*rule HaltSChopImpNotHaltSChopNot*)
hence *41*: $\vdash$ (*halt* (*init w*)$\frown$ ($\neg$ *g*)) $\wedge$ *finite* $\longrightarrow$ $\neg$(*halt* (*init w*)$\frown$ *g*)
by *auto*
have *42*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*)$\frown$ *f1* $\wedge$ *finite* $\longrightarrow$
$\neg$(*halt* (*init w*)$\frown$ *g*) $\vee$ ((*halt* (*init w*) $\wedge$ *f*)$\frown$ (*f1* $\wedge$ *g*))
using *31 41* **by** *fastforce*
from *42* **show** *?thesis* **by** *auto*
qed

lemma *HaltImpBoxSYields*:
$\vdash$ (*halt* (*init w*))$\frown$ *f* $\wedge$ *finite* $\longrightarrow$ ($\Box$($\neg$ (*init w*))) *syields* ((*halt* (*init w*))$\frown$ *f*)
proof −
have *1*: $\vdash$ ($\Box$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$ *df* ($\Box$ ($\neg$ (*init w*)))
by (*rule SChopImpDf*)
have *2*: $\vdash$ $\Box$ ($\neg$ (*init w*)) $\longrightarrow$ $\neg$ (*init w*)
by (*rule BoxElim*)
hence *3*: $\vdash$ *df* ($\Box$ ($\neg$ (*init w*))) $\longrightarrow$ *df* ($\neg$ (*init w*))
by (*rule DfImpDf*)
have *4*: $\vdash$ *df* (*init* ($\neg$ *w*)) = (*init* ($\neg$*w*))
by (*rule DfState*)
have *41*: $\vdash$ (*init* ($\neg$ *w*)) = ($\neg$ (*init w*))
using *Initprop*(*2*) **by** *fastforce*
have *42*: $\vdash$ *df* ($\neg$ (*init w*)) = ($\neg$(*init w*))
using *4 41* **by** (*metis inteq-reflection*)
have *5*: $\vdash$ (($\Box$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*))) $\longrightarrow$ $\neg$ (*init w*)

using *1 2 42* **using** *3* **by** *fastforce*
hence *51*: $\vdash$ (*halt* (*init w*)$\frown$ *f*) $\wedge$ (($\square$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*))) $\longrightarrow$
(*halt* (*init w*)$\frown$ *f*) $\wedge$ $\neg$ (*init w*)
by *fastforce*
have *6*: $\vdash$ *halt* (*init w*)$\frown$ *f* = *if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*))
by (*rule HaltSChopEqv*)
hence *61*: $\vdash$ (*halt* (*init w*)$\frown$ *f* $\wedge$ $\neg$ (*init w*)) =
((*if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*))) $\wedge$ $\neg$ (*init w*))
using *6* **by** *auto*
have *62*: $\vdash$ (*if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*)$\frown$ *f*))) $\wedge$
$\neg$ (*init w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init w*)$\frown$ *f*))
by (*auto simp*: *ifthenelse-d-def*)
have *63*: $\vdash$ *halt* (*init w*)$\frown$ *f* $\wedge$ $\neg$ (*init w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init w*)$\frown$ *f*))
using *61 62* **by** *fastforce*
have *7*: $\vdash$ (*halt* (*init w*)$\frown$ *f*) $\wedge$ ($\square$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$
$\bigcirc$((*halt* (*init w*))$\frown$ *f*)
using *51 63* **using** *lift-imp-trans* **by** *blast*
have *8*: $\vdash$ $\square$ ($\neg$ (*init w*)) $\longrightarrow$ *empty* $\vee$ $\bigcirc$($\square$($\neg$(*init w*)))
using *BoxBoxImpBox BoxEqvAndEmptyOrNextBox* **by** *fastforce*
hence *9*: $\vdash$ (($\square$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*))) $\longrightarrow$
$\neg$ (*halt* (*init w*)$\frown$ *f*) $\vee$ $\bigcirc$(($\square$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
by (*rule EmptyOrNextSChopImpRule*)
hence *10*: $\vdash$ ((*halt* (*init w*))$\frown$ *f*) $\wedge$ ($\square$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$
$\bigcirc$(($\square$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
by *fastforce*
have *11*: $\vdash$ (*halt* (*init w*))$\frown$ *f* $\wedge$ ($\square$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$
$\bigcirc$((*halt* (*init w*))$\frown$ *f*) $\wedge$ $\bigcirc$(($\square$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
using *7 10* **by** *fastforce*
have *12*: $\vdash$ $\bigcirc$((*halt* (*init w*))$\frown$ *f*) $\wedge$ $\bigcirc$(($\square$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
$\longrightarrow$ $\bigcirc$(((*halt* (*init w*))$\frown$ *f*) $\wedge$ (($\square$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*))))
using *NextAndEqvNextAndNext* **by** *fastforce*
have *13*: $\vdash$ (*halt* (*init w*))$\frown$ *f* $\wedge$ ($\square$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$
$\bigcirc$(((*halt* (*init w*))$\frown$ *f*) $\wedge$ (($\square$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*))))
using *11 12* **by** *fastforce*
hence *14*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ((*halt* (*init w*))$\frown$ *f* $\wedge$ ($\square$ ($\neg$ (*init w*)))$\frown$ ($\neg$(*halt* (*init w*)$\frown$ *f*)))
using *NextLoop* **by** *blast*
hence *15*: $\vdash$ (*halt* (*init w*))$\frown$ *f* $\wedge$ *finite* $\longrightarrow$ $\neg$ (($\square$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
by *auto*
from *15* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

14.11 Properties of Groups of strong chops

lemma *NestedSChopImpSChop*:
assumes $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g* $\frown$ (*init w1* $\wedge$ *f1*)
$\vdash$ *init w1* $\wedge$ *f1* $\longrightarrow$ *g1*$\frown$ (*init w2* $\wedge$ *f2*)
shows $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g* $\frown$ (*g1*$\frown$ (*init w2* $\wedge$ *f2*))
proof –
have *1*: $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g*$\frown$ (*init w1* $\wedge$ *f1*) **using** *assms*(*1*) **by** *auto*
have *2*: $\vdash$ *init w1* $\wedge$ *f1* $\longrightarrow$ *g1*$\frown$ (*init w2* $\wedge$ *f2*) **using** *assms*(*2*) **by** *auto*

hence *3*: $\vdash g\frown (init\ w1 \wedge f1) \longrightarrow g\frown (g1\frown (init\ w2 \wedge f2))$ **by** (*rule RightSChopImpSChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

end

15 First Order Infinite ITL theorems

theory *InfiniteFOTheorems*
imports
InfiniteSChopTheorems
begin

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

lemma *EExI-unl*:
$w \models f\, x \Longrightarrow w \models (\exists\exists\ x.\ f\, x)$
using *EExValInfinite EExValFinite*
by (*meson exist-state-d-def*)

lemma *EExNoDep*:
$\vdash (\exists\exists\ x.\ g) = g$
proof −
have *1*: $\vdash g \longrightarrow (\exists\exists\ x.\ g)$ **by** (*meson EExI*)
have *2*: $\bigwedge x. \vdash g \longrightarrow g$ **by** *simp*
have *3*: $\vdash (\exists\exists\ x.\ g) \longrightarrow g$ **using** *2* **by** (*meson EExE*)
from *1 3* **show** *?thesis* **using** *int-iffI* **by** *blast*
qed

lemma *AAxNoDep*:
$\vdash (\forall\forall\ x.\ g) = g$
using *EExNoDep*[*of LIFT*($\neg$ *g*)] *AAxDef EExE EExI*
by (*simp add*: *exist-state-d-def forall-state-d-def intI*)

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

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

lemma *EExImpRule*:
assumes $\vdash f\, x \longrightarrow g\, x$
shows $\vdash (\exists\exists\ x.\ f\, x \longrightarrow g\, x)$

using *assms* **by** (*meson MP EExI*)

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

lemma *EExImpNoDepDist*:
assumes $\vdash f \longrightarrow g\,x$
shows $\vdash f \longrightarrow (\exists\exists\ x.\ g\,x)$
using *assms* **by** (*metis EExI lift-imp-trans*)

lemma *EExOrDist-1*:
$\vdash (\exists\exists\ x.\ h\,x) \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$
proof −
have *1*: $\bigwedge x. \vdash h\,x \longrightarrow f\,x \vee h\,x$ **by** (*simp add*: *Valid-def*)
have *2*: $\bigwedge x. \vdash f\,x \vee h\,x \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$ **by** (*meson EExI*)
have *3*: $\bigwedge x. \vdash h\,x \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$ **using** *1 2* **by** (*meson lift-imp-trans*)
from *3* **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-2*:
$\vdash (\exists\exists\ x.\ f\,x) \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$
proof −
have *1*: $\bigwedge x. \vdash f\,x \longrightarrow f\,x \vee h\,x$ **by** (*simp add*: *Valid-def*)
have *2*: $\bigwedge x. \vdash f\,x \vee h\,x \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$ **by** (*meson EExI*)
have *3*: $\bigwedge x. \vdash f\,x \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$ **using** *1 2* **by** (*meson lift-imp-trans*)
from *3* **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-3*:
$\vdash (\exists\exists\ x.\ f\,x) \vee (\exists\exists\ x.\ h\,x) \longrightarrow (\exists\exists\ x.\ (f\,x) \vee (h\,x))$
using *EExOrDist-2 EExOrDist-1* **by** *fastforce*

lemma *EExOrDist-4*:
$\vdash (\exists\exists\ x.\ (f\,x) \vee (h\,x)) \longrightarrow (\exists\exists\ x.\ f\,x) \vee (\exists\exists\ x.\ h\,x)$
proof −
have *1*: $\bigwedge x. \vdash (f\,x) \vee (h\,x) \longrightarrow (\exists\exists\ x.\ f\,x) \vee (\exists\exists\ x.\ h\,x)$
by (*simp add*: *EExI-unl intI*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExOrDist*:
$\vdash ((\exists\exists\ x.\ f\,x) \vee (\exists\exists\ x.\ h\,x)) = (\exists\exists\ x.\ (f\,x) \vee (h\,x))$

using *EExOrDist-3 EExOrDist-4* **by** *fastforce*

lemma *EExOrImport-1*:
$\vdash g \longrightarrow (\exists\exists\ x.\ g \vee (f\,x))$
by (*simp add*: *EExI-unl Valid-def*)

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

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

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

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

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

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

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

lemma *EExAndDist*:
assumes $\vdash f\,x \wedge g\,x$

shows $\vdash (\exists\exists\ x.\ f\,x) \wedge (\exists\exists\ x.\ g\,x)$
proof −
have *1*: $\vdash f\,x$ **using** *assms* **by** *fastforce*
have *2*: $\vdash g\,x$ **using** *assms* **by** *fastforce*
have *3*: $\vdash (\exists\exists\ x.\ f\,x)$ **using** *1* **by** (*meson EExI MP*)
have *4*: $\vdash (\exists\exists\ x.\ g\,x)$ **using** *2* **by** (*meson EExI MP*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

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

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

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

lemma *AAxI*:
assumes $\bigwedge x.\ \vdash f\,x$
shows $\vdash (\forall\forall\ x.\ f\,x)$
using *assms* **by** (*simp add*: *Valid-def exist-state-d-def forall-state-d-def*)

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

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

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

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

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

lemma *EExImpChopRule*:
assumes $\vdash f\ x \longrightarrow g\ x$
shows $\vdash (\exists\exists\ x.\ h;(f\ x) \longrightarrow h;(g\ x))$
using *RightChopImpChop*[*of f x g x h*]
EExImpRule[*of* $\lambda x.\ LIFT(h;(f\ x))\ x\ \lambda x.\ LIFT(h;(g\ x))$] *assms* **by** *auto*

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

lemma *EExChopRightNoDep*:
$\vdash (\exists\exists\ x.\ (f\, x);g) = (\exists\exists\ x.\ (f\, x));g$
by (*auto simp add*: *exist-state-d-def Valid-def chop-defs sum.case-eq-if*)

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

lemma *EExChopLeftNoDep*:
$\vdash (\exists\exists\ x.\ g;(f\, x)\ ) = g;(\exists\exists\ x.\ f\, x)$
by (*auto simp add*: *exist-state-d-def Valid-def chop-defs sum.case-eq-if*)

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

lemma *EExEExChopEqvEExChopEExA*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (f\, v);(g\, y)\ )) = (\exists\exists\ v.\ (f\, v);(\exists\exists\ y.\ (g\, y)\ )\ )$
by (*simp add*: *exist-state-d-def Valid-def chop-defs sum.case-eq-if*) *blast*

lemma *EExEExChopEqvEExChopEExB*:
$\vdash (\exists\exists\ y.\ (\exists\exists\ v.\ (f\, v);(g\, y)\ )) = (\exists\exists\ y.\ (\exists\exists\ v.\ (f\, v));\ (g\, y))$
by (*simp add*: *exist-state-d-def Valid-def chop-defs sum.case-eq-if*) *blast*

lemma *EExEExChopEqvEExChopEExC*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (f\, v);(g\, y)\ )) = (\exists\exists\ v.\ (f\, v));(\exists\exists\ y.\ (g\, y))$
by (*simp add*: *exist-state-d-def Valid-def chop-defs sum.case-eq-if*) *blast*

lemma *ExLenOrInf*:
$\vdash (\exists\ n.\ len(n)) \lor inf$
by (*simp add*: *Valid-def len-defs sum.case-eq-if infinite-defs*)

lemma *CSPowerChop*:
$\vdash (f^{\star}) = (\exists n.\ power\ (f \land more)\ n);(empty \lor (f \land more) \land inf)$
by (*simp add*: *chopstar-d-def powerstar-d-def Valid-def sum.case-eq-if*)

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

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

lemma *ExExEqvExEx*:
$\vdash (\exists\ x.\ (\exists\ y.\ (f\, x);(g\, y))) = (\exists\ y.\ (\exists\ x.\ (f\, x);(g\, y)))$

by (*auto simp add*: *Valid-def chop-defs*)

end

16 The First Occurrence Operator in finite ITL

theory *First*
imports
Theorems TimeReversal
begin

Runtime verification (RV) has gained significant interest in recent years. The behaviour of a program can be verified in real time by analysing its evolving trace. This approach has two significant benefits over static verification techniques such as model checking. Firstly, it is only necessary to verify actual execution paths rather than all possible paths. Secondly, it is possible to react at runtime should the program diverge from its specified behaviour. RV does not replace traditional verification techniques but it does provide an extra layer of security.

Linear Temporal Logic (LTL) is a popular formalism for writing specifications from which RV monitors can be derived automatically. By contrast, Interval Temporal Logic (ITL) has not been as widely represented in this field despite being more expressive and compositional. The principal issue is efficiency. ITL uses non-deterministic operators to construct sequential and iterative specifications (chop and chop-star, respectively) and these introduce combinatorial complexity. Approaches to mitigate this include using a deterministic subset of ITL or adapting the semantics to include a deterministic chop operator. This work proposes an alternative approach, wholly within existing ITL, and based upon a new, derived operator called "first occurrence".

A theory of first occurrence is developed and used to derive an algebra of RV monitors.

16.1 Definitions

16.1.1 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*

16.1.2 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*

16.2 First and Time Reversal

lemma *BsEqvRule*:

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

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

lemma *FstEqvRule*:
assumes $\vdash f = g$
shows $\vdash \triangleright f = \triangleright g$
proof $-$
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence *3*: $\vdash bs(\neg f) = bs(\neg g)$ **by** (*simp add*: *BsEqvRule*)
hence *4*: $\vdash (f \land bs(\neg f)) = (g \land bs(\neg g))$ **using** *1* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*:*first-d-def*)
qed

lemma *LstEqvRule*:
assumes $\vdash f = g$
shows $\vdash \triangleleft f = \triangleleft g$
proof $-$
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence *3*: $\vdash bt(\neg f) = bt(\neg g)$ **by** (*simp add*: *BtEqvRule*)
hence *4*: $\vdash (f \land bt(\neg f)) = (g \land bt(\neg g))$ **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*: $\vdash (bs\ f)^r = (empty \lor ((bi\ f)\ ;\ skip))^r$
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*:
$\vdash$ ($\rhd$ *f*)^r = ($\lhd$ (f^r))
proof −
have *1*: $\vdash$ ($\rhd$ *f*)^r = (*f* $\wedge$ *bs*($\neg$ *f*))^r **by** (*simp add*: *first-d-def*)
have *2*: $\vdash$ (*f* $\wedge$ *bs*($\neg$ *f*))^r = (f^r $\wedge$ (*bs* ($\neg$ *f*))^r) **using** *RAnd* **by** *blast*
have *3*: $\vdash$ (f^r $\wedge$ (*bs* ($\neg$ *f*))^r) = (f^r $\wedge$ *bt* (($\neg$ *f*)^r)) **using** *RBsEqvBt* **by** *fastforce*
have *4*: $\vdash$ (f^r $\wedge$ *bt* (($\neg$ *f*)^r)) = (f^r $\wedge$ *bt* ($\neg$(f^r))) **using** *RNot int-eq* **by** *fastforce*

have *5*: $\vdash (f^r \wedge bt\ (\neg(f^r))) = (\lhd\ (f^r))$ **by** (*simp add*: *last-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

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

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

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

16.3 Semantic Theorems

16.3.1 Semantics First and Last Operators

lemma *FstAndBisem*:
$(intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\ \sigma \models\ bi\ (\neg f);skip)) =$
$(intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall\ ia<intlen\ (\sigma).\ (prefix\ ia\ \ \sigma \models \neg f))\ )$
proof –
have $(intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\ \sigma \models\ bi\ (\neg f);skip)) =$
$(0 < intlen\ \sigma \wedge (\sigma \models f) \wedge$
$(\exists\ i.\ (i \le intlen\ \sigma \longrightarrow (\forall\ ia \le i.\ \neg\ (prefix\ ia\ (prefix\ i\ \sigma) \models f)) \wedge$
$intlen\ \sigma - i = Suc\ 0) \wedge i \le intlen\ \sigma)$
$)$
using *le-trans* **by** (*auto simp add*: *chop-defs bi-defs skip-defs*, *blast*)
also have ... =
$(0 < intlen\ \sigma \wedge (\sigma \models f) \wedge$
$(\exists\ i.\ (i \le intlen\ \sigma \longrightarrow (\forall\ ia \le i.\ \neg\ (prefix\ ia\ (prefix\ i\ \sigma) \models f)) \wedge$
$i = intlen\ \sigma - Suc\ 0) \wedge i \le intlen\ \sigma)$
$)$
by *auto*

also have ... =
$(0 < intlen\ \sigma \wedge (\sigma \models f) \wedge$
$(\forall\, ia \leq (intlen\ \sigma - Suc\ 0).\ \neg\ (prefix\ ia\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma) \models f)\ )$
$)$
using *diff-le-self* **by** *blast*
also have ... =
$(intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge$
$(\forall\, ia < intlen\ (\ \sigma).\ \neg\ (prefix\ ia\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma) \models f))$
$)$ **by** (*metis Suc-pred less-Suc-eq-le*)
also have ... =
$(intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge$
$(\forall\, ia < intlen\ (\ \sigma).\ (prefix\ ia\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma) \models \neg f))$
$)$
by *auto*
also have ... =
$(intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall\, ia < intlen\ (\sigma).\ (prefix\ ia\ \ \sigma \models \neg f))\ )$
by (*simp add*: *interval-pref-pref-help*)
finally show $(intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\ \sigma \models\ \ bi\ (\neg f);skip)) =$
$(intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall\, ia < intlen\ (\sigma).\ (prefix\ ia\ \ \sigma \models \neg f))\ )$ **.**
qed

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

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

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

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

proof −
have $(\sigma \models \triangleleft\ f) = (\sigma \models (\triangleright\ (f^r))^r)$
using *RRFirstEqvLast* **by** *fastforce*
also have ... $= (intrev\ \sigma \models \triangleright\ (f^r))$

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

16.3.2 Various Semantic Lemmas

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

lemma *PrefixFstsem*:
((*prefix i* $\sigma \models \rhd f$) $\wedge$ $i \leq$*intlen* σ) =
($i \leq$*intlen* σ $\wedge$
(
((*prefix i* $\sigma \models f$) $\wedge$ i = *0*) $\vee$
(i>*0* $\wedge$ (*prefix i* $\sigma \models f$) $\wedge$ ($\forall$ *ia*<i. (*prefix ia* $\sigma \models \neg f$)))

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

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

lemma *DiLenFstsem*:
$(\sigma \models di\ (\triangleright f\ \wedge len(i))) =$
($i\leq intlen\ \sigma \wedge$
(
($(prefix\ i\ \sigma \models f) \wedge i = 0) \vee$

$(\ i{>}0 \wedge (\mathit{prefix}\ i\ \sigma \models f) \wedge (\forall\, ia{<}i.\ (\mathit{prefix}\ ia\ \sigma \models \neg f)))$
$)$
$)$
by (*simp add*: *DiLensem PrefixFstsem*)

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

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

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

16.4 Theorems

16.4.1 Fixed length intervals

lemma *LenZeroEqvEmpty*:
$\vdash len(0) = empty$
by (*simp add*: *len-d-def*)

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

lemma *LenNPlusOneA*:
$\vdash\ len(n{+}1) = skip;len(n)$
by (*simp add*: *len-d-def*)

lemma *LenEqvLenChopLen*:
$\vdash len(i{+}j) = len(i);len(j)$

proof
(*induct i*)
case *0*
then show *?case*
by (*metis EmptyChop LenZeroEqvEmpty add.left-neutral inteq-reflection*)
next
case (*Suc i*)
then show *?case*
by (*metis ChopAssoc LenNPlusOneA add.commute add-Suc inteq-reflection plus-1-eq-Suc*)
qed

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

lemma *LenCommute*:
$\vdash$ (*skip*;(*len n*)) = (*len n*);*skip*
proof
(*induct n*)
case *0*
then show *?case*
by (*metis LenEqvLenChopLen LenNPlusOneA LenOneEqvSkip inteq-reflection*)
next
case (*Suc n*)
then show *?case*
by (*metis LenEqvLenChopLen LenNPlusOneA LenOneEqvSkip inteq-reflection*)
qed

lemma *SkipTrueEqvTrueSkip*:
$\vdash$ *skip*;#*True* = #*True*;*skip*
using *TrueChopSkipEqvSkipChopTrue* **by** *fastforce*

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

lemma *PowerRev*:
$\vdash$ (*power skip n*)^r = (*power skip n*)

proof
(*induct n*)
case *0*
then show *?case* **using** *REmptyEqvEmpty* **by** *auto*
next
case (*Suc n*)
then show *?case* **using** *PowerCommute RevChop pow-Suc* **by** (*metis RevSkip int-eq*)
qed

lemma *RLenEqvLen*:
$\vdash (len\ k)^r = (len\ k)$
proof
(*induct k*)
case *0*
then show *?case*
using *LenZeroEqvEmpty REmptyEqvEmpty inteq-reflection* **by** *force*
next
case (*Suc k*)
then show *?case*
by (*metis PowerRev len-d-def*)
qed

lemma *PowerSkipEqvLen*:
$\vdash (power\ skip\ n) = (len\ n)$
by (*simp add*: *len-d-def*)

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*:
$\vdash ((f0 \wedge len(k));g0 \wedge (f1 \wedge len(k));g1) = ((f0 \wedge f1) \wedge len(k));(g0 \wedge g1)$
by (*auto simp add*: *min.absorb1 Valid-def len-defs chop-defs*)

lemma *RFixedAndDistr*:
$\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g1 \wedge len(k))) = (f0 \wedge f1);((g0 \wedge g1) \wedge len(k))$
by (*simp add*: *Valid-def min.absorb1 len-defs chop-defs*) (*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 (*auto simp add*: *bi-defs Valid-def*)
hence *2*: $\vdash$ (*bi* (*f* $\wedge$ *g*));*skip* = (*bi f* $\wedge$ *bi g*);*skip*
    by (*rule LeftChopEqvChop*)
have *3*: $\vdash$ (*bi f* $\wedge$ *bi g*);*skip* = ((*bi f*);*skip* $\wedge$ (*bi g*);*skip*)
    using *ChopSkipAndChopSkip* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

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

lemma *ChopEmptyAndEmpty*:

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

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

lemma *NotChopFixed*:
$\vdash (\neg(f;(g \wedge len(k)))) = (\neg(\Diamond(g \wedge len(k))) \vee ((\neg f);(g \wedge len(k))))$
by (*auto simp add*: *len-defs Valid-def sometimes-defs chop-defs*)
(*metis diff-diff-cancel*)

lemma *NotFixedChop*:
$\vdash (\neg((g \wedge len(k));f)) = (\neg(di(g \wedge len(k))) \vee ((g \wedge len(k));(\neg f)))$
by (*auto simp add*: *len-defs min.absorb1 Valid-def di-defs chop-defs*)

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

16.4.2 Additional ITL theorems

lemma *BiOrBiImpBiOr*:
$\vdash bi\ f \vee bi\ g \longrightarrow bi(f \vee g)$

proof −
 have *1*: ⊢ *f* ⟶ *f* ∨ *g* **by** *auto*
 hence *2*: ⊢ *bi f* ⟶ *bi*(*f* ∨ *g*) **by** (*rule BiImpBiRule*)
 have *3*: ⊢ *g* ⟶ *f* ∨ *g* **by** *auto*
 hence *4*: ⊢ *bi g* ⟶ *bi*(*f* ∨ *g*) **by** (*rule BiImpBiRule*)
 from *2 4* **show** *?thesis* **by** *fastforce*
qed

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

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

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

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

lemma *BiImpBiBiOrB*:
⊢ *bi g* ⟶ *bi* (*bi f* ∨ *bi g*)
proof −
 have *1*: ⊢ *bi g* ⟶ *bi f* ∨ *bi g* **by** *auto*

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

lemma *DiEqvOrDiChopSkipA*:
$\vdash di\ f = (f \lor di(f;skip))$
proof –
have *1*: $\vdash di\ f = f\ ;\#True$ **by** (*simp add*: *di-d-def*)
hence *2*: $\vdash di\ f = f;\ (\ empty \lor more)$ **by** (*simp add*: *empty-d-def*)
hence *3*: $\vdash f;\ (\ empty \lor more) = (f;empty \lor f;more)$ **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*:
$\vdash di\ f = (f \lor (di\ f);skip)$
proof –
have *1*: $\vdash (di\ f) = (f \lor di(f;skip))$ **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*: $\vdash (f \lor di(f;skip)) = (f \lor (di\ f);skip)$ **by** *auto*
from *1 11* **show** *?thesis* **by** *fastforce*
qed

lemma *BiEqvAndEmptyOrBiChopSkip*:
$\vdash bi\ f = (f \land (empty \lor (bi\ f);skip))$
proof –
have *1*: $\vdash di\ (\neg\ f) = (\neg\ f \lor (di\ (\neg\ f);skip))$ **by** (*rule DiEqvOrDiChopSkipB*)
have *2*: $\vdash di\ (\neg\ f) = (\neg(\ bi\ f\ ))$ **by** (*rule DiNotEqvNotBi*)

have *3*: ⊢ $(\neg(\ bi\ f\ )) = \ (\neg\ f \vee (di\ (\neg\ f);skip))$ **using** *1 2* **by** *fastforce*
hence *4*: ⊢ $bi\ f = (\neg(\neg\ f \vee (di\ (\neg\ f);skip)))$ **by** *auto*
have *5*: ⊢ $(\neg(\neg\ f \vee (di\ (\neg\ f);skip))) = (f \wedge \neg(di\ (\neg\ f);skip))$ **by** *auto*
have *6*: ⊢ $di\ (\neg\ f);skip = ((\neg(bi\ f));skip)$ **by** (*simp add*: *2 LeftChopEqvChop*)
hence *7*: ⊢ $(\neg(di\ (\neg\ f);skip)) = (\neg((\neg(bi\ f));skip))$ **by** *auto*
have *8*: ⊢ $(\neg((\neg(bi\ f));skip)) = (empty \vee (bi\ f);skip)$ **using** *NotNotChopSkip* **by** *blast*
hence *9*: ⊢ $(f \wedge \neg(di\ (\neg\ f);skip)) = (f \wedge (empty \vee (bi\ f);skip))$ **using** *7 8* **by** *fastforce*
from *4 5 9* **show** *?thesis* **by** *fastforce*
qed

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

lemma *BiInduct*:
⊢ $bi(f \longrightarrow wprev\ f) \wedge f \longrightarrow bi\ f$
proof −
have *1*: ⊢ $\square((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \square(f^r)$ **using** *BoxInduct* **by** *blast*
hence *2*: ⊢ $(\square((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \square(f^r))^r$ **using** *ReverseEqv* **by** *blast*
have *3*: ⊢ $((f^r)^r\ ) = f$ **by** (*simp add*: *EqvReverseReverse*)
have *4*: ⊢ $(\square\ (f^r))^r = bi\ (f)$ **using** *RRBoxEqvBi* **by** *blast*
have *5*: ⊢ $((f^r) \longrightarrow wnext(f^r))^r = (\ (f^r)^r \longrightarrow (wnext(f^r))^r\ )$ **by** (*simp add*: *rev-fun2*)
have *6*: ⊢ $(wnext(f^r))^r = wprev(f)$ **using** *RRWNextEqvWPrev* **by** *blast*
have *7*: ⊢ $(\ (f^r)^r \longrightarrow (wnext(f^r))^r\ ) = (\ f \longrightarrow wprev(f)\ )$ **using** *6 3* **by** *fastforce*
have *8*: ⊢ $bi(\ (f^r)^r \longrightarrow (wnext(f^r))^r\ ) = bi(\ f \longrightarrow wprev(f)\ )$ **using** *7 3 BiEqvBi* **by** *blast*
have *9*: ⊢ $(\square((f^r) \longrightarrow wnext(f^r)))^r = bi\ (\ ((f^r) \longrightarrow wnext(f^r))^r\ )$ **using** *RBoxEqvBi* **by** *blast*
have *10*: ⊢ $(\square((f^r) \longrightarrow wnext(f^r)))^r = bi(\ f \longrightarrow wprev(f)\ )$ **using** *8 9 5 int-eq* **by** *fastforce*
have *11*: ⊢ $(\square((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \square(f^r))^r =$

$(((\Box((f^r) \longrightarrow wnext(f^r)))^r \wedge (f^r)^r \longrightarrow (\Box(f^r))^r))$ **by** (*metis int-eq rev-fun2*)
have *12*: $\vdash ((\Box((f^r) \longrightarrow wnext(f^r)))^r \wedge (f^r)^r \longrightarrow (\Box(f^r))^r) =$
$(bi(\ f \longrightarrow wprev(f)\ ) \wedge f \longrightarrow bi\ f\ )$ **using** *8 3 4 10* **by** *fastforce*
from *2 11 12* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *PrevLoop*:
assumes $\vdash f \longrightarrow prev\ f$
shows $\vdash \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow prev\ f$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow (\ more \wedge wprev\ f)$
by (*metis ChopSkipImpMore Prop05 Prop12 WprevEqvEmptyOrPrev inteq-reflection lift-imp-trans prev-d-def*)
hence *3*: $\vdash f \longrightarrow wprev\ f$ **by** *auto*
hence *4*: $\vdash bi(f \longrightarrow wprev\ f)$ **by** (*rule BiGen*)
have *5*: $\vdash bi(f \longrightarrow wprev\ f) \wedge f \longrightarrow bi\ f$ **by** (*rule BiInduct*)
hence *6*: $\vdash bi(f \longrightarrow wprev\ f) \longrightarrow (f \longrightarrow bi\ f)$ **by** *fastforce*
have *7*: $\vdash (f \longrightarrow bi\ f)$ **using** *4 6 MP* **by** *blast*
have *8*: $\vdash bi\ f \longrightarrow f$ **by** (*rule BiElim*)
have *9*: $\vdash f = bi\ f$ **using** *7 8* **by** *fastforce*
have *10*: $\vdash f \longrightarrow more$ **using** *2* **by** *auto*
hence *11*: $\vdash bi\ f \longrightarrow\ bi\ more$ **using** *BiImpBiRule* **by** *blast*
have *12*: $\vdash \neg(bi\ more)$ **using** *DiEmpty bi-d-def empty-d-def* **by** (*simp add*: *bi-d-def empty-d-def*)
from *7 9 11 12* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

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

lemma *BiEqvAndWprevBi*:
$\vdash bi\ f = (f \wedge wprev(bi\ f))$
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*: $\vdash di(f \wedge g;more) \longrightarrow di(f \wedge di\ g) \vee di(g \wedge di\ f)$
by *auto*
have *28*: $\vdash f;more \longrightarrow di\ f$
by (*simp add*: *ChopImpDi*)
hence *29*: $\vdash g \wedge f;more \longrightarrow g \wedge di\ f$
by *auto*
hence *30*: $\vdash di(g \wedge f;more) \longrightarrow di(g \wedge di\ f)$
using *DiImpDi* **by** *blast*
hence *31*: $\vdash di(g \wedge f;more) \longrightarrow di(f \wedge di\ g) \vee di(g \wedge di\ f)$
by *auto*
have *32*: $\vdash di(f \wedge g) \vee di(f \wedge g;more) \vee di(g \wedge f;more) \longrightarrow$
$di(f \wedge di\ g) \vee di(g \wedge di\ f)$
using *23 27 31* **by** *fastforce*
have *33*: $\vdash di((f \wedge g) \vee (f \wedge g;more) \vee (g \wedge f;more)) \longrightarrow$
$di(f \wedge di\ g) \vee di(g \wedge di\ f)$
using *19 32* **by** *fastforce*
have *34*: $\vdash (di\ f \wedge di\ g) \longrightarrow di(f \wedge di\ g) \vee di(g \wedge di\ f)$
using *18 33* **by** *fastforce*
have *35*: $\vdash f \longrightarrow di\ f$
using *DiIntro* **by** *blast*
hence *36*: $\vdash f \wedge di\ g \longrightarrow di\ f \wedge di\ g$
by *auto*
hence *37*: $\vdash di\ (f \wedge di\ g) \longrightarrow di\ (di\ f \wedge di\ g)$
using *DiImpDi* **by** *blast*
have *38*: $\vdash di\ (di\ f \wedge di\ g) = (di\ f \wedge di\ g)$
using *DiDiAndEqvDi* **by** *blast*
have *39*: $\vdash di\ (f \wedge di\ g) \longrightarrow di\ f \wedge di\ g$
using *37 38* **by** *fastforce*
have *40*: $\vdash g \longrightarrow di\ g$
using *DiIntro* **by** *blast*
hence *41*: $\vdash g \wedge di\ f \longrightarrow di\ f \wedge di\ g$
by *auto*

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

lemma *BoxStateEqvBiFinState*:
$\vdash$ $\Box$ (*init w*) = *bi* (*fin* (*init w*))
proof −
have *1*: $\vdash$ $\Diamond$ ($\neg$ (*init w*)) = #*True* ; ($\neg$(*init w*))
    by (*simp add*: *sometimes-d-def*)
have *2*: $\vdash$ $\Diamond$ (*init*($\neg$ *w*)) = #*True* ; *init* ($\neg$ *w*)
    by (*simp add*: *sometimes-d-def*)
have *3*: $\vdash$ *di* (#*True* $\wedge$ *fin* (*init* ($\neg$ *w*))) = #*True* ; *init* ($\neg$ *w*)
    using *DiAndFinEqvChopState* **by** *blast*
have *4*: $\vdash$ $\Diamond$ (*init*($\neg$ *w*)) = *di* (#*True* $\wedge$ *fin* (*init* ($\neg$ *w*)))
    using *1 2 3* **by** *fastforce*
have *5*: $\vdash$ ($\neg$ ($\Diamond$ (*init*($\neg$ *w*)))) = ($\neg$ (*di* (#*True* $\wedge$ *fin* (*init* ($\neg$ *w*)))))
    using *4* **by** *fastforce*
have *6*: $\vdash$ $\Box$ (*init w*) = ($\neg$ (*di* (#*True* $\wedge$ *fin* (*init* ($\neg$ *w*)))))
    using *5 always-d-def Initprop*(*2*) **by** (*metis int-eq*)
have *7*: $\vdash$ $\Box$ (*init w*) = *bi* ($\neg$ (*fin* (*init* ($\neg$ *w*))))
    using *6* **by** (*simp add*: *bi-d-def*)
have *8*: $\vdash$ *init* ($\neg$ *w*) = ($\neg$ (*init w*))
    using *Initprop*(*2*) **by** *fastforce*
have *9*: $\vdash$ *fin* (*init* ($\neg$ *w*)) = *fin* ($\neg$ (*init w*))
    using *8 FinEqvFin* **by** *blast*
have *10*: $\vdash$ *fin* (*init* ($\neg$ *w*)) = ($\neg$ (*fin* (*init w*)))
    using *8 FinNotStateEqvNotFinState FinEqvFin* **by** *blast*
have *11*: $\vdash$ ($\neg$ (*fin* (*init* ($\neg$ *w*)))) = (*fin* (*init w*))
    using *10* **by** *fastforce*
have *12*: $\vdash$ *bi* ($\neg$ (*fin* (*init* ($\neg$ *w*)))) = *bi* (*fin* (*init w*))
    using *11* **by** (*simp add*: *BiEqvBi*)
have *13*: $\vdash$ $\Box$ (*init w*) = *bi* (*fin* (*init w*))
    using *7 12* **by** *fastforce*
from *13* **show** *?thesis* **by** *simp*
qed

lemma *DiamondStateEqvDiFinState*:
$\vdash$ $\Diamond$ (*init w*) = *di* (*fin* (*init w*))
proof −
have *1*: $\vdash$ $\Box$ (*init* ($\neg$ *w*)) = *bi* (*fin* (*init* ($\neg$ *w*)))
    using *BoxStateEqvBiFinState* **by** *blast*
have *2*: $\vdash$ ($\neg$ ($\Box$ (*init* ($\neg$ *w*)))) = ($\neg$ (*bi* (*fin* (*init* ($\neg$ *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*: ⊢ (¬(*more*;#*True*)) = (¬ *more*) **by** *fastforce*
from *1 2 3 5* **show** *?thesis* **using** *NotEmptyEqvMore* **by** *fastforce*
qed

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

lemma *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

16.4.3 Strict initial intervals

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

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

lemma *BsEqvNotDsNot*:
$\vdash$ *bs f* = ($\neg$(*ds* ($\neg$ *f*)))
proof –
have *1*: $\vdash$ *ds* ($\neg$ *f*) = (*more* $\wedge$ (*di* ($\neg$ *f*));*skip*)
by (*rule DsMoreDi*)
hence *2*: $\vdash$ ($\neg$(*ds* ($\neg$ *f*))) = ($\neg$(*more* $\wedge$ (*di* ($\neg$ *f*));*skip*))
by *auto*
have *3*: $\vdash$ ($\neg$(*more* $\wedge$ (*di* ($\neg$ *f*));*skip*)) = (*empty* $\vee$ $\neg$((*di* ($\neg$ *f*));*skip*))
using *NotEmptyEqvMore* **by** *auto*
have *4*: $\vdash$ (*empty* $\vee$ $\neg$((*di* ($\neg$ *f*));*skip*)) = (*empty* $\vee$ $\neg$(($\neg$(*bi f*));*skip*))
using *DiNotEqvNotBi* **by** (*metis 3 inteq-reflection*)
have *5*: $\vdash$ ($\neg$(($\neg$(*bi f*));*skip*)) = (*empty* $\vee$ (*bi f*);*skip*)

by (*rule NotNotChopSkip*)
hence *6*: ⊢ (*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*:
⊢ (¬(*bs f*)) = *ds* (¬ *f*)
proof −
have *1*: ⊢ *bs f* = (¬(*ds* (¬ *f*))) **by** (*rule BsEqvNotDsNot*)
hence *2*: ⊢ (¬(*bs f*)) = (¬¬(*ds* (¬ *f*))) **by** *auto*
from *2* **show** *?thesis* **by** *auto*
qed

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

lemma *NotDsAndEmpty*:
⊢ ¬(*ds f* ∧ *empty*)
proof −
have *1*: ⊢ *ds f* = (*more* ∧ (*di f*);*skip*) **by** (*rule DsMoreDi*)
have *2*: ⊢ *more* ∧ (*di f*);*skip* ∧ *empty* ⟶ #*False* **using** *NotEmptyEqvMore* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

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

lemma *BsAndEqv*:
⊢ (*bs f* ∧ *bs g*) = *bs*(*f* ∧ *g*)
proof −
have *1*: ⊢ *bs f* = (*empty* ∨ (*bi f*) ;*skip*)
by (*simp add*: *bs-d-def*)
have *2*: ⊢ *bs g* = (*empty* ∨ (*bi g*) ;*skip*)
by (*simp add*: *bs-d-def*)
have *3*: ⊢ (*bs f* ∧ *bs g*) = ((*empty* ∨ (*bi f*) ;*skip*) ∧ (*empty* ∨ (*bi g*) ;*skip*))
using *1 2* **by** *fastforce*

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

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

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

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

lemma *DsAndImp*:
$\vdash$ *ds* (*f* $\wedge$ *g*) $\longrightarrow$ *ds f* $\wedge$ *ds g*
proof –
have *1*: $\vdash$ *bs* ($\neg$*f*) $\vee$ *bs* ($\neg$*g*) $\longrightarrow$ *bs*($\neg$*f* $\vee$ $\neg$*g*) **by** (*rule BsOrImp*)

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

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

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

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

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

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

lemma *BsImpBsRule*:
 assumes $\vdash f \longrightarrow g$
 shows $\vdash bs\ f \longrightarrow bs\ g$
proof −
 have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
 hence *2*: $\vdash bi\ f \longrightarrow bi\ g$ **by** (*rule BiImpBiRule*)
 hence *3*: $\vdash (bi\ f);skip \longrightarrow (bi\ g);skip$ **by** (*rule LeftChopImpChop*)
 hence *4*: $\vdash empty \lor (bi\ f);skip \longrightarrow empty \lor (bi\ g);skip$ **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*:
$\vdash (bs\ f \lor bs\ g) = bs(bi\ f \lor bi\ g)$
proof −
 have *1*: $\vdash (bs\ f \lor bs\ g) = ((empty \lor (bi\ f);skip) \lor (empty \lor (bi\ g);skip))$
  by (*simp add*: *bs-d-def*)
 have *2*: $\vdash ((empty \lor (bi\ f);skip) \lor (empty \lor (bi\ g);skip)) = (empty \lor (bi\ f);skip \lor (bi\ g);skip)$
  by *auto*
 have *3*: $\vdash ((bi\ f);skip \lor (bi\ g);skip) = (bi\ f \lor bi\ g);skip$
  using *OrChopEqv* **by** *fastforce*
 hence *4*: $\vdash (empty \lor (bi\ f);skip \lor (bi\ g);skip) = (empty \lor (bi\ f \lor bi\ g);skip)$
  by *auto*
 have *5*: $\vdash (bi\ f \lor bi\ g) = bi\ (\ bi\ f \lor bi\ g)$
  by (*meson BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB Prop02 int-iffI*)
 hence *6*: $\vdash (bi\ f \lor bi\ g);skip = bi\ (\ bi\ f \lor bi\ g);skip$
  by (*simp add*: *LeftChopEqvChop*)
 hence *7*: $\vdash (empty \lor bi\ (\ bi\ f \lor bi\ g);skip) = (empty \lor (bi\ f \lor bi\ g);skip)$
  by *auto*
 have *8*: $\vdash (empty \lor (bi\ f \lor bi\ g);skip) = bs(bi\ f \lor bi\ g)$ **using** *bs-d-def*
  by (*metis 4 5 inteq-reflection*)
 from *1 2 4 8* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *DiOrDsEqvDi*:
$\vdash$ *di f* $\vee$ *ds f* $=$ *di f*
proof $-$
have *1*: $\vdash$ *di f* $\longrightarrow$ *di f* $\vee$ *ds f* **by** *auto*
have *2*: $\vdash$ *di f* $\longrightarrow$ *di f* **by** *auto*
have *3*: $\vdash$ *ds f* $\longrightarrow$ *di f* **by** (*rule DsImpDi*)
have *4*: $\vdash$ *di f* $\vee$ *ds f* $\longrightarrow$ *di f* **using** *2 3* **by** *auto*
from *1 4* **show** *?thesis* **by** *auto*
qed

lemma *DiAndDsEqvDs*:
$\vdash$ (*di f* $\wedge$ *ds f*) $=$ *ds f*
proof $-$
have *1*: $\vdash$ *di f* $\wedge$ *ds f* $\longrightarrow$ *ds f* **by** *auto*
have *2*: $\vdash$ *ds f* $\longrightarrow$ *ds f* **by** *auto*
have *3*: $\vdash$ *ds f* $\longrightarrow$ *di f* **by** (*rule DsImpDi*)
have *4*: $\vdash$ *ds f* $\longrightarrow$ *di f* $\wedge$ *ds f* **using** *2 3* **by** *auto*
from *1 4* **show** *?thesis* **by** *auto*
qed

lemma *OrDsEqvDi*:
$\vdash$ (*f* $\vee$ *ds f*) $=$ *di f*
proof $-$
have *1*: $\vdash$ *ds f* $=$ (*di f*);*skip* **by** (*rule DsDi*)
hence *2*: $\vdash$ (*f* $\vee$ *ds f*) $=$ (*f* $\vee$ (*di f*);*skip*) **by** *auto*
from *2* **show** *?thesis* **using** *DiEqvOrDiChopSkipB* **by** *fastforce*
qed

lemma *AndBsEqvBi*:
$\vdash$ (*f* $\wedge$ *bs f*) $=$ *bi f*
proof $-$
have *1*: $\vdash$ (*f* $\wedge$ *bs f*) $=$ (*f* $\wedge$ (*empty* $\vee$ (*bi f*);*skip*)) **by** (*simp add*: *bs-d-def*)
from *1* **show** *?thesis* **using** *BiEqvAndEmptyOrBiChopSkip* **by** *fastforce*
qed

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

lemma *StateImpBs*:
$\vdash$ *init w* $\longrightarrow$ *bs* (*init w*)
proof $-$
have *1*: $\vdash$ *init w* $=$ *bi* (*init w*) **by** (*rule StateEqvBi*)
have *2*: $\vdash$ *bi* (*init w*) $\longrightarrow$ *bs* (*init w*) **by** (*rule BiImpBs*)

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

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

lemma *BsEqvBiMoreImpChop*:
$\vdash bs\ f = bi(more \longrightarrow f;skip)$
proof −
have *1*: $\vdash bs\ f = (empty \vee (bi\ f;skip))$
by (*simp add*: *bs-d-def*)
have *2*: $\vdash (empty \vee (bi\ f;skip)) = ((\neg(\neg(bi\ f));skip))$
using *NotNotChopSkip* **by** *fastforce*
have *3*: $\vdash \neg((\neg(bi\ f));skip) = (\neg(di\ (\neg\ f);skip))$
by (*simp add*: *bi-d-def*)
have *4*: $\vdash (\neg(di\ (\neg\ f);skip)) = (\neg(((\neg\ f)\ ;\#True);skip))$
by (*simp add*:*di-d-def*)
have *5*: $\vdash (\neg(((\neg\ f)\ ;\#True);skip)) = (\neg((\neg\ f)\ ;(\#True;skip)))$
using *ChopAssocB* **by** *fastforce*
have *6*: $\vdash (\neg((\neg\ f)\ ;(\#True;skip))) = (\neg((\neg\ f)\ ;(skip;\#True)))$
using *SkipTrueEqvTrueSkip TrueChopSkipEqvSkipChopTrue RightChopEqvChop* **by** *fastforce*
have *7*: $\vdash (\neg((\neg\ f)\ ;(skip;\#True))) = (\neg(((\neg\ f)\ ;skip);\#True))$
using *ChopAssoc* **by** *fastforce*
have *8*: $\vdash (\neg(((\neg\ f)\ ;skip);\#True)) = (\neg(di\ ((\neg\ f);skip)))$
by (*simp add*: *di-d-def*)

have *9*: $\vdash$ ($\neg$(*di* (($\neg$ *f*);*skip*))) = *bi* ($\neg$(($\neg$ *f*) ;*skip*))
  using *NotDiEqvBiNot* **by** *blast*
have *10*: $\vdash$ *bi* ($\neg$(($\neg$ *f*) ;*skip*)) = *bi*(*empty* $\vee$ (*f*;*skip*))
  using *NotNotChopSkip* **using** *BiEqvBi* **by** *blast*
have *11*: $\vdash$ *bi*(*empty* $\vee$ (*f*;*skip*)) = *bi*($\neg$ *more* $\vee$ (*f*;*skip*))
  by (*simp add*: *empty-d-def*)
have *12*: $\vdash$ ($\neg$ *more* $\vee$ (*f*;*skip*)) = (*more* $\longrightarrow$ *f*;*skip*) **by** *auto*
have *13*: $\vdash$ *bi*($\neg$ *more* $\vee$ (*f*;*skip*)) = *bi*(*more* $\longrightarrow$ *f*;*skip*) **using** *12* **using** *BiEqvBi* **by** *blast*
have *14*: $\vdash$ *bs f* = ($\neg$ ((($\neg$ *f*);*skip*);#*True*)) **using** *1 2 3 4 5 6 7* **by** *fastforce*
have *15*: $\vdash$ ($\neg$ ((($\neg$ *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*:
$\vdash$ $\Box$(*more* $\longrightarrow$ $\neg$ (*init w*)) = *bs*($\neg$(*fin*(*init w*)))
proof −
have *1*: $\vdash$ $\Box$(*more* $\longrightarrow$ $\neg$ (*init w*)) = ($\neg$($\Diamond$($\neg$(*more* $\longrightarrow$ $\neg$ (*init w*)))))
  by (*simp add*: *always-d-def*)
have *01*: $\vdash$ ($\neg$(*more* $\longrightarrow$ $\neg$ (*init w*))) = (*init w* $\wedge$ *more*) **by** *auto*
hence *2*: $\vdash$ $\neg$($\Diamond$($\neg$(*more* $\longrightarrow$ $\neg$ (*init w*)))) = ($\neg$(#*True*;(*init w* $\wedge$ *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* $\wedge$ *more*) = (*init w* $\wedge$ (#*True*; *skip*))
  using *3* **by** *auto*
have *5*: $\vdash$ (*init w* $\wedge$ (#*True*; *skip*)) = ((*init w* $\wedge$ *empty*);(#*True*;*skip*))
  using *StateAndEmptyChop* **by** *fastforce*
have *6*: $\vdash$ (*init w* $\wedge$ *more*) = ((*init w* $\wedge$ *empty*);(#*True*;*skip*))
  using *4 5* **by** *fastforce*
have *7*: $\vdash$ (#*True*;(*init w* $\wedge$ *more*)) = (#*True*;((*init w* $\wedge$ *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*: ⊢ *empty* ∨ (¬((#*True*;(*init w* ∧ *empty*));#*True*));*skip* =
empty ∨(*bi*(□ (¬(*init w* ∧ *empty*))));*skip*
by *auto*
have *16*: ⊢ (¬((((#*True*;(*init w* ∧ *empty*));#*True*);*skip*))) =
(*empty* ∨ (*bi*(□ (¬(*init w* ∧ *empty*)));*skip*))
using *12 15* **using** *14 NotChopNotSkip int-eq* **by** *fastforce*
have *171*: ⊢ (¬(*init w* ∧ *empty*)) = (¬(*init w*) ∨ ¬*empty*)
by *auto*
hence *172*: ⊢ □(¬(*init w* ∧ *empty*)) = □(¬(*init w*) ∨ ¬*empty*)
by (*simp add*: *BoxEqvBox*)
hence *173*: ⊢ *bi*(□(¬(*init w* ∧ *empty*))) = *bi*(□(¬(*init w*) ∨ ¬*empty*))
by (*simp add*: *BiEqvBi*)
hence *174*: ⊢ *bi*(□(¬(*init w* ∧ *empty*)));*skip* = *bi*(□(¬(*init w*) ∨ ¬*empty*));*skip*
using *LeftChopEqvChop* **by** *blast*
hence *17*: ⊢ (*empty* ∨ (*bi*(□ (¬(*init w* ∧ *empty*)));*skip*)) =
(*empty* ∨ (*bi*(□ (¬(*init w*) ∨ ¬*empty*));*skip*))
by *auto*
have *181*: ⊢ (¬(*init w*) ∨ ¬*empty*) = (¬*empty* ∨ ¬(*init w*))
by *auto*
hence *18*: ⊢ □ (¬(*init w*) ∨ ¬*empty*) = □ (¬*empty* ∨ ¬(*init w*))
by (*simp add*: *BoxEqvBox*)
have *191*: ⊢ (¬*empty* ∨ ¬(*init w*)) = (*empty* ⟶ ¬(*init w*))
by *auto*
hence *19*: ⊢ □ (¬*empty* ∨ ¬(*init w*)) = □(*empty* ⟶ ¬(*init w*))
by (*simp add*: *BoxEqvBox*)
have *20*: ⊢ □(*empty* ⟶ ¬(*init w*)) = *fin* (¬(*init w*))
by (*simp add*: *fin-d-def*)
have *21*: ⊢ *fin* (¬(*init w*)) = (¬(*fin* (*init w*)))
using *FinEqvFin FinNotStateEqvNotFinState Initprop*(*2*) **by** *fastforce*
have *22*: ⊢ *bi*(□ (¬(*init w*) ∨ ¬*empty*)) = *bi* (¬(*fin* (*init w*)))
using *18 19 20 21 BiEqvBi* **by** (*metis int-eq*)
hence *23*: ⊢ (*bi*(□ (¬(*init w*) ∨ ¬*empty*)));*skip* = (*bi* (¬(*fin* (*init w*))));*skip*
using *RightChopEqvChop* **by** (*simp add*: *LeftChopEqvChop*)
hence *24*: ⊢ (*empty* ∨ (*bi*(□ (¬(*init w*) ∨ ¬*empty*)));*skip*) =
(*empty* ∨ (*bi* (¬(*fin* (*init w*))));*skip*)
by *auto*
hence *25*: ⊢ (*empty* ∨ (*bi* (¬(*fin* (*init w*))));*skip*) = *bs*(¬(*fin* (*init w*)))
by (*simp add*:*bs-d-def*)
from *1 2 11 16 17 24 25* **show** *?thesis* **by** *fastforce*
qed

lemma *BsFalseEqvEmpty*:
⊢ *bs* #*False* = *empty*
proof −
have *1*: ⊢ *bs* #*False* = (*empty* ∨ *bi* #*False*;*skip*)
by (*simp add*: *bs-d-def*)
have *2*: ⊢ ¬(*bi* #*False*;*skip*)
by (*metis BiEqvAndWprevBi MoreEqvSkipChopTrue NotChopSkipEqvMoreAndNotChopSkip*
SkipTrueEqvTrueSkip int-eq int-iffD1 int-simps(*14*) *int-simps*(*19*) *int-simps*(*2*)

$int\text{-}simps(21)$)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

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

qed

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

lemma *FstTrue*:
$\vdash \rhd\ \#True = empty$
proof −
have *1*: $\vdash \rhd\ \#True = (\#True \wedge bs\ (\neg\ \#True))$
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*: $\vdash \neg((bi\ (\neg\ \#True));skip)$
by (*metis AndChopA BiEqvAndEmptyOrBiChopSkip MoreEqvSkipChopTrue NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip int-eq int-simps*(*14*) *int-simps*(*21*))
have *5*: $\vdash bs\ (\neg\ \#True) = empty$
using *2 4* **by** *fastforce*
from *1 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstFalse*:
$\vdash \neg(\rhd\ \#False)$
proof −
have *1*: $\vdash \rhd\ \#False = (\#False \wedge bs\ \#True)$ **by** (*simp add*: *first-d-def*)
from *1* **show** *?thesis* **by** *auto*
qed

lemma *FstChopFalseEqvFalse*:
$\vdash \neg(\rhd f\ ;\ \#False)$
by (*simp add*:*Valid-def chop-defs*)

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

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

lemma *FstEmptyOrEqvEmpty*:
$\vdash \triangleright$(*empty* $\vee$ *f*) = *empty*
proof −
 have *1*: $\vdash \triangleright$(*empty* $\vee$ *f*) = (($\triangleright$*empty* $\wedge$ *bs* ($\neg$ *f*)) $\vee$ ($\triangleright$*f* $\wedge$ *bs* ($\neg$ *empty*))) **using** *FstWithOrEqv* **by** *blast*
 have *2*: $\vdash$ ($\neg$ *empty*) = *more* **by** (*simp add*: *empty-d-def*)
 hence *3*: $\vdash$ *bs* ($\neg$ *empty*) = *bs more* **using** *BsEqvRule* **by** *blast*
 have *4*: $\vdash$ *bs more* = *empty* **using** *BsMoreEqvEmpty* **by** *blast*
 have *5*: $\vdash$ ($\triangleright$*f* $\wedge$ *bs* ($\neg$ *empty*)) = ($\triangleright$*f* $\wedge$ *empty*) **using** *3 4* **by** *fastforce*
 have *6*: $\vdash \triangleright$*empty* = *empty* **using** *FstEmpty* **by** *blast*
 hence *7*: $\vdash$ ($\triangleright$*empty* $\wedge$ *bs* ($\neg$ *f*)) = (*empty* $\wedge$ *bs* ($\neg$ *f*)) **by** *auto*
 have *8*: $\vdash$ (*empty* $\wedge$ *bs* ($\neg$ *f*)) = (*empty* $\wedge$(*empty* $\vee$ *bi* ($\neg$ *f*);*skip*)) **by** (*simp add*:*bs-d-def*)
 have *9*: $\vdash$ (*empty* $\wedge$(*empty* $\vee$ *bi* ($\neg$ *f*);*skip*)) = *empty* **by** *auto*
 have *10*: $\vdash$ (*empty* $\wedge$ *bs* ($\neg$ *f*)) = *empty* **using** *8 9* **by** *auto*
 have *11*: $\vdash$ (($\triangleright$*empty* $\wedge$ *bs* ($\neg$ *f*)) $\vee$ ($\triangleright$*f* $\wedge$ *bs* ($\neg$ *empty*))) =
   (*empty* $\vee$ ($\triangleright$*f* $\wedge$ *empty*)) **using** *7 10 5* **by** *fastforce*
 have *12*: $\vdash$ (*empty* $\vee$ ($\triangleright$*f* $\wedge$ *empty*)) = *empty* **by** *auto*
 from *1 11 12* **show** *?thesis* **by** *fastforce*
qed

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

lemma *FstMoreEqvSkip*:
$\vdash \triangleright$ *more* = *skip*
proof −
 have *1*: $\vdash \triangleright$ *more* = (*more* $\wedge$ *bs* ($\neg$ *more*)) **by** (*simp add*: *first-d-def*)
 have *2*: $\vdash$ (*more* $\wedge$ *bs* ($\neg$ *more*)) = (*more* $\wedge$ (*empty* $\vee$ *bi* ($\neg$ *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*: ⊢ (*di f* ∧ (*bs* (¬ *f*) ∨ *di f*)) = *di f* **by** *auto*
from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

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

lemma *DiEqvDiFst*:
⊢ *di f* = *di* (▷ *f*)
proof −
have *1*: ⊢ *di* (▷ *f*) = *di* (*f* ∧ *bs* (¬*f*))
by (*simp add*: *first-d-def*)
have *2*: ⊢ *di* (*f* ∧ *bs* (¬*f*)) ⟶ *di f* ∧ *di* (*bs* (¬ *f*))
using *DiAndImpAnd* **by** *auto*

hence *3*: $\vdash$ *di* $(f \wedge$ *bs* $(\neg f)) \longrightarrow$ *di f*
  by *auto*
have *4*: $\vdash$ *di* $(\triangleright f) \longrightarrow$ *di f* **using** *1 3*
  by *fastforce*
have *5*: $\vdash$ (*di f* $\wedge$ *empty*) = ($f \wedge$ *empty*)
  using *DiAndEmptyEqvAndEmpty* **by** *blast*
have *6*: $\vdash$ ($\triangleright f \wedge$ *empty*) = ($f \wedge$ *empty*)
  using *FstAndEmptyEqvAndEmpty* **by** *auto*
have *7*: $\vdash$ *di f* $\wedge$ *empty* $\longrightarrow \triangleright f$
  using *5 6* **by** *fastforce*
have *8*: $\vdash \triangleright f \longrightarrow$ *di* $(\triangleright f)$
  using *DiIntro* **by** *auto*
have *9*: $\vdash$ *di f* $\wedge$ *empty* $\longrightarrow$ *di* $(\triangleright f)$
  using *7 8* **using** *lift-imp-trans* **by** *blast*
hence *10*: $\vdash$ *empty* $\longrightarrow$ (*di f* $\longrightarrow$ *di* $(\triangleright f)$)
  by *auto*
have *11*: $\vdash$ *prev* (*di f* $\longrightarrow$ *di* $(\triangleright f)) \longrightarrow$ *more*
  by (*simp add*: *ChopSkipImpMore prev-d-def*)
have *12*: $\vdash$ *more* $\longrightarrow$(*prev* (*di f* $\longrightarrow$ *di* $(\triangleright f)$) = (*prev*(*di f*) $\longrightarrow$ *prev*(*di* $(\triangleright f)$)))
  using *MoreImpImpPrevEqv* **by** *auto*
have *13*: $\vdash$ (*more* $\wedge$ *prev* (*di f* $\longrightarrow$ *di* $(\triangleright f)$)) = (*more* $\wedge$ (*prev*(*di f*) $\longrightarrow$ *prev*(*di* $(\triangleright f)$)))
  using *12* **by** *fastforce*
have *14*: $\vdash$ *prev* (*di f* $\longrightarrow$ *di* $(\triangleright f)$) = (*more* $\wedge$ (*prev*(*di f*) $\longrightarrow$ *prev*(*di* $(\triangleright f)$)))
  using *11 13* **by** *fastforce*
have *15*: $\vdash$ *di f* = ($f \vee$ *ds f*)
  using *OrDsEqvDi* **by** *fastforce*
have *16*: $\vdash$ *di f* = (*di f* $\wedge$ (*bs* $(\neg f) \vee \neg$(*bs* $(\neg f)$)))
  by *auto*
have *17*: $\vdash$ (*di f* $\wedge$ (*bs* $(\neg f) \vee \neg$(*bs* $(\neg f)$))) = ((*di f* $\wedge$ *bs* $(\neg f)$) $\vee$ (*di f* $\wedge \neg$(*bs* $(\neg f)$)))
  by *auto*
have *18*: $\vdash$ (*di f* $\wedge$ *bs* $(\neg f)$) = (($f \vee$ *ds f*) $\wedge$ *bs* $(\neg f)$)
  using *15* **by** *auto*
have *19*: $\vdash$ (($f \vee$ *ds f*) $\wedge$ *bs* $(\neg f)$) = (($f \wedge$ *bs* $(\neg f)$) $\vee$ (*ds f* $\wedge$ *bs* $(\neg f)$))
  by *auto*
have *20*: $\vdash \neg$(*ds f* $\wedge$ *bs* $(\neg f)$)
  by (*simp add*: *ds-d-def*)
have *21*: $\vdash$ (($f \wedge$ *bs* $(\neg f)$) $\vee$ (*ds f* $\wedge$ *bs* $(\neg f)$)) = ($f \wedge$ *bs* $(\neg f)$)
  using *20* **by** *auto*
have *22*: $\vdash$ (*di f* $\wedge$ *bs* $(\neg f)$) = ($f \wedge$ *bs* $(\neg f)$)
  using *18 19 21* **by** *fastforce*
have *23*: $\vdash$ ($f \wedge$ *bs* $(\neg f)$) = $\triangleright f$
  by (*simp add*: *first-d-def*)
have *24*: $\vdash$ $(\triangleright f) \longrightarrow$ *di* $(\triangleright f)$
  using *DiIntro* **by** *auto*
have *25*: $\vdash$ ($f \wedge$ *bs* $(\neg f)$) $\longrightarrow$ *di* $(\triangleright f)$
  using *23 24* **by** *fastforce*
have *26*: $\vdash$ (*di f* $\wedge$ *bs* $(\neg f)$) $\longrightarrow$ *di* $(\triangleright f)$
  using *25 22* **by** *fastforce*
hence *27*: $\vdash$ (*di f* $\wedge$ *bs* $(\neg f) \wedge$ (*prev* (*di f* $\longrightarrow$ *di* $(\triangleright f)$))) $\longrightarrow$ *di* $(\triangleright f)$
  by *auto*

have *28*: $\vdash$ $(di\ f \wedge \neg(bs\ (\neg\ f))) = (di\ f \wedge ds\ f)$
  by (*simp add*: *ds-d-def*)
hence *29*: $\vdash$ $(di\ f \wedge \neg(bs\ (\neg\ f)) \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f)))) =$
    $(di\ f \wedge ds\ f \wedge\ (prev\ (di\ f \longrightarrow di\ (\rhd\ f))))$
  by *auto*
have *30*: $\vdash$ $ds\ f = prev(di\ f)$
  using *DsDi* **by** (*metis prev-d-def*)
hence *31*: $\vdash$ $(di\ f \wedge ds\ f \wedge\ (prev\ (di\ f \longrightarrow di\ (\rhd\ f)))) =$
    $(di\ f \wedge prev(di\ f) \wedge\ (prev\ (di\ f \longrightarrow di\ (\rhd\ f))))$
  by *auto*
have *32*: $\vdash$ $prev\ (\ di\ f \longrightarrow di\ (\rhd\ f)) \longrightarrow\ (prev(di\ f) \longrightarrow prev(di\ (\rhd f)))$
  using *14* **by** *auto*
hence *33*: $\vdash$ $di\ f \wedge prev(di\ f) \wedge prev\ (\ di\ f \longrightarrow di\ (\rhd\ f)) \longrightarrow$
    $di\ f \wedge prev(di\ f) \wedge (prev(di\ f) \longrightarrow prev(di\ (\rhd f)))$
  by *auto*
have *34*: $\vdash$ $di\ f \wedge prev(di\ f) \wedge (prev(di\ f) \longrightarrow prev(di\ (\rhd f))) \longrightarrow prev(di\ (\rhd f))$
  by *auto*
have *35*: $\vdash$ $prev(di\ (\rhd f))= (di\ (\rhd f));skip$
  by (*simp add*: *prev-d-def*)
have *36*: $\vdash$ $(di\ (\rhd f));skip \longrightarrow di(di\ (\rhd f))$
  using *ChopImpDi* **by** *auto*
have *37*: $\vdash$ $di(di\ (\rhd f)) = di\ (\rhd f)$
  using *DiEqvDiDi* **by** *fastforce*
have *38*: $\vdash$ $di\ f \wedge prev(di\ f) \wedge (prev(di\ f) \longrightarrow prev(di\ (\rhd f))) \longrightarrow di\ (\rhd f)$
  using *37 36 35 34* **by** *fastforce*
have *39*: $\vdash$ $di\ f \wedge \neg(bs\ (\neg\ f)) \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow di\ (\rhd f)$
  using *29 31 33 38* **by** *fastforce*
hence *40*: $\vdash$ $\neg(bs\ (\neg\ f)) \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow (di\ f \longrightarrow di\ (\rhd f))$
  by *fastforce*
have *41*: $\vdash$ $bs\ (\neg\ f) \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow(di\ f\ \longrightarrow di\ (\rhd f))$
  using *27* **by** *fastforce*
have *42*: $\vdash$ $(\neg(bs\ (\neg\ f)) \vee bs\ (\neg\ f)) \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow(di\ f\ \longrightarrow di\ (\rhd f))$
  using *40 41* **by** *fastforce*
have *43*: $\vdash$ $(\neg(bs\ (\neg\ f)) \vee bs\ (\neg\ f))$
  by *auto*
have *44*: $\vdash$ $(prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow(di\ f\ \longrightarrow di\ (\rhd f))$
  using *42 43* **by** *fastforce*
have *45*: $\vdash$ $di\ f\ \longrightarrow di\ (\rhd f)$
  using *10 44 EmptyChopSkipInduct* **by** *blast*
from *4 45* **show** *?thesis* **by** *fastforce*
qed

lemma *FstDiEqvFst*:
$\vdash \rhd(di\ f) = \rhd f$
proof –
have *1*: $\vdash$ $\rhd(di\ f) = (di\ f \wedge bs\ (\neg\ (di\ f)))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash$ $(\neg\ (di\ f)) = bi\ (\neg\ f)$ **by** (*simp add*: *NotDiEqvBiNot*)
hence *3*: $\vdash$ $bs\ (\neg\ (di\ f)) = bs\ (bi\ (\neg\ f))$ **using** *BsEqvRule* **by** *blast*
have *4*: $\vdash$ $bs\ (bi\ (\neg\ f)) = bs\ (\ \neg\ f)$ **using** *BsEqvBsBi* **by** *fastforce*
hence *5*: $\vdash$ $(di\ f \wedge bs\ (\neg\ (di\ f))) = (di\ f \wedge bs\ (\ \neg\ f))$ **using** *3* **by** *fastforce*

have *6* : $\vdash di\ f = (f \lor ds\ f)$ **using** *OrDsEqvDi* **by** *fastforce*
hence *7*: $\vdash (di\ f \land bs\ (\neg\ f)) = ((f \lor ds\ f) \land bs\ (\neg\ f))$ **by** *auto*
have *8*: $\vdash ((f \lor ds\ f) \land bs\ (\neg\ f)) = ((f \land bs\ (\neg\ f)) \lor (ds\ f \land bs\ (\neg\ f)))$ **by** *auto*
have *9*: $\vdash \neg(ds\ f \land bs\ (\neg\ f))$ **by** (*simp add*: *ds-d-def*)
have *10*: $\vdash (f \land bs\ (\neg\ f)) = \triangleright f$ **by** (*simp add*: *first-d-def*)
have *11*: $\vdash ((f \land bs\ (\neg\ f)) \lor (ds\ f \land bs\ (\neg\ f))) = \triangleright f$ **using** *9 10* **by** *fastforce*
from *1 5 7 8 11* **show** *?thesis* **by** (*metis int-eq*)
qed

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

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

lemma *FstDiAndDiEqv*:
$\vdash \triangleright(di\ f \land di\ g) = ((\triangleright f \land di\ g) \lor (\triangleright g \land di\ f))$
proof −
have *1*: $\vdash \triangleright(di\ f \land di\ g) = ((di\ f \land di\ g) \land bs\ (\neg(di\ f \land di\ g)))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash (\neg(di\ f \land di\ g)) = (bi\ (\neg f) \lor bi\ (\neg g))$ **by** (*auto simp add*: *bi-d-def*)
hence *3*: $\vdash bs\ (\neg(di\ f \land di\ g)) = bs(bi\ (\neg f) \lor bi\ (\neg g))$ **using** *BsEqvRule* **by** *blast*
hence *4*: $\vdash ((di\ f \land di\ g) \land bs\ (\neg(di\ f \land di\ g))) =$
$(di\ f \land di\ g \land bs(bi\ (\neg f) \lor bi\ (\neg g)))$ **by** *auto*
have *5*: $\vdash (bs\ (\neg f) \lor bs\ (\neg g)) = bs(bi\ (\neg f) \lor bi\ (\neg g))$ **using** *BsOrBsEqvBsBiOrBi* **by** *blast*
hence *6*: $\vdash (di\ f \land di\ g \land bs(bi\ (\neg f) \lor bi\ (\neg g))) =$
$(di\ f \land di\ g \land (bs\ (\neg f) \lor bs\ (\neg g)))$ **by** *auto*
have *7*: $\vdash (di\ f \land di\ g \land (bs\ (\neg f) \lor bs\ (\neg g))) =$
$((bs\ (\neg f) \land di\ f \land di\ g) \lor (di\ f \land bs\ (\neg g) \land di\ g))$ **by** *auto*
have *8*: $\vdash \triangleright f = (bs\ (\neg f) \land di\ f)$ **using** *FstEqvBsNotAndDi* **by** *blast*
hence *9*: $\vdash (bs\ (\neg f) \land di\ f \land di\ g) = (\triangleright f \land di\ g)$ **by** *auto*
have *10*: $\vdash \triangleright g = (bs\ (\neg g) \land di\ g)$ **using** *FstEqvBsNotAndDi* **by** *blast*
hence *11*: $\vdash (di\ f \land bs\ (\neg g) \land di\ g) = (di\ f \land \triangleright g)$ **by** *auto*
have *12*: $\vdash (di\ f \land di\ g \land (bs\ (\neg f) \lor bs\ (\neg g))) =$
$((\triangleright f \land di\ g) \lor (di\ f \land \triangleright g))$ **using** *7 9 11* **by** (*metis int-eq*)
from *1 4 6 12* **show** *?thesis* **using** *inteq-reflection lift-and-com* **by** *fastforce*
qed

lemma *BiNotFstEqvBiNot*:
$\vdash bi\ (\neg\ (\triangleright f)) = bi\ (\neg\ f)$
proof −

have *1*: $\vdash$ *di f* = *di* ($\triangleright$*f*) **using** *DiEqvDiFst* **by** *blast*
hence *2*: $\vdash$ ($\neg$(*di f*)) = ($\neg$(*di* ($\triangleright$*f*))) **by** *auto*
from *1 2* **show** *?thesis* **using** *NotDiEqvBiNot* **by** *fastforce*
qed

lemma *BsNotFstEqvBsNot*:
$\vdash$ *bs* ($\neg$ ($\triangleright$*f*)) = *bs* ($\neg$ *f*)
proof −
have *1*: $\vdash$ *bs* ($\neg$ ($\triangleright$*f*)) = (*empty* $\vee$ *bi* ($\neg$ ($\triangleright$*f*));*skip*) **by** (*simp add*: *bs-d-def*)
have *2*: $\vdash$ *bi* ($\neg$ ($\triangleright$*f*)) = *bi* ($\neg$ *f*) **using** *BiNotFstEqvBiNot* **by** *blast*
hence *3*: $\vdash$ *bi* ($\neg$ ($\triangleright$*f*));*skip* = *bi* ($\neg$ *f*);*skip* **using** *LeftChopEqvChop* **by** *blast*
hence *4*: $\vdash$ (*empty* $\vee$ *bi* ($\neg$ ($\triangleright$*f*));*skip*) = (*empty* $\vee$ *bi* ($\neg$ *f*);*skip*) **by** *auto*
from *1 4* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *FstState*:
$\vdash$ $\triangleright$ (*init w*) = (*empty* $\wedge$ *init w*)
proof −
have *1*: $\vdash$ $\triangleright$ (*init w*) = (*init w* $\wedge$ *bs* ($\neg$(*init w*))) **by** (*simp add*: *first-d-def*)
hence *2*: $\vdash$ $\triangleright$ (*init w*) $\longrightarrow$ *init w* **by** *auto*
have *3*: $\vdash$ *init w* $\longrightarrow$ *bs* (*init w*) **using** *StateImpBs* **by** *auto*
have *4*: $\vdash$ $\triangleright$ (*init w*) $\longrightarrow$ *bs* (*init w*) **using** *2 3* **by** *fastforce*
have *5*: $\vdash$ $\triangleright$ (*init w*) $\longrightarrow$ *bs* ($\neg$(*init w*)) **using** *1* **by** *auto*
have *6*: $\vdash$ $\triangleright$ (*init w*) $\longrightarrow$ *bs* (*init w*) $\wedge$ *bs* ($\neg$(*init w*)) **using** *4 5* **by** *fastforce*
have *7*: $\vdash$ (*bs* (*init w*) $\wedge$ *bs* ($\neg$(*init w*))) = (*bs*((*init w*) $\wedge$ $\neg$(*init w*))) **using** *BsAndEqv* **by** *blast*
have *8*: $\vdash$ ((*init w*) $\wedge$ $\neg$(*init w*)) = #*False* **by** *auto*
hence *9*: $\vdash$ (*bs*((*init w*) $\wedge$ $\neg$(*init w*))) = *bs* #*False* **using** *BsEqvRule* **by** *blast*
have *10*: $\vdash$ *bs* #*False* = *empty* **using** *BsFalseEqvEmpty* **by** *auto*
have *11*: $\vdash$ $\triangleright$ (*init w*) $\longrightarrow$ *empty* **using** *10 9 7 6* **by** *fastforce*
have *12*: $\vdash$ $\triangleright$ (*init w*) $\longrightarrow$ *empty* $\wedge$ *init w* **using** *11 2* **by** *fastforce*
have *13*: $\vdash$ *empty* $\wedge$ *init w* $\longrightarrow$ *empty* **by** *auto*
hence *14*: $\vdash$ *empty* $\wedge$ *init w* $\longrightarrow$ *empty* $\vee$ *bi* ($\neg$(*init w*));*skip* **by** *auto*
hence *15*: $\vdash$ *empty* $\wedge$ *init w* $\longrightarrow$ *bs* ($\neg$(*init w*)) **by** (*simp add*: *bs-d-def*)
have *16*: $\vdash$ *empty* $\wedge$ *init w* $\longrightarrow$ *init w* **by** *auto*
have *17*: $\vdash$ *empty* $\wedge$ *init w* $\longrightarrow$ *init w* $\wedge$ *bs* ($\neg$(*init w*)) **using** *16 15* **by** *auto*
hence *18*: $\vdash$ *empty* $\wedge$ *init w* $\longrightarrow$ $\triangleright$(*init w*) **by** (*simp add*: *first-d-def*)
from *12 18* **show** *?thesis* **by** *fastforce*
qed

lemma *FstStateAndBsNotEmpty*:
$\vdash$ ($\triangleright$ (*init w*) $\wedge$ *bs* ($\neg$ *empty*)) = $\triangleright$ (*init w*)
proof −
have *1*: $\vdash$ ($\triangleright$ (*init w*) $\wedge$ *bs* ($\neg$ *empty*)) = ($\triangleright$ (*init w*) $\wedge$ *bs more*)
  using *BsEqvRule NotEmptyEqvMore* **by** (*simp add*: *empty-d-def*)
have *2*: $\vdash$ ($\triangleright$ (*init w*) $\wedge$ *bs more*) = ($\triangleright$ (*init w*) $\wedge$ *empty*)
  using *BsMoreEqvEmpty* **by** *fastforce*
have *3*: $\vdash$ $\triangleright$ (*init w*) = (*empty* $\wedge$ (*init w*))
  using *FstState* **by** *blast*
hence *4*: $\vdash$ ($\triangleright$ (*init w*) $\wedge$ *empty*) = (*empty* $\wedge$ (*init w*) $\wedge$ *empty*)
  by *auto*

have *5*: ⊢ (*empty* ∧ (*init w*) ∧ *empty*) = (*empty* ∧ (*init w*))
  by *auto*
have *6*: ⊢ (*empty* ∧ (*init w*)) = ▷(*init w*)
  using *FstState* **by** *fastforce*
from *1 2 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FstStateImpFstStateOr*:
⊢ ▷(*init w*) ⟶ ▷(*init w* ∨ *f*)
proof −
have *1*: ⊢ ▷(*init w*) = (*empty* ∧ *init w*)
  using *FstState* **by** *blast*
have *2*: ⊢ (*empty* ∧ *init w*) = (*empty* ∧ (*empty* ∨ *bi* (¬ *f*);*skip*) ∧ *init w*)
  by *auto*
have *3*: ⊢ (*empty* ∧ (*empty* ∨ *bi* (¬ *f*);*skip*) ∧ *init w*) =
    (*empty* ∧ *bs* (¬ *f*) ∧ *init w*)
  by (*simp add*: *bs-d-def*)
have *4*: ⊢ (*empty* ∧ *bs* (¬ *f*) ∧ *init w*) = (*empty* ∧ *init w* ∧ *bs* (¬ *f*))
  by *auto*
have *5*: ⊢ (*empty* ∧ *init w*) = ▷ (*init w*)
  using *FstState* **by** *fastforce*
hence *6*: ⊢ (*empty* ∧ *init w* ∧ *bs* (¬ *f*)) = (▷ (*init w*) ∧ *bs* (¬ *f*))
  by *auto*
have *7*: ⊢ ▷ (*init w*) ∧ *bs* (¬ *f*) ⟶ (▷ (*init w*) ∧ *bs* (¬ *f*)) ∨ (▷*f* ∧ *bs* (¬(*init w*)))
  by *auto*
have *8*: ⊢ ▷(*init w* ∨ *f*) = ((▷ (*init w*) ∧ *bs* (¬ *f*)) ∨ (▷*f* ∧ *bs* (¬(*init w*))))
  using *FstWithOrEqv* **by** *blast*
from *1 2 3 4 5 6 7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *FstLenSame*:
(∀ σ. (σ ⊨ *di* (▷*f* ∧ *len*(*i*)) ∧ *di* (▷*f* ∧ *len*(*j*))) ⟶ (*i*=*j*))
by (*simp add*: *DiLenFstsem FstLenSamesem*)

lemma *FstLenSame-1*:
⊢ *di* (▷*f* ∧ *len*(*i*)) ∧ *di* (▷*f* ∧ *len*(*j*)) ⟶ (#*i*=#*j*)
using *FstLenSame Valid-def* **by** *fastforce*

lemma *FstAndLenSame*:
(∀ σ. (σ ⊨ *di* ((▷*f* ∧ *g1*) ∧ *len*(*i*)) ∧ *di* ((▷*f* ∧ *g2*) ∧ *len*(*j*))) ⟶ (*i*=*j*))
using *linorder-neqE-nat* **by** (*simp add*: *DiLenFstAndsem*) *blast*

lemma *FstAndLenSame-1*:
⊢ *di* ((▷*f* ∧ *g1*) ∧ *len*(*i*)) ∧ *di* ((▷*f* ∧ *g2*) ∧ *len*(*j*)) ⟶ (#*i*=#*j*)
using *FstAndLenSame Valid-def* **by** *fastforce*

lemma *FstLenSameChop*:
(∀ σ. (σ ⊨ ((▷*f* ∧ *g1*) ∧ *len*(*i*));*h1* ∧ ((▷*f* ∧ *g2*) ∧ *len*(*j*));*h2*) ⟶ (*i*=*j*))
proof

fix *σ*
show (*σ* ⊨ ((▷*f* ∧ *g1*) ∧ *len*(*i*));*h1* ∧ ((▷*f* ∧ *g2*) ∧ *len*(*j*));*h2*) ⟶ (*i*=*j*)
proof
assume *0*: (*σ* ⊨ ((▷*f* ∧ *g1*) ∧ *len*(*i*));*h1* ∧ ((▷*f* ∧ *g2*) ∧ *len*(*j*));*h2*)
have *1*: (*σ* ⊨ ((▷*f* ∧ *g1*) ∧ *len*(*i*));*h1*) **using** *0* **by** *auto*
have *2*: (*σ* ⊨ ((▷*f* ∧ *g1*) ∧ *len*(*i*));*h1*) ⟶
(*σ* ⊨ ((▷*f* ∧ *g1*) ∧ *len*(*i*));#*True*) **by** (*metis ChopImpDi Valid-def di-d-def unl-lift2*)
have *3*: (*σ* ⊨ *di*((▷*f* ∧ *g1*) ∧ *len*(*i*))) **using** *1 2* **by** (*simp add*: *di-d-def*)
have *4*: (*σ* ⊨ ((▷*f* ∧ *g2*) ∧ *len*(*j*));*h2*) **using** *0* **by** *auto*
have *5*: (*σ* ⊨ ((▷*f* ∧ *g2*) ∧ *len*(*j*));*h2*) ⟶
(*σ* ⊨ ((▷*f* ∧ *g2*) ∧ *len*(*j*));#*True*) **by** (*metis ChopImpDi Valid-def di-d-def unl-lift2*)
have *6*: (*σ* ⊨ *di*((▷*f* ∧ *g2*) ∧ *len*(*j*))) **using** *4 5* **by** (*simp add*: *di-d-def*)
have *7*: (*σ* ⊨ *di*((▷*f* ∧ *g1*) ∧ *len*(*i*)) ∧ *di*((▷*f* ∧ *g2*) ∧ *len*(*j*))) **using** *3 6* **by** *auto*
thus (*i*=*j*) **using** *FstAndLenSame* **by** *blast*
qed
qed

lemma *FstLenSameChop-1*:
⊢ ((▷*f* ∧ *g1*) ∧ *len*(*i*));*h1* ∧ ((▷*f* ∧ *g2*) ∧ *len*(*j*));*h2* ⟶ (#*i*=#*j*)
using *FstLenSameChop Valid-def* **by** *fastforce*

lemma *DiImpExistsOneDiLenAndFst*:
(∀ *σ*. (*σ*⊨ *di f*) ⟶ (∃! *k*. (*σ* ⊨ *di*(▷*f* ∧ *len*(*k*)))))
proof
fix *σ*
show (*σ*⊨ *di f*) ⟶ (∃! *k*. (*σ* ⊨ *di*(▷*f* ∧ *len*(*k*))))
proof
assume *0*: (*σ*⊨ *di f*)
have *1*: (*σ* ⊨ *di* (▷ *f*))
using *0 DiEqvDiFst Valid-def* **by** *force*
have *2*: (*σ* ⊨ ▷ *f*) = ((*σ*⊨▷ *f*) ∧ (∃ *k*. (*σ* ⊨ *len*(*k*))))
using *AndExistsLen*[*of TEMP* ▷ *f*] **by** (*simp add*: *Valid-def*)
have *3*: ((*σ*⊨▷ *f*) ∧ (∃ *k*. (*σ* ⊨ *len*(*k*)))) =
(∃ *k*. (*σ*⊨▷ *f*) ∧ (*σ* ⊨ *len*(*k*)))
by *auto*
have *4*: (*σ* ⊨ *di*(▷ *f*)) = (∃ *k*. (*σ* ⊨ *di*(▷*f* ∧ *len*(*k*))))
using *2 3* **by** (*metis 1 DiLensem di-defs*)
have *5*: (∃ *k*. (*σ* ⊨ *di*(▷*f* ∧ *len*(*k*))))
using *1* **using** *4* **by** *auto*
then obtain *i* **where** *6*: (*σ* ⊨ *di*(▷*f* ∧ *len*(*i*))) **by** *blast*
from *5* **obtain** *j* **where** *7*: (*σ* ⊨ *di*(▷*f* ∧ *len*(*j*))) **by** *blast*
have *8*: (*σ* ⊨ *di*(▷*f* ∧ *len*(*i*))) ∧ (*σ* ⊨ *di*(▷*f* ∧ *len*(*j*)))
using *6 7* **by** *auto*
hence *9*: (*σ* ⊨ *di*(▷*f* ∧ *len*(*i*)) ∧ *di*(▷*f* ∧ *len*(*j*)))
by *simp*
hence *10*: *i*=*j*
using *FstLenSame* **by** *blast*
have *11*: ⋀*j*. (*σ* ⊨ *di*(▷*f* ∧ *len*(*j*))) ⟶ (*j*=*i*)
using *9 10* **using** *FstLenSame* **by** *auto*
thus (∃! *k*. (*σ* ⊨ *di*(▷*f* ∧ *len*(*k*))))

using *11 5* **by** *blast*
qed
qed

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

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

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

lemma *LFstAndDistrsem*:
$(\forall\ \sigma.\ (\sigma \models ((\triangleright f \wedge g1);h1 \wedge (\triangleright f \wedge g2);h2) = (\triangleright f \wedge g1 \wedge g2);(h1 \wedge h2)))$
proof
fix σ
show $(\sigma \models ((\triangleright f \wedge g1);h1 \wedge (\triangleright f \wedge g2);h2) = (\triangleright f \wedge g1 \wedge g2);(h1 \wedge h2))$
proof −
have *1*: $(\sigma \models (\triangleright f \wedge g1);h1) = (\exists\ i.\ (\sigma \models ((\triangleright f \wedge g1) \wedge len(i));h1)\ )$
using *AndExistsLenChop*[*of TEMP* $(\triangleright\ f \wedge g1)$] **by** *fastforce*
have *2*: $(\sigma \models (\triangleright f \wedge g2);h2) = (\exists\ j.\ (\sigma \models ((\triangleright f \wedge g2) \wedge len(j));h2)\ )$
using *AndExistsLenChop*[*of TEMP* $(\triangleright f \wedge g2)$] **by** *fastforce*
have *3*: $(\sigma \models (\triangleright f \wedge g1);h1 \wedge (\triangleright f \wedge g2);h2) =$
$(\ (\exists\ i\,j.\ (\sigma \models ((\triangleright f \wedge g1) \wedge len(i));h1\ \wedge$
$((\triangleright f \wedge g2) \wedge len(j));h2)\ )$
$)$
using *1 2* **by** *auto*
have *4*: $(\ (\exists\ i\,j.\ (\sigma \models ((\triangleright f \wedge g1) \wedge len(i));h1\ \wedge$

$((\triangleright f \wedge g2) \wedge len(j));h2)$)

) =

(($\exists$ k. ($\sigma \models ((\triangleright f \wedge g1) \wedge len(k));h1 \wedge$

$((\triangleright f \wedge g2) \wedge len(k));h2)$)

)

using *FstLenSameChop* **by** *blast*

have *5*: ($\exists$ k. ($\sigma \models ((\triangleright f \wedge g1) \wedge len(k));h1 \wedge ((\triangleright f \wedge g2) \wedge len(k));h2$)) =

($\exists$ k. ($\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge len(k));(h1 \wedge h2)$))

using *LFstAndDist-help-1* **by** *blast*

have *6* : ($\exists$ k. ($\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge len(k));(h1 \wedge h2)$)) =

($\sigma \models ((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2));(h1 \wedge h2)$)

using *AndExistsLenChop*[*of TEMP* $((\triangleright f \wedge g1) \wedge \triangleright f \wedge g2)$] **by** *fastforce*

have *7* : ($\sigma \models ((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2));(h1 \wedge h2)$) =

($\sigma \models (\triangleright f \wedge g1 \wedge g2);(h1 \wedge h2)$)

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

from *3 4 5 6 7* **show** *?thesis* **by** *auto*

qed

qed

lemma *LFstAndDistr*:

$\vdash ((\triangleright f \wedge g1);h1 \wedge (\triangleright f \wedge g2);h2) = (\triangleright f \wedge g1 \wedge g2);(h1 \wedge h2)$

using *LFstAndDistrsem* **by** *fastforce*

lemma *LFstAndDistrA*:

$\vdash ((\triangleright f \wedge g1);h \wedge (\triangleright f \wedge g2);h) = (\triangleright f \wedge g1 \wedge g2);h$

proof −

have *1*: $\vdash ((\triangleright f \wedge g1);h \wedge (\triangleright f \wedge g2);h) = (\triangleright f \wedge g1 \wedge g2);(h \wedge h)$ **using** *LFstAndDistr* **by** *blast*

have *2*: $\vdash (\triangleright f \wedge g1 \wedge g2);(h \wedge h) = (\triangleright f \wedge g1 \wedge g2);h$ **by** *auto*

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

qed

lemma *LFstAndDistrB*:

$\vdash ((\triangleright f \wedge g);h1 \wedge (\triangleright f \wedge g);h2) = (\triangleright f \wedge g);(h1 \wedge h2)$

proof −

have *1*: $\vdash ((\triangleright f \wedge g);h1 \wedge (\triangleright f \wedge g);h2) = (\triangleright f \wedge g \wedge g);(h1 \wedge h2)$ **using** *LFstAndDistr* **by** *blast*

have *2*: $\vdash (\triangleright f \wedge g \wedge g);(h1 \wedge h2) = (\triangleright f \wedge g);(h1 \wedge h2)$ **by** *auto*

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

qed

lemma *LFstAndDistrC*:

$\vdash ((\triangleright f\ );h1 \wedge (\triangleright f\ );h2) = (\triangleright f);(h1 \wedge h2)$

proof −

have *1*: $\vdash ((\triangleright f \wedge \#True);h1 \wedge (\triangleright f \wedge \#True);h2) = (\triangleright f \wedge \#True \wedge \#True);(h1 \wedge h2)$

using *LFstAndDistr* **by** *blast*

have *2*: $\vdash (\triangleright f \wedge \#True);h1 = (\triangleright f\ );h1$

by *auto*

have *3*: $\vdash (\triangleright f \wedge \#True);h2 = (\triangleright f\ );h2$

by *auto*

have *4*: $\vdash (\triangleright f \wedge \#True \wedge \#True);(h1 \wedge h2) = (\triangleright f);(h1 \wedge h2)$

by *auto*

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

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

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

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

using *RevChop* **by** *fastforce*
have *5*: $\vdash (h1^r)^r = h1$
using *EqvReverseReverse* **by** *blast*
have *6*: $\vdash (h2^r)^r = h2$
using *EqvReverseReverse* **by** *blast*
have *7*: $\vdash (g1^r)^r = g1$
using *EqvReverseReverse* **by** *blast*
have *8*: $\vdash (g2^r)^r = g2$
using *EqvReverseReverse* **by** *blast*
have *9*: $\vdash (f^r)^r = f$
using *EqvReverseReverse* **by** *blast*
have *10*: $\vdash (\triangleright(f^r) \wedge g1^r)^r = ((\triangleright(f^r))^r \wedge (g1^r)^r)$
using *RAnd* **by** *blast*
have *11*: $\vdash (\triangleright(f^r) \wedge g2^r)^r = ((\triangleright(f^r))^r \wedge (g2^r)^r)$
using *RAnd* **by** *blast*
have *12*: $\vdash (\triangleright(f^r))^r = \triangleleft (f)$
using *RRFirstEqvLast* **by** *blast*

have *13*: $\vdash ((\triangleright(f^r))^r \ \wedge (g1^r)^r) = (\triangleleft\ f \wedge g1)$
  using *12 7* **by** *fastforce*
have *14*: $\vdash ((\triangleright(f^r))^r \ \wedge (g2^r)^r) = (\triangleleft\ f \wedge g2)$
  using *12 8* **by** *fastforce*
have *15*: $\vdash (h1;(\triangleleft\ f \wedge g1) \wedge h2;(\triangleleft\ f \wedge g2)) =$
  $((h1^r)^r;(\triangleright(f^r) \wedge g1^r)^r \wedge (h2^r)^r;(\triangleright(f^r) \wedge (g2^r))^r)$

  using *14 13 10 11 5 6* **by** (*metis 4 int-eq*)
have *16*: $\vdash (((\triangleright(f^r) \wedge (g1^r) \wedge (g2^r));((h1^r) \wedge (h2^r)))^r) =$
  $((h1^r) \wedge (h2^r))^r;((\triangleright\ (f^r)) \wedge (g1^r) \wedge (g2^r))^r$
  by (*simp add*: *RevChop*)
have *17*: $\vdash ((\triangleright\ (f^r)) \wedge (g1^r) \wedge (g2)^r)^r = ((\triangleright\ (f^r))^r \wedge (g1^r)^r \wedge (g2^r)^r)$
  by (*metis inteq-reflection rev-fun2*)
have *18*: $\vdash ((\triangleright\ (f^r))^r \wedge (g1^r)^r \wedge (g2^r)^r) = (\triangleleft\ f \wedge g1 \wedge g2)$
  using *12 7 8* **by** *fastforce*
have *19*: $\vdash ((h1^r) \wedge (h2^r))^r = (h1 \wedge h2)$
  using *RRAnd* **by** *auto*
have *20*: $\vdash ((h1^r) \wedge (h2^r))^r;((\triangleright\ (f^r)) \wedge (g1^r) \wedge (g2^r))^r =$
  $(h1 \wedge h2);(\triangleleft\ f \wedge g1 \wedge g2)$
  using *19 17 18* **using** *ChopEqvChop* **by** (*metis int-eq*)
from *15 4 3 2 16 20* **show** *?thesis* **using** *int-eq* **by** *metis*
qed

lemma *LstAndDistrA*:
$\vdash (h;(\triangleleft\ f \wedge g1) \wedge h;(\triangleleft\ f \wedge g2)) = h;(\triangleleft\ f \wedge g1 \wedge g2)$
proof −
have *1*: $\vdash (h;(\triangleleft\ f \wedge g1) \wedge h;(\triangleleft\ f \wedge g2)) = (h \wedge h);(\triangleleft\ f \wedge g1 \wedge g2)$
  using *LstAndDistr* **by** *blast*
have *2*: $\vdash (h \wedge h);(\triangleleft\ f \wedge g1 \wedge g2) = h;(\triangleleft\ f \wedge g1 \wedge g2)$
  by *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

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

lemma *LstAndDistrC*:
$\vdash\ (h1;(\triangleleft\ f\ ) \wedge h2;(\triangleleft\ f\ )) = (h1 \wedge h2);(\triangleleft\ f\ )$
proof −
have *1*: $\vdash (h1;(\triangleleft\ f \wedge \#True) \wedge h2;(\triangleleft\ f \wedge \#True)) = (h1 \wedge h2);(\triangleleft\ f \wedge \#True \wedge \#True\ \ )$
  using *LstAndDistr* **by** *blast*
have *2*: $\vdash (h1 \wedge h2);(\triangleleft\ f \wedge \#True \wedge \#True\ \ ) = (h1 \wedge h2);(\triangleleft\ f\ )$
  by *auto*

have *3*: $\vdash h1;(\lhd\ f \wedge \#True) = h1;(\lhd\ f\ )$
  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*: $\vdash ((\rhd f;\ (\neg g)) \wedge (\rhd f;g)) = \rhd f;(\neg g \wedge g)$ **using** *LFstAndDistrC* **by** *blast*
have *7*: $\vdash \neg(\rhd f;(\neg g \wedge g))$ **by** (*simp add*: *FstChopFalseEqvFalse*)
have *8*: $\vdash \rhd f;\ (\neg g) \longrightarrow \neg(\rhd f;g)$ **using** *5 6 7* **by** *fastforce*
have *9*: $\vdash \neg(di(\rhd f)) \vee (\rhd f;\ (\neg g)) \longrightarrow \neg(\rhd f;g)$ **using** *4 8* **by** *fastforce*
have *10*: $\vdash di(\rhd f) \vee \neg(di(\rhd f))$ **by** *auto*
hence *11*: $\vdash (\rhd f;\#True) \vee \neg(di(\rhd f))$ **by** (*simp add*: *di-d-def*)
hence *12*: $\vdash (\rhd f;(g \vee \neg g)) \vee \neg(di(\rhd f))$ **by** *auto*
have *13*: $\vdash (\rhd f;(g \vee \neg g)) = ((\rhd f;g) \vee (\rhd f;(\neg g)))$ **using** *ChopOrEqv* **by** *fastforce*
have *14*: $\vdash ((\rhd f;g) \vee (\rhd f;(\neg g))) \vee \neg(di(\rhd f))$ **using** *12 13* **by** *fastforce*
hence *15*: $\vdash \neg(\rhd f;g) \longrightarrow \neg(di(\rhd f)) \vee (\rhd f;\ (\neg g))$ **by** *auto*
from *9 15* **show** *?thesis* **by** *fastforce*
qed

lemma *BsNotFstChop*:
$\vdash bs(\neg(\rhd f;g)) = (empty \vee \neg(di(\rhd f)) \vee (\rhd f;bs(\neg g)))$
proof −
have *1*: $\vdash bs(\neg(\rhd f;g)) = (empty \vee bi(\neg(\rhd f;g));skip)$
  by (*simp add*:*bs-d-def*)
have *2*: $\vdash (empty \vee bi(\neg(\rhd f;g));skip) = (empty \vee (\neg(di(\rhd f;g)));skip)$
  by (*metis 1 NotDiEqvBiNot int-eq*)
have *3*: $\vdash (empty \vee (\neg(di(\rhd f;g)));skip) = (empty \vee (\neg((\rhd f;g);\#True));skip)$

by (*simp add*: *di-d-def*)
have *4*: ⊢ (¬((▷*f*;*g*);#*True*));*skip* = (¬(▷*f*;(*g*;#*True*)));*skip*
by (*metis ChopAssocB LeftChopEqvChop int-simps*(*15*) *inteq-reflection*)
hence *5*: ⊢ (*empty* ∨ (¬((▷*f*;*g*);#*True*));*skip*) = (*empty* ∨ (¬(▷*f*;(*g*;#*True*)));*skip*)
by *auto*
have *6*: ⊢ (*empty* ∨ (¬(▷*f*;(*g*;#*True*)));*skip*) = (*empty* ∨ (¬(▷*f*;*di*(*g*)));*skip*)
by (*simp add*: *di-d-def*)
have *7*: ⊢ (*empty* ∨ (¬(▷*f*;*di*(*g*)));*skip*) = (*empty* ∨ ¬(¬((¬(▷*f*;*di*(*g*)));*skip*)))
by *auto*
have *8*: ⊢ ¬(¬((¬(▷*f*;*di*(*g*)));*skip*)) = (¬(*empty* ∨ (▷*f*;*di*(*g*));*skip*))
using *NotNotChopSkip* **by** *fastforce*
hence *9*: ⊢ (*empty* ∨ ¬(¬((¬(▷*f*;*di*(*g*)));*skip*))) = (*empty* ∨ ¬(*empty* ∨ (▷*f*;*di*(*g*));*skip*))
by *auto*
have *10*: ⊢ (*empty* ∨ ¬(*empty* ∨ (▷*f*;*di*(*g*));*skip*)) = (*empty* ∨ (*more* ∧ ¬((▷*f*;*di*(*g*));*skip*)))
by (*meson 6 7 9 NotChopSkipEqvMoreAndNotChopSkip Prop04 Prop06*)
have *11*: ⊢ (*empty* ∨ (*more* ∧ ¬((▷*f*;*di*(*g*));*skip*))) = (*empty* ∨ ¬((▷*f*;*di*(*g*));*skip*))
by (*auto simp add*: *empty-d-def*)
have *12*: ⊢ (*empty* ∨ ¬((▷*f*;*di*(*g*));*skip*)) = (*empty* ∨ ¬(▷*f*;(*di*(*g*);*skip*)))
using *ChopAssocB 11* **by** *fastforce*
have *13*: ⊢ (¬(▷*f*;(*di*(*g*);*skip*))) = (¬(▷*f*;(*ds*(*g*))))
using *DsDi* **using** *RightChopEqvChop* **by** *fastforce*
hence *14*: ⊢ (*empty* ∨ ¬(▷*f*;(*di*(*g*);*skip*))) = (*empty* ∨ ¬(▷*f*;(*ds*(*g*))))
by *auto*
have *15*: ⊢ (*empty* ∨ ¬(▷*f*;(*ds*(*g*)))) = (*empty* ∨ ¬(*di* (▷*f*)) ∨ (▷*f*;(¬(*ds g*))))
using *NotFstChop* **by** *fastforce*
have *16*: ⊢ (▷*f*;(¬(*ds g*)))= (▷*f*;(*bs* (¬*g*)))
using *NotDsEqvBsNot RightChopEqvChop* **by** *blast*
hence *17*: ⊢ ((*empty* ∨ ¬(*di* (▷*f*))) ∨ (▷*f*;(¬(*ds g*)))) = ((*empty* ∨ ¬(*di* (▷*f*))) ∨ (▷*f*;(*bs* (¬*g*))))
by *auto*
from *1 2 3 5 6 7 9 10 11 12 14 15 17* **show** *?thesis* **by** *fastforce*
qed

lemma *FstFstChopEqvFstChopFst*:
⊢ ▷(▷*f*;*g*) = ▷*f*;▷*g*
proof −
have *1*: ⊢ ▷(▷*f*;*g*) = ((▷*f*;*g*) ∧ *bs* (¬(▷*f*;*g*)))
by (*simp add*: *first-d-def*)
have *2*: ⊢ *bs* (¬(▷*f*;*g*)) = (*empty* ∨ ¬(*di*(▷*f*)) ∨ (▷*f*;*bs*(¬*g*)))
using *BsNotFstChop* **by** *auto*
hence *3*: ⊢ ((▷*f*;*g*) ∧ *bs* (¬(▷*f*;*g*))) = ((▷*f*;*g*) ∧ (*empty* ∨ ¬(*di*(▷*f*)) ∨ (▷*f*;*bs*(¬*g*))))
by *auto*
have *4*: ⊢ ((▷*f*;*g*) ∧ (*empty* ∨ ¬(*di*(▷*f*)) ∨ (▷*f*;*bs*(¬*g*)))) =
(((▷*f*;*g*) ∧ *empty*) ∨ ((▷*f*;*g*) ∧ ¬(*di*(▷*f*))) ∨ ((▷*f*;*g*) ∧ (▷*f*;*bs*(¬*g*))))
by *auto*
have *5*: ⊢ ¬((▷*f*;*g*) ∧ ¬(*di*(▷*f*)))
using *ChopImpDi* **by** *fastforce*
hence *6*: ⊢ (((▷*f*;*g*) ∧ *empty*) ∨ ((▷*f*;*g*) ∧ ¬(*di*(▷*f*))) ∨ ((▷*f*;*g*) ∧ (▷*f*;*bs*(¬*g*)))) =
(((▷*f*;*g*) ∧ *empty*) ∨ ((▷*f*;*g*) ∧ (▷*f*;*bs*(¬*g*))))
by *auto*
have *7*: ⊢ ((▷*f*;*g*) ∧ (▷*f*;(*bs*(¬*g*)))) = ((▷*f*;(*g* ∧ (*bs*(¬*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*: $\vdash$ ($\triangleright$ *f*) ;(*power* ($\triangleright$*f*) *n*) = ($\triangleright$ *f*) ; $\triangleright$ (*power* ($\triangleright$ *f*) *n*)
using *RightChopEqvChop Suc.hyps* **by** *blast*
have *6*: $\vdash$ ($\triangleright$ *f*) ; $\triangleright$ (*power* ($\triangleright$ *f*) *n*) = $\triangleright$($\triangleright$*f*;(*power* ($\triangleright$ *f*) *n*))
using *FstFstChopEqvFstChopFst* **by** *fastforce*
have *7*: $\vdash$ $\triangleright$($\triangleright$*f*;(*power* ($\triangleright$ *f*) *n*)) = $\triangleright$(*power* ($\triangleright$ *f*) (*Suc n*))
by *simp*
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed
qed

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

lemma *FstLenAndEqvLenAnd*:
$\vdash$ $\triangleright$(*len*(*k*) $\wedge$ *f*) = (*len*(*k*) $\wedge$ *f*)
proof –
have *1*: $\vdash$ *len*(*k*) $\wedge$ *f* $\wedge$ *ds*(*len*(*k*)$\wedge$ *f*) $\longrightarrow$ *ds* (*len*(*k*))
using *DsAndImpElimL* **by** *fastforce*
hence *2*: $\vdash$ *len*(*k*) $\wedge$ *f* $\wedge$ *ds*(*len*(*k*)$\wedge$ *f*) $\longrightarrow$ (*di*(*len*(*k*)));*skip*
using *DsDi* **by** *fastforce*
hence *3*: $\vdash$ *len*(*k*) $\wedge$ *f* $\wedge$ *ds*(*len*(*k*)$\wedge$ *f*) $\longrightarrow$ ((*len*(*k*);#*True*));*skip*
by (*simp add*: *di-d-def*)
hence *4*: $\vdash$ *len*(*k*) $\wedge$ *f* $\wedge$ *ds*(*len*(*k*)$\wedge$ *f*) $\longrightarrow$ (*len*(*k*);(#*True*;*skip*))
using *ChopAssocB* **by** *fastforce*
hence *5*: $\vdash$ *len*(*k*) $\wedge$ *f* $\wedge$ *ds*(*len*(*k*)$\wedge$ *f*) $\longrightarrow$ (*len*(*k*);(*skip*;#*True*))
using *SkipTrueEqvTrueSkip TrueChopSkipEqvSkipChopTrue RightChopEqvChop* **by** *fastforce*
hence *6*: $\vdash$ *len*(*k*) $\wedge$ *f* $\wedge$ *ds*(*len*(*k*)$\wedge$ *f*) $\longrightarrow$ (*len*(*k*);(*skip*;#*True*)) $\wedge$ *len*(*k*)
by *auto*
hence *7*: $\vdash$ *len*(*k*) $\wedge$ *f* $\wedge$ *ds*(*len*(*k*)$\wedge$ *f*) $\longrightarrow$ (*len*(*k*);(*skip*;#*True*)) $\wedge$ *len*(*k*);*empty*
using *ChopEmpty* **by** (*metis int-eq*)
hence *8*: $\vdash$ *len*(*k*) $\wedge$ *f* $\wedge$ *ds*(*len*(*k*)$\wedge$ *f*) $\longrightarrow$ (*len*(*k*);((*skip*;#*True*) $\wedge$ *empty*))
using *LFixedAndDistrB1* **by** *fastforce*
have *9*: $\vdash$ $\neg$(*len*(*k*);((*skip*;#*True*) $\wedge$ *empty*))
by (*simp add*: *empty-d-def more-d-def next-d-def chop-defs Valid-def*)
have *10*: $\vdash$ *len*(*k*) $\wedge$ *f* $\longrightarrow$ $\neg$(*ds*(*len*(*k*)$\wedge$ *f*))
using *8 9* **by** *fastforce*
hence *11*: $\vdash$ *len*(*k*) $\wedge$ *f* $\longrightarrow$ *bs* ($\neg$(*len*(*k*)$\wedge$ *f*))
using *NotDsEqvBsNot* **by** *fastforce*
hence *12*: $\vdash$ *len*(*k*) $\wedge$ *f* $\longrightarrow$ (*len*(*k*) $\wedge$ *f*) $\wedge$ *bs* ($\neg$(*len*(*k*)$\wedge$ *f*))
by *auto*
hence *13*: $\vdash$ *len*(*k*) $\wedge$ *f* $\longrightarrow$ $\triangleright$(*len*(*k*) $\wedge$ *f*)
by (*simp add*: *first-d-def*)
have *14*: $\vdash$ $\triangleright$(*len*(*k*) $\wedge$ *f*) $\longrightarrow$ *len*(*k*) $\wedge$ *f*
by (*auto simp add*: *first-d-def*)

from *13 14* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndElimL*:
$\vdash \rhd f \longrightarrow f$
by (*auto simp add*: *first-d-def*)

lemma *FstImpNotDiChopSkip*:
$\vdash \rhd f \longrightarrow \neg(di\ f;skip)$
proof –
have *1*: $\vdash \rhd f \longrightarrow bs\ (\neg\ f)$ **by** (*auto simp add*: *first-d-def*)
hence *2*: $\vdash \rhd f \longrightarrow \neg(ds\ f)$ **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*:
$\vdash \rhd f \longrightarrow \neg(di\ (f;skip))$
proof –
have *1*: $\vdash \rhd f \longrightarrow bs\ (\neg\ f)$
by (*auto simp add*: *first-d-def*)
hence *2*: $\vdash \rhd f \longrightarrow \neg(ds\ f)$
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*: $\vdash di\ f \longrightarrow f \lor (di\ f;skip)$ **using** *DiEqvOrDiChopSkipB* **by** *fastforce*
have *3*: $\vdash \rhd f\ \land di\ f \longrightarrow (f \lor (di\ f;skip)) \land \neg(di\ f;skip)$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash ((f \lor (di\ f;skip)) \land \neg(di\ f;skip)) = (f\ \land \neg(di\ f;skip))$ **by** *auto*
have *5*: $\vdash\ \rhd f\ \land di\ f \longrightarrow f\ \land \neg(di\ f;skip)$ **using** *3 4* **by** *fastforce*
hence *6*: $\vdash\ \rhd f\ \land di\ f \longrightarrow f$ **by** *fastforce*

hence *7*: $\vdash \triangleright f \longrightarrow (di\ f \longrightarrow f)$ **using** *FstAndElimL* **by** *fastforce*
have *8*: $\vdash f \longrightarrow di\ f$ **using** *DiIntro* **by** *auto*
hence *9*: $\vdash \triangleright f \longrightarrow (f \longrightarrow (di\ f))$ **by** *auto*
from *7 9* **show** *?thesis* **by** *fastforce*
qed

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

lemma *FstAndDiImpBsNotAndDi*:
$\vdash (\triangleright f \wedge di\ g) \longrightarrow (bs\ (\neg(di\ f \wedge g)))$
proof –
have *1*: $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds(di\ f \wedge g)$
by (*auto simp add*: *ds-d-def*)
hence *2*: $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds(di\ f)$

using *DsAndImp* **by** *fastforce*
hence *3*: $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow di(di\ f);skip$
using *DsDi* **by** *fastforce*
hence *4*: $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow di\ f;skip$
using *DiEqvDiDi* **by** (*metis int-eq*)
hence *5*: $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds\ f$
using *DsDi* **by** *fastforce*
hence *6*: $\vdash\ (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow \neg\ (\triangleright\ f)$
using *DsImpNotFst* **by** *fastforce*
from *6* **show** *?thesis* **by** *auto*
qed

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

hence *15*: $\vdash ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f) \wedge \neg\ g)) = ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f \vee g)))$
by *auto*
have *16*: $\vdash ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f \vee g))) = \triangleright(\triangleright f \vee g)$
by (*simp add*: *first-d-def*)
from *16 15 12 11 10 9 8 7 6 5 1* **show** *?thesis* **by** (*metis int-eq*)
qed

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

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

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

lemma *FstSkip*:
$\vdash \triangleright\ skip = skip$
proof −
have *1*: $\vdash skip = len(1)$ **using** *LenOneEqvSkip* **by** *fastforce*
hence *2*: $\vdash \triangleright skip = \triangleright\ (len(1))$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \triangleright(\ len(1)) = len(1)$ **using** *FstLenEqvLen* **by** *blast*
from *1 2 3* **show** *?thesis* **using** *LenOneEqvSkip* **by** *fastforce*
qed

lemma *HaltStateEqvFstFinState*:
$\vdash halt\ (init\ w) = \triangleright\ (fin\ (init\ w))$
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 (*auto simp add*: *empty-d-def*)
hence *5*: $\vdash \Box\ ((init\ w) \longrightarrow empty) = \Box\ (more \longrightarrow \neg(init\ w))$ **using** *BoxEqvBox* **by** *blast*
have *6*: $\vdash \Box\ (more \longrightarrow \neg(init\ w)) = bs(\neg(fin(init\ w)))$ **using** *BoxMoreStateEqvBsFinState* **by** *blast*
have *7*: $\vdash \Box((empty \longrightarrow (init\ w))) = fin(init\ w)$ **by** (*simp add*: *fin-d-def*)
have *8*: $\vdash (\Box((empty \longrightarrow (init\ w))) \wedge \Box((init\ w) \longrightarrow empty)) =$
  $(fin(init\ w) \wedge bs(\neg(fin(init\ w))))$ **using** *5 6 7* **by** *fastforce*
from *1 2 3 8* **show** *?thesis* **by** (*metis first-d-def inteq-reflection*)
qed

lemma *FstLenEqvLenFst*:
$\vdash \rhd(len\ k\ ;\ f) = len\ k\ ;\ \rhd\ f$
proof –
have *1*: $\vdash len\ k\ ;\ f = \rhd(len\ k)\ ;\ f$ **using** *FstLenEqvLen LeftChopEqvChop* **by** *fastforce*
have *2*: $\vdash \rhd(len\ k\ ;\ f) = \rhd\ (\rhd(len\ k)\ ;\ f)$ **using** *1 FstEqvRule* **by** *blast*
have *3*: $\vdash \rhd\ (\rhd(len\ k)\ ;\ f) = \rhd(len\ k)\ ;\ \rhd f$ **using** *FstFstChopEqvFstChopFst* **by** *blast*
have *4*: $\vdash \rhd(len\ k)\ ;\ \rhd f = len\ k\ ;\ \rhd f$ **using** *FstLenEqvLen LeftChopEqvChop* **by** *fastforce*
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

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

lemma *FstDiamondStateEqvHalt*:
$\vdash \rhd\ (\Diamond\ (init\ w)) = halt\ (init\ w)$
proof –
have *1*: $\vdash \Diamond\ (init\ w) = \Diamond\ ((init\ w) \wedge \#True)$ **by** *simp*
have *2*: $\vdash fin\ (init\ w)\ ;\ \#True = \Diamond\ ((init\ w) \wedge \#True)$ **using** *1 FinChopEqvDiamond* **by** *blast*
have *3*: $\vdash fin\ (init\ w)\ ;\ \#True = di\ (fin\ (init\ w))$ **by** (*simp add*: *di-d-def*)
have *4*: $\vdash (\Diamond\ (init\ w)) = (di\ (fin\ (init\ w)))$ **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash \rhd\ (\Diamond\ (init\ w)) = \rhd\ (di\ (fin\ (init\ w)))$ **using** *4 FstEqvRule* **by** *blast*
hence *6*: $\vdash \rhd\ (\Diamond\ (init\ w)) = \rhd\ (fin\ (init\ w))$ **using** *FstDiEqvFst* **by** *fastforce*
hence *7*: $\vdash \rhd\ (\Diamond\ (init\ w)) = halt\ (init\ w)$ **using** *HaltStateEqvFstFinState* **by** *fastforce*
from *7* **show** *?thesis* **by** *simp*
qed

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

lemma *FstAndFstStarEqvFst*:
$\vdash (\rhd f \wedge (\rhd f)^{\star}) = \rhd f$
proof −
  have *1*: $\vdash (\rhd f)^{\star} = (empty \vee (\rhd f);(\rhd f)^{\star})$
      using *CSEqvOrChopCS* **by** *fastforce*
  have *2*: $\vdash ((\rhd f)^{\star} \wedge \rhd f) = ((empty \vee (\rhd f);(\rhd f)^{\star}) \wedge \rhd f)$
      using *1* **by** *fastforce*
  have *3*: $\vdash ((empty \vee (\rhd f);(\rhd f)^{\star}) \wedge \rhd f) = ((empty \wedge \rhd f) \vee ((\rhd f);(\rhd f)^{\star} \wedge \rhd f))$
      by *auto*
  have *4*: $\vdash ((\rhd f)^{\star} \wedge \rhd f) = ((empty \wedge \rhd f) \vee ((\rhd f);(\rhd f)^{\star} \wedge \rhd f))$
      using *2 3* **by** *fastforce*
  have *5*: $\vdash ((\rhd f);(\rhd f)^{\star} \wedge \rhd f) = ((\rhd f);(\rhd f)^{\star} \wedge \rhd f;empty)$
      using *ChopEmpty* **by** (*metis inteq-reflection*)
  have *6*: $\vdash ((\rhd f);(\rhd f)^{\star} \wedge \rhd f;empty) = (\rhd f);((\rhd f)^{\star} \wedge empty)$
      using *LFstAndDistrC* **by** *blast*
  have *7*: $\vdash ((\rhd f)^{\star} \wedge empty) = empty$
      using *EmptyImpCS* **by** *fastforce*
  have *8*: $\vdash (\rhd f);((\rhd f)^{\star} \wedge empty) = \rhd f$

using *7 ChopEmpty* **by** (*metis inteq-reflection*)
have *9*: $\vdash ((\rhd f);(\rhd f)^\star \wedge \rhd f) = \rhd f$
using *5 6 8* **by** *fastforce*
have *10*: $\vdash ((\rhd f)^\star \wedge \rhd f) = ((empty \wedge \rhd f) \vee \rhd f)$
using *4 9* **by** *fastforce*
have *11*: $\vdash ((empty \wedge \rhd f) \vee \rhd f) = \rhd f$
by *auto*
have *12*: $\vdash ((\rhd f)^\star \wedge \rhd f) = \rhd f$
using *10 11* **by** *fastforce*
from *12* **show** *?thesis* **by** *auto*
qed

lemma *HaltStateEqvFstHaltState*:
$\vdash halt(init(w)) = \rhd(halt(init(w)))$
proof −
have *1*: $\vdash halt\ (init\ w) = \rhd\ (fin\ (init\ w))$
by (*simp add*: *HaltStateEqvFstFinState*)
have *2*: $\vdash \rhd\ (fin\ (init\ w)) = \rhd\ (\rhd\ (fin\ (init\ w)))$
using *FstEqvRule FstFixFst* **by** *fastforce*
have *3*: $\vdash \rhd\ (\rhd\ (fin\ (init\ w))) = \rhd(halt(init(w)))$
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*: $\vdash (di(halt\ (init\ w) \wedge f) \wedge di(halt\ (init\ w) \wedge g)) =$
$(di(\rhd(fin\ (init\ w)) \wedge f) \wedge di\ (\rhd(fin\ (init\ w)) \wedge g))$
using *HaltStateEqvFstFinState* **by** (*metis LFstAndDistrD inteq-reflection*)
have *2*: $\vdash (di(\rhd(fin\ (init\ w)) \wedge f) \wedge di(\rhd(fin\ (init\ w)) \wedge g)) =$
$di(\rhd(fin\ (init\ w)) \wedge f \wedge g)$
using *LFstAndDistrD* **by** *fastforce*
have *3*: $\vdash di(\rhd(fin\ (init\ w)) \wedge f \wedge g) = di(halt\ (init\ w) \wedge f \wedge g)$
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*:
$\vdash ((\rhd(len(5)) \wedge \rhd(len(2)))\ ;\ (len(5) \vee len(2))) = \#False$
proof −
have *1*: $\vdash ((\rhd(len(5)) \wedge \rhd(len(2)))\ ;\ (len(5) \vee len(2))) =$
$(len(5) \wedge len(2)); (len(5) \vee len(2))$
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

17 Monitors

theory *Monitor*
imports *First*

begin

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

17.1 Syntax

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

fun *MON* :: ($'a$::*world*) *monitor* $\Rightarrow$ $'a$ *formula*
where (*MON* (*FIRST* *f*)) = *LIFT*($\triangleright$ *f*)
  | (*MON* (*a* *UPTO* *b*)) = *LIFT*($\triangleright$((*MON* *a*) $\vee$ (*MON* *b*)))
  | (*MON* (*a* *THRU* *b*)) = *LIFT*($\triangleright$(*di*(*MON* *a*) $\wedge$ *di*(*MON* *b*)))
  | (*MON* (*a* *THEN* *b*)) = *LIFT*((*MON* *a*);(*MON* *b*))
  | (*MON* (*a* *WITH* *f*)) = *LIFT*((*MON* *a*) $\wedge$ *f*)

syntax
 -MON :: $'a$ *monitor* $\Rightarrow$ *lift* (($\mathcal{M}$ -) [*80*] *80*)

translations
 -MON == *CONST MON*

definition *eq-d* :: ($'a$:: *world*) *monitor* $\Rightarrow$ $'a$ *monitor* $\Rightarrow$ *bool* ((- $\simeq$ -) [*84*,*84*] *83*)
where
 eq-d a b $\equiv$ ($\vdash$ ($\mathcal{M}$ *a*) = ($\mathcal{M}$ *b*))

lemma *MonEqRefl*:
 $a \simeq a$
by (*simp add*: *eq-d-def*)

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

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

lemma *MonEq*:
$(a \simeq b) = (\vdash (\mathcal{M}\ a) = (\mathcal{M}\ b))$
by (*simp add*: *eq-d-def*)

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

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

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

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

17.2 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 $\equiv$ *LIFT*(*bs* ($\neg$ *f*))

definition *UNTIL-d* :: (*'a* :: *world*) *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a monitor*
where
UNTIL-d w1 w2 $\equiv$ (*HALT w2*) *WITH* (*LIFT*(*bm w1*))

syntax
-*FAIL-d* :: *'a monitor* (*FAIL*)
-*ALWAYS-d* :: [*'a monitor*,*lift*] $\Rightarrow$ *'a monitor* ((- *ALWAYS* -) [*84*,*84*] *83*)
-*SOMETIME-d* :: [*'a monitor*,*lift*] $\Rightarrow$ *'a monitor* ((- *SOMETIME* -) [*84*,*84*] *83*)
-*LIMIT-d* :: *lift* $\Rightarrow$ *lift* ((*Limit* -) [*84*] *83*)
-*UNTIL-d* :: [*lift*,*lift*] $\Rightarrow$ *'a monitor* ((- *UNTIL* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-*FAIL-d* :: *'a monitor* (*FAIL*)
-*ALWAYS-d* :: [*'a monitor*,*lift*] $\Rightarrow$ *'a monitor* ((- *ALWAYS* -) [*84*,*84*] *83*)
-*SOMETIME-d* :: [*'a monitor*,*lift*] $\Rightarrow$ *'a monitor* ((- *SOMETIME* -) [*84*,*84*] *83*)
-*LIMIT-d* :: *lift* $\Rightarrow$ *lift* ((*Limit* -) [*84*] *83*)
-*UNTIL-d* :: [*lift*,*lift*] $\Rightarrow$ *'a monitor* ((- *UNTIL* -) [*84*,*84*] *83*)

translations
-*FAIL-d* $\rightleftharpoons$ *CONST FAIL-d*
-*ALWAYS-d* $\rightleftharpoons$ *CONST ALWAYS-d*
-*SOMETIME-d* $\rightleftharpoons$ *CONST SOMETIME-d*
-*LIMIT-d* $\rightleftharpoons$ *CONST LIMIT-d*
-*UNTIL-d* $\rightleftharpoons$ *CONST UNTIL-d*

definition *WITHIN-d* :: (*'a* :: *world*) *monitor* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a monitor*
where
WITHIN-d a f $\equiv$ (*a WITH LIFT*(*Limit f*))

syntax
-*WITHIN-d* :: [*'a monitor*,*lift*] $\Rightarrow$ *'a monitor* ((- *WITHIN* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-*WITHIN-d* :: [*'a monitor*,*lift*] $\Rightarrow$ *'a monitor* ((- *WITHIN* -) [*84*,*84*] *83*)

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

definition *AND-d* :: (*'a* :: *world*) *monitor* $\Rightarrow$ *'a monitor* $\Rightarrow$ *'a monitor*
where
AND-d a b $\equiv$ (*a WITH LIFT*($\mathcal{M}$ *b*))

definition *ITERATE-d* :: (*'a* :: *world*) *monitor* $\Rightarrow$ *'a monitor* $\Rightarrow$ *'a monitor*
where
ITERATE-d a b $\equiv$ (*a WITH* (*LIFT* ($\mathcal{M}$ *b*)*))

syntax
-AND-d :: [*'a monitor*,*'a monitor*] ⇒ *'a monitor* ((- *AND* -) [*84*,*84*] *83*)
-ITERATE-d :: [*'a monitor*,*'a monitor*] ⇒ *'a monitor* ((- *ITERATE* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-AND-d :: [*'a monitor*,*'a monitor*] ⇒ *'a monitor* ((- *AND* -) [*84*,*84*] *83*)
-ITERATE-d :: [*'a monitor*,*'a monitor*] ⇒ *'a monitor* ((- *ITERATE* -) [*84*,*84*] *83*)

translations
-AND-d ⇌ *CONST AND-d*
-ITERATE-d ⇌ *CONST ITERATE-d*

definition *STAR-d* :: (*'a* :: *world*) *monitor* ⇒ *'a formula* ⇒ *'a monitor*
where
STAR-d a f ≡ ((*FIRST LIFT*(◇ *f*)) *ITERATE* (*a*))

definition *REPEAT-d* :: (*'a* :: *world*) *monitor* ⇒ *'a formula* ⇒ *'a monitor*
where
REPEAT-d a w ≡ ((*HALT w*) *ITERATE* (*a WITH LIFT*(*keep*(¬ (*init w*)))))

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*

17.3 Monitor Laws

lemma *MFixFst*:
$\vdash (\mathcal{M}\ a) = \triangleright\ (\mathcal{M}\ a)$
proof
(*induct a*)
case (*mFIRST-d x*)
then show *?case*
proof −
have *1*: $\vdash (\mathcal{M}\ (FIRST\ x)) = \triangleright\ x$ **by** *simp*
have *2*: $\vdash \triangleright\ x = \triangleright\ (\triangleright\ x)$ **using** *FstFixFst* **by** *fastforce*
have *3*: $\vdash\ \triangleright\ (\triangleright\ x) = \triangleright(\mathcal{M}\ (FIRST\ x))$ **by** *simp*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed
next

case (*mUPTO-d a1 a2*)
then show *?case*
proof −
have *1*: $\vdash (\mathcal{M}\ (a1\ UPTO\ a2)) = \rhd(\ (\mathcal{M}\ a1) \vee (\mathcal{M}\ a2))$
by (*simp*)
have *2*: $\vdash\ \rhd(\ (\mathcal{M}\ a1) \vee (\mathcal{M}\ a2)) = \rhd(\rhd((\mathcal{M}\ a1) \vee (\mathcal{M}\ a2)))$
using *FstFixFst* **by** *fastforce*
have *3*: $\vdash \rhd(\rhd((\mathcal{M}\ a1) \vee (\mathcal{M}\ a2))) = \rhd(\mathcal{M}\ (a1\ UPTO\ a2))$
using *2* **by** *simp*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed
next
case (*mTHRU-d a1 a2*)
then show *?case*
proof −
have *1*: $\vdash (\mathcal{M}\ (a1\ THRU\ a2)) = \rhd(\ di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2))$
by (*simp*)
have *2*: $\vdash \rhd(\ di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2)) = \rhd(\rhd(di(\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2)))$
using *FstFixFst* **by** *fastforce*
have *3*: $\vdash \rhd(\rhd(\ di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2))) = \rhd(\mathcal{M}\ (a1\ THRU\ a2))$
using *2* **by** *simp*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed
next
case (*mTHEN-d a1 a2*)
then show *?case*
proof −
have *1*: $\vdash (\mathcal{M}\ (a1\ THEN\ a2)) = (\mathcal{M}\ a1)\ ;\ (\mathcal{M}\ a2)$
by (*simp*)
have *2*: $\vdash (\mathcal{M}\ a1)\ ;\ (\mathcal{M}\ a2) = \rhd(\mathcal{M}\ a1)\ ;\ \rhd(\mathcal{M}\ a2)$
using *ChopEqvChop mTHEN-d.hyps*(*1*) *mTHEN-d.hyps*(*2*) **by** *blast*
have *3*: $\vdash \rhd(\mathcal{M}\ a1)\ ;\ \rhd(\mathcal{M}\ a2) = \rhd(\rhd(\mathcal{M}\ a1)\ ;\ (\mathcal{M}\ a2))$
using *FstFstChopEqvFstChopFst* **by** *fastforce*
have *4*: $\vdash \rhd(\rhd(\mathcal{M}\ a1)\ ;\ (\mathcal{M}\ a2)) = \rhd((\mathcal{M}\ a1)\ ;\ (\mathcal{M}\ a2))$
using *FstEqvRule LeftChopEqvChop mTHEN-d.hyps*(*1*) **by** (*metis inteq-reflection*)
have *5*: $\vdash \rhd((\mathcal{M}\ a1)\ ;\ (\mathcal{M}\ a2)) = \rhd(\mathcal{M}\ (a1\ THEN\ a2))$
using *4* **by** *simp*
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed
next
case (*mWITH-d a x2*)
then show *?case*
proof −
have *1*: $\vdash (\mathcal{M}\ (a\ WITH\ x2)) = ((\mathcal{M}\ a) \wedge (\ x2))$
by (*simp*)
have *2*: $\vdash ((\mathcal{M}\ a) \wedge (\ x2)) = (\rhd(\mathcal{M}\ a) \wedge (\ x2))$
using *mWITH-d.hyps* **by** *fastforce*
have *3*: $\vdash (\rhd(\mathcal{M}\ a) \wedge (\ x2)) = \rhd(\rhd(\mathcal{M}\ a) \wedge (\ x2))$
using *FstFstAndEqvFstAnd* **by** *fastforce*
have *4*: $\vdash \rhd(\rhd(\mathcal{M}\ a) \wedge (\ x2)) = \rhd((\mathcal{M}\ a) \wedge (\ x2))$

using *2 FstEqvRule* **by** *fastforce*
have *5*: $\vdash \triangleright((\mathcal{M}\ a) \wedge (\ x2)) = \triangleright(\mathcal{M}\ (a\ WITH\ x2))$
using *4* **by** *simp*
from *1 2 3 4 5* **show** *?thesis* **by** (*metis inteq-reflection*)
qed
qed

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

lemma *MFirstFalseEqvFalse*:
$\vdash\ \mathcal{M}(FIRST\ LIFT\ \#False) = \#False$
proof −
have *1*: $\vdash \mathcal{M}(FIRST\ LIFT\ \#False) = \triangleright\ \#False$ **by** (*simp*)
have *2*: $\vdash \mathcal{M}(FIRST\ LIFT\ \#False) = \#False$ **using** *FstFalse* **by** *fastforce*
from *2* **show** *?thesis* **by** *auto*
qed

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

lemma *MFailEqvFirstFalseWithinEmpty*:
$FAIL \simeq\ ((FIRST\ LIFT\ \#False)\ WITHIN\ empty)$
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*: $\vdash\ (empty \wedge (\ init\ w)) = (empty \wedge init\ w)$ **by** *simp*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

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

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

using *BoxStateEqvBiFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

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

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

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

by *simp*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *MUptoAlt*:
$\vdash \mathcal{M}(a\ UPTO\ b) = ((\ (\mathcal{M}\ a) \wedge bi\ (\neg(\mathcal{M}\ b))) \vee ((\mathcal{M}\ b) \wedge bi\ (\neg(\mathcal{M}\ a))) \vee ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b)))$
proof −
have *1*: $\vdash \mathcal{M}\ (a\ UPTO\ b) = \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))$
by (*simp*)
have *2*: $\vdash \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = ((\triangleright(\mathcal{M}\ a) \wedge\ (bs\ (\neg(\mathcal{M}\ b)))) \vee (\triangleright(\mathcal{M}\ b) \wedge\ (bs\ (\neg(\mathcal{M}\ a)))))$
using *FstWithOrEqv* **by** *blast*
have *3*: $\vdash ((\triangleright(\mathcal{M}\ a) \wedge\ (bs\ (\neg(\mathcal{M}\ b)))) \vee (\triangleright(\mathcal{M}\ b) \wedge\ (bs\ (\neg(\mathcal{M}\ a))))) =$
$(((\mathcal{M}\ a) \wedge ((\mathcal{M}\ b) \vee \neg(\mathcal{M}\ b)) \wedge\ (bs\ (\neg(\mathcal{M}\ b)))) \vee$
$((\mathcal{M}\ b) \wedge ((\mathcal{M}\ a) \vee \neg(\mathcal{M}\ a)) \wedge\ (bs\ (\neg(\mathcal{M}\ a)))))$
using *MFixFst* **by** *fastforce*
have *4*: $\vdash (((\mathcal{M}\ a) \wedge ((\mathcal{M}\ b) \vee \neg(\mathcal{M}\ b)) \wedge\ (bs\ (\neg(\mathcal{M}\ b)))) \vee$
$((\mathcal{M}\ b) \wedge ((\mathcal{M}\ a) \vee \neg(\mathcal{M}\ a)) \wedge\ (bs\ (\neg(\mathcal{M}\ a))))) =$
$(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ ))$
by *auto*
have *5*: $\vdash (((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ )) =$
$(((\mathcal{M}\ a) \wedge (\ (\triangleright(\mathcal{M}\ b)) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ (\triangleright(\mathcal{M}\ a)) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ ))$
by (*simp add*: *first-d-def*)
have *6*: $\vdash (((\mathcal{M}\ a) \wedge (\ (\triangleright(\mathcal{M}\ b)) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ (\triangleright(\mathcal{M}\ a)) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ )) =$
$(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ ))$
using *MFixFst* **by** *fastforce*
have *7*: $\vdash (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))) = bi(\neg(\mathcal{M}\ b))$
using *AndBsEqvBi* **by** *blast*
have *8*: $\vdash (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) = bi(\neg(\mathcal{M}\ a))$
using *AndBsEqvBi* **by** *blast*
have *9*: $\vdash (((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee ((\neg(\mathcal{M}\ b)) \wedge bs(\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee ((\neg(\mathcal{M}\ a)) \wedge bs(\neg(\mathcal{M}\ a))))\ )) =$
$(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee (\ bi(\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\ bi(\neg(\mathcal{M}\ a))))\ ))$
using *7 8* **by** *fastforce*
have *10*: $\vdash(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee (\ bi(\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\ bi(\neg(\mathcal{M}\ a))))\ )) =$
$(((\ (\mathcal{M}\ a) \wedge (\mathcal{M}\ b)) \vee (\ (\mathcal{M}\ a) \wedge bi(\neg(\mathcal{M}\ b)))) \vee$
$((\ (\mathcal{M}\ b) \wedge (\mathcal{M}\ a)) \vee (\ (\mathcal{M}\ b) \wedge bi(\neg(\mathcal{M}\ a)))))$
by *auto*
have *11*: $\vdash (((\ (\mathcal{M}\ a) \wedge (\mathcal{M}\ b)) \vee (\ (\mathcal{M}\ a) \wedge bi(\neg(\mathcal{M}\ b)))) \vee$
$((\ (\mathcal{M}\ b) \wedge (\mathcal{M}\ a)) \vee (\ (\mathcal{M}\ b) \wedge bi(\neg(\mathcal{M}\ a))))) =$
$((\ (\mathcal{M}\ a) \wedge bi\ (\neg(\mathcal{M}\ b))) \vee ((\mathcal{M}\ b) \wedge bi\ (\neg(\mathcal{M}\ a))) \vee ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b)))$
by *auto*
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*: $\vdash \mathcal{M}(a\ THRU\ b) = \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))$
    by (*simp*)
  have *2*: $\vdash \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) = ((\rhd(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (\rhd(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)))$
    using *FstDiAndDiEqv* **by** *auto*
  have *3*: $\vdash ((\rhd(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (\rhd(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a))) =$
      $(((\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee ((\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)))$
    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*: $\vdash \mathcal{M}(FIRST\ LIFT(fin\ (init\ \ w))) = \rhd\ (fin\ (init\ \ w))$ **by** (*simp*)
  have *3*: $\vdash \rhd\ (fin\ (init\ \ w)) = halt(init\ w)$ **using** *HaltStateEqvFstFinState* **by** *fastforce*
  from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailUpto*:
$(FAIL\ UPTO\ a) \simeq (\ a)$
proof −
  have *1*: $\vdash\ \ \mathcal{M}(FAIL\ UPTO\ a) = \rhd(\ (\mathcal{M}\ FAIL) \vee (\mathcal{M}\ a))$ **by** (*simp*)
  have *2*: $\vdash\ \ (\mathcal{M}\ FAIL \vee \mathcal{M}\ a) = (\#False \vee \mathcal{M}\ a)$ **using** *MFailAlt* **by** *auto*
  have *3*: $\vdash \rhd(\mathcal{M}\ FAIL \vee (\mathcal{M}\ a)) = \rhd(\#False \vee (\mathcal{M}\ a))$ **using** *2 FstEqvRule* **by** *blast*
  have *4*: $\vdash (\#False \vee (\mathcal{M}\ a)) = \mathcal{M}\ a$ **by** *simp*
  have *5*: $\vdash \rhd(\#False \vee (\mathcal{M}\ a)) =\ \ \rhd(\mathcal{M}\ a)$ **using** *4 FstEqvRule* **by** *blast*
  have *6*: $\vdash \rhd(\mathcal{M}\ a) = \mathcal{M}\ a$ **using** *MFixFst* **by** *fastforce*
  from *1 2 3 4 5 6* **show** *?thesis* **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*: $\vdash$ $\mathcal{M}$(*FAIL THEN a*) = ($\mathcal{M}$ *FAIL*);($\mathcal{M}$ *a*) **by** (*simp*)
  have *2*: $\vdash$ ($\mathcal{M}$ *FAIL*);($\mathcal{M}$ *a*) = *#False*;($\mathcal{M}$ *a*) **using** *MFailAlt* **using** *LeftChopEqvChop* **by** *blast*
  have *3*: $\vdash$ *#False*;($\mathcal{M}$ *a*) = *#False* **by** (*simp add*: *chop-d-def Valid-def*)
  have *4*: $\vdash$ *#False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
 from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

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

lemma *MWithFalse*:
  (*a WITH* (*LIFT*(*#False*))) $\simeq$ *FAIL*
proof –

have *1*: $\vdash$ $\mathcal{M}$ (*a WITH LIFT*(#*False*)) = (($\mathcal{M}$ *a*) $\wedge$ #*False*) **by** (*simp*)
have *2*: $\vdash$ (($\mathcal{M}$ *a*) $\wedge$ #*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*))) $\simeq$ *a*
proof −
have *1*: $\vdash$ $\mathcal{M}$ (*a WITH LIFT*(#*True*)) = (($\mathcal{M}$ *a*) $\wedge$ #*True*) **by** (*simp*)
have *2*: $\vdash$ (($\mathcal{M}$ *a*) $\wedge$ #*True*) = $\mathcal{M}$ *a* **by** *simp*
from *1 2* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

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

lemma *MEmptyThru*:
(*EMPTY THRU a*) $\simeq$ (*a*)
proof −
have *1*: $\vdash$ $\mathcal{M}$(*EMPTY THRU a*) = $\rhd$(*di*($\mathcal{M}$ *EMPTY*) $\wedge$ *di*($\mathcal{M}$ *a*)) **by** (*simp*)
have *2*: $\vdash$ *di*($\mathcal{M}$ *EMPTY*) = *di empty* **using** *MEmptyAlt DiEqvDi* **by** *blast*
hence *3*: $\vdash$ (*di*($\mathcal{M}$ *EMPTY*) $\wedge$ *di*($\mathcal{M}$ *a*)) = (*di empty* $\wedge$ *di*($\mathcal{M}$ *a*)) **by** *auto*
hence *4*: $\vdash$ (*di empty* $\wedge$ *di*($\mathcal{M}$ *a*)) = *di*($\mathcal{M}$ *a*) **using** *DiEmpty* **by** *auto*
have *5*: $\vdash$ (*di*($\mathcal{M}$ *EMPTY*) $\wedge$ *di*($\mathcal{M}$ *a*)) = *di*($\mathcal{M}$ *a*) **using** *3 4* **by** *fastforce*
hence *6*: $\vdash$ $\rhd$(*di*($\mathcal{M}$ *EMPTY*) $\wedge$ *di*($\mathcal{M}$ *a*)) = $\rhd$(*di*($\mathcal{M}$ *a*)) **using** *FstEqvRule* **by** *blast*
have *7*: $\vdash$ $\rhd$(*di*($\mathcal{M}$ *a*)) = $\rhd$($\mathcal{M}$ *a*) **using** *FstDiEqvFst* **by** *blast*
have *8*: $\vdash$ $\rhd$($\mathcal{M}$ *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*)*)
    by (*simp add*: *ITERATE-d-def*)
  have *2*: $\vdash$ $\mathcal{M}$ (*EMPTY WITH LIFT*($\mathcal{M}$ *b*)*) = ($\mathcal{M}$ *EMPTY* $\wedge$ ($\mathcal{M}$ *b*)*)
    by (*simp*)
  have *3*: $\vdash$ ($\mathcal{M}$ *EMPTY* $\wedge$ ($\mathcal{M}$ *b*)*) = (*empty* $\wedge$ ($\mathcal{M}$ *b*)*)
    using *MEmptyAlt* **by** *auto*
  have *4*: $\vdash$ (*empty* $\wedge$ ($\mathcal{M}$ *b*)*) = (*empty* $\wedge$ (*empty* $\vee$ ((($\mathcal{M}$ *b*) $\wedge$ *more*);($\mathcal{M}$ *b*)*)))
    using *ChopstarEqv* **by** *fastforce*
  have *5*: $\vdash$ (*empty* $\wedge$ (*empty* $\vee$ ((($\mathcal{M}$ *b*) $\wedge$ *more*);($\mathcal{M}$ *b*)*))) = *empty*
    by *auto*
  have *6*: $\vdash$ $\mathcal{M}$(*EMPTY ITERATE b*) = $\mathcal{M}$ *EMPTY*
    using *1 2 3 4 5 MEmptyAlt* **by** *fastforce*
 from *6* **show** *?thesis* **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*)*) **by** (*simp add*: *ITERATE-d-def*)
  have *2*: $\vdash$ $\mathcal{M}$(*a WITH LIFT*($\mathcal{M}$ *a*)*) = (($\mathcal{M}$ *a*) $\wedge$ ($\mathcal{M}$ *a*)*) **by** (*simp*)
  have *3*: $\vdash$ (($\mathcal{M}$ *a*) $\wedge$ ($\mathcal{M}$ *a*)*) = ($\triangleright$($\mathcal{M}$ *a*) $\wedge$ ($\triangleright$($\mathcal{M}$ *a*))*) **using** *MFixFst*
  by (*metis ImpCS inteq-reflection Prop10*)
  have *4*: $\vdash$ ($\triangleright$($\mathcal{M}$ *a*) $\wedge$ ($\triangleright$($\mathcal{M}$ *a*))*) = $\triangleright$($\mathcal{M}$ *a*) **using** *FstAndFstStarEqvFst* **by** *fastforce*
  have *5*: $\vdash$ $\triangleright$($\mathcal{M}$ *a*) = $\mathcal{M}$ *a* **using** *MFixFst* **by** *fastforce*
  from *1 2 3 4 5* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

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

lemma *MThruIdemp*:
 (*a THRU a*) $\simeq$ (*a*)
proof −
  have *1*: $\vdash$ $\mathcal{M}$(*a THRU a*) = $\triangleright$(*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *a*)) **by** *auto*

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

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

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

lemma *MUptoCommut*:
(a *UPTO* b) $\simeq$ (b *UPTO* a)
proof −
have *1*: $\vdash \mathcal{M}(a\ UPTO\ b) = \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))$ **by** (*simp*)
have *2*: $\vdash ((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = ((\mathcal{M}\ b) \vee (\mathcal{M}\ a))$ **by** *auto*
hence *3*: $\vdash \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *blast*
have *4*: $\vdash \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ a)) = \mathcal{M}(b\ UPTO\ a)$ **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*: $\vdash \mathcal{M}(a\ THRU\ b) = \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))$ **by** (*simp*)
have *2*: $\vdash (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) = (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a))$ **by** *auto*
hence *3*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) = \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *blast*
have *4*: $\vdash \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)) = \mathcal{M}(b\ THRU\ a)$ **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*: $\vdash \mathcal{M}((a\ UPTO\ b)\ UPTO\ c) = \triangleright(\mathcal{M}(a\ UPTO\ b) \vee (\mathcal{M}\ c))$
    by (*simp*)
  have *2*: $\vdash \triangleright(\mathcal{M}(a\ UPTO\ b) \vee (\mathcal{M}\ c)) = \triangleright(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c))$
    by *auto*
  have *3*: $\vdash \triangleright(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = \triangleright(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c))$
    using *FstFstOrEqvFstOrL* **by** *blast*
  have *4*: $\vdash (((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = ((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
    by *auto*
  hence *5*: $\vdash \triangleright(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = \triangleright((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
    using *FstEqvRule* **by** *blast*
  have *6*: $\vdash \triangleright((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \triangleright((\mathcal{M}\ a) \vee \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
    using *FstFstOrEqvFstOrR* **by** *fastforce*
  have *7*: $\vdash \triangleright((\mathcal{M}\ a) \vee \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \triangleright((\mathcal{M}\ a) \vee \mathcal{M}(b\ UPTO\ c))$
    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*)) = $\triangleright$(($\mathcal{M}$ *a*) $\vee$ $\triangleright$(*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*)))
    by *simp*
have *2*: $\vdash$ $\triangleright$(($\mathcal{M}$ *a*) $\vee$ $\triangleright$(*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*))) =
        $\triangleright$(($\mathcal{M}$ *a*) $\vee$ (*di*($\mathcal{M}$ *a*) $\wedge$ *di*($\mathcal{M}$ *b*)))
    using *FstFstOrEqvFstOrR* **by** *auto*

have *3*: $\vdash$ $((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$(((\mathcal{M}\ a) \vee di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
by *auto*
have *4*: $\vdash$ $(((\mathcal{M}\ a) \vee di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
using *OrDiEqvDi* **by** *fastforce*
have *5*: $\vdash$ $((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
using *3 4* **by** *auto*
hence *6*: $\vdash \rhd ((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$\rhd ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
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 (*auto simp add*: *first-d-def*)
have *8*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) =$
$((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
by *auto*
hence *9*: $\vdash (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$(\neg((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *fastforce*
have *10*: $\vdash (\neg((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(\neg(((\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *AndDiEqv* **using** *5* **by** *auto*
have *11*: $\vdash (\neg(((\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
by *auto*
have *12*: $\vdash (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$(\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
using *9 10 11* **by** *auto*
hence *13*: $\vdash bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$bs\ (\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
using *BsEqvRule* **by** *blast*
have *14*: $\vdash bs\ ((\neg(\mathcal{M}\ a)) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) =$
$(bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *BsAndEqv* **by** *fastforce*
have *141*: $\vdash bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$(bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *13 14* **by** *fastforce*
hence *15*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))))) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *auto*
have *16*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$((bs\ ((\neg(\mathcal{M}\ a))) \wedge di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *auto*

have *17*: $\vdash ((bs\ ((\neg(\mathcal{M}\ a))) \wedge di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$((\rhd(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *FstEqvBsNotAndDi* **by** *fastforce*
have *18*: $\vdash ((\rhd(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(((\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
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*: $\vdash ((\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$(\rhd(\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))))$
using *MFixFst* **by** *fastforce*
have *28*: $\vdash (\rhd(\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))))$
by (*auto simp add*: *first-d-def*)
have *29*: $\vdash ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)))$
by *auto*
have *30*: $\vdash ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) = \rhd(\mathcal{M}\ a)$
by (*simp add*: *first-d-def*)
have *31*: $\vdash \rhd(\mathcal{M}\ a) = (\mathcal{M}\ a)$
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*: $\vdash \mathcal{M}((a\ UPTO\ b)\ THRU\ (a\ UPTO\ c)) =$
$\triangleright(\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ )$
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*: $\vdash \rhd(di(\mathcal{M}\ a)) = \rhd((\mathcal{M}\ a))$
using *FstDiEqvFst* **by** *blast*
have *9*: $\vdash \rhd((\mathcal{M}\ a)) = (\mathcal{M}\ a)$
using *MFixFst* **by** *fastforce*
have *10*: $\vdash \rhd(di(\mathcal{M}\ a)) = (\mathcal{M}\ a)$
using *8 9* **by** *fastforce*
hence *11*: $\vdash (\rhd(di(\mathcal{M}\ a)) \vee \rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) =$
$((\mathcal{M}\ a) \vee \rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
by *auto*
hence *12*: $\vdash \rhd(\rhd(di(\mathcal{M}\ a)) \vee \rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) =$
$\rhd((\mathcal{M}\ a) \vee \rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
using *FstEqvRule* **by** *blast*
have *13*: $\vdash\rhd((\mathcal{M}\ a) \vee \rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) = \mathcal{M}(a\ UPTO\ (b\ THRU\ c))$
by *simp*
from *1 6 7 12 13* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruUptoDistrib*:
$(a\ THRU\ (b\ UPTO\ c)) \simeq ((a\ THRU\ b)\ UPTO\ (a\ THRU\ c))$
proof –
have *1*: $\vdash \mathcal{M}((a\ THRU\ b)\ UPTO\ (a\ THRU\ c)) =$
$\rhd(\rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c)))$
by *simp*
have *2*: $\vdash \rhd(\rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c))) =$
$\rhd((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c)))$
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*: $\vdash (di(\mathcal{M}\ a) \wedge di(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c))) =$
$(di(\mathcal{M}\ a) \wedge di(\rhd(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$ **using** *DiEqvDiFst* **by** *fastforce*
have *6*: $\vdash ((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c))) =$
$(di(\mathcal{M}\ a) \wedge di(\rhd(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$ **using** *3 4 5* **by** *fastforce*
hence *7*: $\vdash \rhd((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c))) =$
$\rhd(di(\mathcal{M}\ a) \wedge di(\rhd(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$ **using** *FstEqvRule* **by** *blast*
have *8*: $\vdash \rhd(di(\mathcal{M}\ a) \wedge di(\rhd(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c)))) =$
$\mathcal{M}(a\ THRU\ (b\ UPTO\ c))$ **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*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ UPTO\ ((HALT\ w)\ WITH\ g)) =$
$\rhd(\mathcal{M}((HALT\ w)\ WITH\ f) \vee \mathcal{M}((HALT\ w)\ WITH\ g))$
by (*simp*)
have *2*: $\vdash \rhd(\mathcal{M}((HALT\ w)\ WITH\ f) \vee \mathcal{M}((HALT\ w)\ WITH\ g)) =$
$\rhd((\mathcal{M}(HALT\ w) \wedge f) \vee (\mathcal{M}(HALT\ w) \wedge g))$
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*: $\vdash \rhd((\mathcal{M}(HALT\ w) \wedge f) \vee (\mathcal{M}(HALT\ w) \wedge g)) = \rhd(\mathcal{M}(HALT\ w) \wedge (f \vee g))$
using *3 FstEqvRule* **by** *fastforce*
have *5*: $\vdash \rhd(\mathcal{M}(HALT\ w) \wedge (f \vee g)) = \rhd(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \vee g)))$
by *simp*
have *6*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ LIFT(f \vee g))) = \rhd(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \vee g)))$
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*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g)) =$
$\rhd(\ di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)\ )$
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*: $\vdash$ $(\mathcal{M}(a))\ ;\ (\mathcal{M}(b) \wedge \mathcal{M}(c)) = \ \triangleright\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b) \wedge \mathcal{M}(c))$
using *MFixFst LeftChopEqvChop* **by** *blast*
have *4*: $\vdash$ $\triangleright\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b) \wedge \mathcal{M}(c)) = \ ((\triangleright\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b))) \wedge (\triangleright\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(c))))$
using *LFstAndDistrC* **by** *fastforce*
have *5*: $\vdash((\triangleright\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b))) \wedge (\triangleright\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(c)))) =$
$((\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b))) \wedge (\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(c)))\ )$ **using** *MFixFst*
by (*metis 4 inteq-reflection*)
have *6*: $\vdash$ $((\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b))) \wedge (\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(c)))\ ) =$
$(\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c))$
by *simp*
have *7*: $\vdash$ $(\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c)) = \mathcal{M}((a\ THEN\ b)\ AND\ (a\ THEN\ c))$
by (*simp add*: *AND-d-def*)
from *1 2 3 4 5 6 7* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenUptoDistrib*:
$(a\ THEN\ (b\ UPTO\ c)) \simeq ((a\ THEN\ b)\ UPTO\ (a\ THEN\ c))$
proof $-$
have *1*: $\vdash$ $(\mathcal{M}\ (a\ THEN\ (b\ UPTO\ c))) = ((\mathcal{M}\ a);(\triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$

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

end

18 Finite ITL Examples

theory *Example*
imports
FOTheorems TimeReversal
begin

18.1 Example 1

definition *F1* :: *nat statefun* $\Rightarrow$ *temporal*
where *F1 w* $\equiv$ *TEMP* $\square$ ($\#0 \leq \$w$)

definition *Init1* :: *nat statefun* $\Rightarrow$ *temporal*
where *Init1 w* $\equiv$ *TEMP* $\$w = \#0$

lemma *init1*:
$(\langle s0,s1,s2\rangle \models len(2) \wedge Init1\ w) = ((w\ s0) = 0)$
by (*simp add*: *Init1-def current-val-d-def len-defs*)

lemma *exist-test-F1* :
$\vdash \exists\exists\ w.\ F1\ w$
proof −
have *1*: $\bigwedge w. \vdash F1\ w$ **by** (*simp add*: *always-defs current-val-d-def F1-def Valid-def*)
from *1* **show** *?thesis* **by** (*meson EExI MP*)
qed

18.2 Example 2

locale *Test* =
fixes *v* :: *state* $\Rightarrow$ *nat*
fixes *v1* :: *state* $\Rightarrow$ *nat*
fixes *y* :: *state* $\Rightarrow$ *bool*
fixes *z* :: *state* $\Rightarrow$ *int*

fixes *F2* :: *nat statefun* ⇒ *temporal*
fixes *F3* :: *bool statefun* ⇒ *temporal*
fixes *F4* :: *int statefun* ⇒ *temporal*
fixes *F5* :: *nat statefun* ⇒ *temporal*
fixes *Init2* :: *nat statefun* ⇒ *temporal*
fixes *Init3* :: *bool statefun* ⇒ *temporal*
defines $F2 \equiv (\lambda\ v.\ TEMP\ \Box\ (\ \#0 \le \$v\ ))$
defines $F3 \equiv (\lambda\ p.\ TEMP\ \Box\ (\ \$p \vee \neg\ \$p))$
defines $F4 \equiv (\lambda\ z.\ TEMP\ \Box\ (\ \#0 \le \$z \vee \$z < \#0))$
defines $F5 \equiv (\lambda\ v.\ TEMP\ \$v{=}\#0 \wedge v\ gets\ \$v{+}\#1)$
defines $Init2 \equiv (\lambda\ v.\ TEMP\ \$v = \#0)$
defines $Init3 \equiv (\lambda\ p.\ TEMP\ \$p)$

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

lemma (**in** *Test*) *nextempty-test* :
$(\langle s0\rangle \models v\$) =\ (\epsilon\ x.\ x{=}x)$
by (*simp add*: *next-val-d-def*)

lemma (**in** *Test*) *nextempty-test-1* :
$(\langle s0\rangle \models v\$ = v\$)$
by *simp*

lemma (**in** *Test*) *nextempty-test-2* :
$(\langle s0\rangle \models v\$ = v1\$)$
by (*simp add*: *Test.nextempty-test*)

lemma (**in** *Test*) *nextcurrent-test*:
$(\langle s0,s1\rangle \models skip \wedge (\$v{=}\#0) \wedge (v\${=}\$v{+}\#1)) = (((v\ s0) = 0) \wedge ((v\ s1) = 1\ \ ))$
unfolding *current-val-d-def next-val-d-def skip-defs* **by** *auto*

lemma (**in** *Test*) *nextcurrentfinpenult-test*:
$(\langle s0,s1,s2,s3\rangle \models len(3) \wedge v =: !v{-}\#1 \wedge v \leftarrow \#3 \wedge \$v{=}\#0 \wedge v := \$v{+}\#1\ ) =$
$(((v\ s0) = 0) \wedge ((v\ s1) = 1 \wedge (v\ (s2)) = 2 \wedge ((v\ s3) = 3\ \ \ )))$
unfolding *current-val-d-def next-val-d-def fin-val-d-def penult-val-d-def next-assign-d-def prev-assign-d-def temporal-assign-d-def len-defs* **by** *auto*

lemma (**in** *Test*) *stable-test*:
$(\langle s0,s1,s2,s3\rangle \models len(3) \wedge stable\ v \wedge \$v{=}\#0) =$
$((v\ s0) = 0 \wedge (v\ s1) = 0 \wedge (v\ s2) = 0 \wedge (v\ s3) = 0)$
by (*auto simp*: *stable-defs len-defs current-val-d-def next-val-d-def Nitpick.case-nat-unfold*)

lemma (**in** *Test*) *revnextcurrentfinpenult-test*:
$(\langle s0,s1,s2,s3\rangle \models (len\ 3 \wedge v! = !v{-}\#1 \wedge !v = \#3 \wedge \$v{=}\#0 \wedge v\${=}\$v{+}\#1)^r) =$
$(((v\ s3) = 0) \wedge ((v\ s2) = 1 \wedge (v\ (s1)) = 2 \wedge ((v\ s0) = 3\ \ \ )))$

unfolding *reverse-d-def len-defs current-val-d-def next-val-d-def*
penult-val-d-def fin-val-d-def **by** *auto*

lemma (**in** *Test*) *exist-test-F2* :
$\vdash \exists\exists\ v.\ F2\ v$
proof −
have *1*: $\vdash$ *F2 v* **by** (*simp add*: *always-defs current-val-d-def F2-def Valid-def*)
from *1* **show** *?thesis* **by** (*meson EExI MP*)
qed

lemma (**in** *Test*) *exist-test-F3* :
$\vdash \exists\exists\ y.\ F3\ y$
proof −
have *1*: $\vdash$ *F3 y* **by** (*simp add*: *always-defs current-val-d-def F3-def Valid-def*)
from *1* **show** *?thesis* **by** (*meson EExI MP*)
qed

18.3 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*:
(
$x\ (Interval.nth\ w\ (0::nat)) = (0::nat) \wedge$
$(\forall\, i<intlen\ w.\ x\ (Interval.nth\ w\ (Suc\ i)) = Suc\ (x\ (Interval.nth\ w\ i))) \wedge$
$x\ (Interval.nth\ w\ (intlen\ w)) = n) \longrightarrow$
(
$x\ (Interval.nth\ w\ (0::nat)) = (0::nat) \wedge$
$(\forall\, i\le intlen\ w.\ x\ (Interval.nth\ w\ (i)) = i) \wedge$
$x\ (Interval.nth\ w\ (intlen\ w)) = n)$
proof *auto*
show $\bigwedge i.\ x\ (Interval.nth\ w\ 0) = 0 \Longrightarrow$
$\forall\, i<intlen\ w.\ x\ (Interval.nth\ w\ (Suc\ i)) = Suc\ (x\ (Interval.nth\ w\ i)) \Longrightarrow$
$n = x\ (Interval.nth\ w\ (intlen\ w)) \Longrightarrow i \le intlen\ w \Longrightarrow x\ (Interval.nth\ w\ i) = i$
proof −
fix *i*
assume *0*: $x\ (Interval.nth\ w\ 0) = 0$
assume *1*: $\forall\, i<intlen\ w.\ x\ (Interval.nth\ w\ (Suc\ i)) = Suc\ (x\ (Interval.nth\ w\ i))$
assume *2*: $n = x\ (Interval.nth\ w\ (intlen\ w))$
assume *3*: $i \le intlen\ w$
show $x\ (Interval.nth\ w\ i) = i$
using *0 1 2 3*
proof (*induct i*)
case *0*
then show *?case* **by** *simp*
next

case (*Suc i*)
then show *?case* **by** *simp*
qed
qed
qed

lemma (**in** *Test1*) *test-E-F5-2*:
(
x (*Interval.nth w* (*0::nat*)) = (*0::nat*) $\wedge$
($\forall$ *i*$\le$*intlen w*. *x* (*Interval.nth w* (*i*)) = *i*) $\wedge$
x (*Interval.nth w* (*intlen w*)) = *n*) $\longrightarrow$ (
x (*Interval.nth w* (*0::nat*)) = (*0::nat*) $\wedge$
($\forall$ *i*<*intlen w*. *x* (*Interval.nth w* (*Suc i*)) = *Suc* (*x* (*Interval.nth w i*))) $\wedge$
x (*Interval.nth w* (*intlen w*)) = *n*)

by *simp*

lemma (**in** *Test1*) *test-E-F5-3*:
(
x (*Interval.nth w* (*0::nat*)) = (*0::nat*) $\wedge$
($\forall$ *i*<*intlen w*. *x* (*Interval.nth w* (*Suc i*)) = *Suc* (*x* (*Interval.nth w i*))) $\wedge$
x (*Interval.nth w* (*intlen w*)) = *n*) =
(
x (*Interval.nth w* (*0::nat*)) = (*0::nat*) $\wedge$
($\forall$ *i*$\le$*intlen w*. *x* (*Interval.nth w* (*i*)) = *i*) $\wedge$
x (*Interval.nth w* (*intlen w*)) = *n*)
using *test-E-F5-1 test-E-F5-2* **by** *auto*

lemma (**in** *Test1*) *test-E-F5-4*:
($\exists$ *x::state* $\Rightarrow$ *nat*.
x (*Interval.nth w* (*0::nat*)) = (*0::nat*) $\wedge$
($\forall$ *i*<*intlen w*. *x* (*Interval.nth w* (*Suc i*)) = *Suc* (*x* (*Interval.nth w i*))) $\wedge$
x (*Interval.nth w* (*intlen w*)) = *n*) =
($\exists$ *x::state* $\Rightarrow$ *nat*.
x (*Interval.nth w* (*0::nat*)) = (*0::nat*) $\wedge$
($\forall$ *i*$\le$*intlen w*. *x* (*Interval.nth w* (*i*)) = *i*) $\wedge$
x (*Interval.nth w* (*intlen w*)) = *n*)
by (*simp add*: *Test1.test-E-F5-3*)

lemma (**in** *Test1*) *test-E-F5*:
$\vdash$ ($\exists\exists$ *v*. (*F5 v n*)) $\longrightarrow$ (*len n*)
by (*auto simp add*: *Valid-def F5-def exist-state-d-def gets-defs current-val-d-def*
fin-defs sub-def len-defs)
(*metis Test1.test-E-F5-1 nat-le-linear*)

18.4 Example 4

locale *Testrev* =

fixes x :: $state \Rightarrow nat$
fixes $F1$ :: $nat\ statefun \Rightarrow temporal$
defines $F1 \equiv (\lambda\ v.\ TEMP\ \$v{=}\#0 \wedge skip \wedge v{:=}\ \$v{+}\#1\ )$

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 ... =
$(\sigma \models ((!x{=}\#0) \wedge skip \wedge (x! = !x{+}\#1)))$
using *RRAnd*
by (*simp add*: *all-rev-eq*(1) *all-rev-eq*(12) *all-rev-eq*(3) *all-rev-eq*(8) *all-rev-eq*(9) *next-assign-d-def*)
also have ... =
$(\sigma \models ((x\${=}\#0) \wedge skip \wedge (\$x = x\${+}\#1)))$
by (*simp add*: *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

18.5 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

19 Monitor Example

theory *MonitorExample*
imports

FOTheorems Monitor
begin

locale *Test* =
fixes *v* :: *state* ⇒ *nat*
fixes *y* :: *state* ⇒ *bool*
fixes *z* :: *state* ⇒ *nat*
fixes *F2* :: *nat statefun* ⇒ *temporal*
fixes *F3* :: *bool statefun* ⇒ *temporal*
fixes *F4* :: *nat statefun* ⇒ *temporal*
fixes *F5* :: *nat statefun* ⇒ *temporal*
fixes *Init2* :: *nat statefun* ⇒ *temporal*
fixes *Init3* :: *bool statefun* ⇒ *temporal*
fixes *Mon1* :: *state monitor*
fixes *Mon2* :: *state monitor*
fixes *Mon3* :: *state monitor*
fixes *Mon4* :: *state monitor*
fixes *Mon5* :: *state monitor*
fixes *Mon6* :: *state monitor*
defines *F2* ≡ (λ *v*. *TEMP* □ (#*0* ≤ $*v*))
defines *F3* ≡ (λ *p*. *TEMP* □ ($*p* ∨ ¬ $*p*))
defines *F4* ≡ (λ *z*. *TEMP* $*z*=#*0* ∧ *z gets* $*z*+#*1*)
defines *F5* ≡ (λ *z*. *TEMP fin*($*z*=#*4*))
defines *Init2* ≡ (λ *v*. *TEMP* $*v* = #*0*)
defines *Init3* ≡ (λ *p*. *TEMP* $*p*)
defines *Mon1* ≡ *FIRST*(*F2 v*)
defines *Mon2* ≡ *EMPTY UPTO Mon1*
defines *Mon3* ≡ *Mon1 WITH* (*F2 v*)
defines *Mon4* ≡ *Mon2 THEN Mon1*
defines *Mon5* ≡ *Mon3 THRU Mon4*
defines *Mon6* ≡ (*FIRST F4 z*) *WITH* (*F5 z*)

lemma (**in** *Test*) *test*:
⊢ 𝓜(*Mon1*) = *empty*
proof −
have *1*: ⊢ 𝓜(*Mon1*) = ▷(□ (#*0* ≤ $*v*))
using *F2-def Mon1-def* **by** *fastforce*
have *2*: ⊢ □ (#*0* ≤ $*v*)
by (*simp add*: *Valid-def always-defs current-val-d-def*)
have *3*: ⊢ ▷(□ (#*0* ≤ $*v*)) = *empty*
using *2* **by** (*metis FstTrue int-eq int-eq-true*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test1*:
⊢ 𝓜(*Mon2*) = *empty*
proof −

have *1*: $\vdash \mathcal{M}(Mon2) = \mathcal{M}(EMPTY\ UPTO\ Mon1)$
using *Mon2-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(EMPTY\ UPTO\ Mon1) = \rhd(\mathcal{M}(EMPTY) \vee \mathcal{M}(Mon1))$
by *fastforce*
have *3*: $\vdash \rhd(\mathcal{M}(EMPTY) \vee \mathcal{M}(Mon1)) = \rhd(empty \vee empty)$
using *test* **by** (*metis 2 MEmptyAlt int-eq*)
have *4*: $\vdash \rhd(empty \vee empty) = empty$
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*: $\vdash \mathcal{M}(Mon3\ THRU\ Mon4) = \rhd(di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4)))$
by *fastforce*
have *3*: $\vdash (di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4))) = (di(empty) \wedge di(empty))$
using *test3 test2* **by** (*metis inteq-reflection lift-and-com*)
hence *4*: $\vdash \rhd(di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4))) = \rhd(di(empty) \wedge di(empty))$
by (*simp add*: *FstEqvRule*)
have *5*: $\vdash \rhd(di(empty) \wedge di(empty)) = \rhd(di(empty))$
by *simp*
have *6*: $\vdash \rhd(di(empty)) = empty$

    using *FstDiEqvFst FstEmpty* **by** *fastforce*
 from *6 5 4 2 1* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test5*:
$\vdash \mathcal{M}\ (Mon6) = (\triangleright(\$z{=}\#0\ \wedge\ z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)\ )$
proof −
 have *1*: $\vdash \mathcal{M}(Mon6) = (\mathcal{M}(FIRST\ F4\ z) \wedge (F5\ z))$
    using *Mon6-def* **by** *fastforce*
 have *2*: $\vdash (\mathcal{M}(FIRST\ F4\ z) \wedge (F5\ z)) = (\triangleright(F4\ z) \wedge fin(\$z{=}\#4))$
    using *F5-def* **by** *fastforce*
 have *3*: $\vdash (\triangleright(F4\ z) \wedge fin(\$z{=}\#4)) = (\triangleright(\$z{=}\#0 \wedge\ z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)\ )$
    using *F4-def* **by** *fastforce*
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test5-1*:
$\vdash \triangleright(\$z{=}\#0 \wedge\ z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)\ \longrightarrow$
  $\triangleright((\$z{=}\#0 \wedge z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4))$

using *FstWithAndImp* **by** *blast*

lemma (**in** *Test*) *test5-2*:
 $(s \models (\$z{=}\#0 \wedge z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)) =$
  $(z\ (nth\ s\ 0) = 0\ \wedge (\forall\ i{<}\ intlen\ s.\ z\ (nth\ s\ (Suc\ i)) = Suc(z\ (nth\ s\ i))) \wedge$
   $z\ (nth\ s\ (intlen\ s)) = 4)$
by (*simp add*: *gets-defs fin-defs current-val-d-def sub-def*)

lemma (**in** *Test*) *test5-3*:
  $(z\ (nth\ s\ 0) = 0\ \wedge (\forall\ i{<}\ intlen\ s.\ z\ (nth\ s\ (Suc\ i)) = Suc(z\ (nth\ s\ i))) \wedge$
   $z\ (nth\ s\ (intlen\ s)) = 4)$
  $\Longrightarrow$
 $(z\ (nth\ s\ 0) = 0\ \wedge (\forall\ i{\le}\ intlen\ s.\ z\ (nth\ s\ i) = i)$
  $\wedge\ z\ (nth\ s\ (intlen\ s)) = 4)$

proof −
  assume *0*: $(z\ (nth\ s\ 0) = 0\ \wedge (\forall\ i{<}\ intlen\ s.\ z\ (nth\ s\ (Suc\ i)) = Suc(z\ (nth\ s\ i))) \wedge$
   $z\ (nth\ s\ (intlen\ s)) = 4)$
  show $(z\ (nth\ s\ 0) = 0\ \wedge (\forall\ i{\le}\ intlen\ s.\ z\ (nth\ s\ i) = i)$
  $\wedge\ z\ (nth\ s\ (intlen\ s)) = 4)$
  proof −
   have *1*: $z\ (nth\ s\ 0) = 0$ **using** *0* **by** *auto*
   have *2*: $z\ (nth\ s\ (intlen\ s)) = 4$ **using** *0* **by** *auto*
   have *3*: $(\forall\ i{\le}\ intlen\ s.\ z\ (nth\ s\ i) = i)$
   proof
    fix *i*
    show $i \le intlen\ s \longrightarrow z\ (Interval.nth\ s\ i) = i$
    proof
     (*induct i*)
     case *0*

```
      then show ?case by (simp add: 1)
      next
      case (Suc i)
      then show ?case by (simp add: 0)
      qed
   qed
   from 1 2 3 show ?thesis by auto
  qed
qed

lemma (in Test) test5-4:
   (z (nth s 0) =0  ∧ (∀ 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* :
$(s \models \triangleright((\$z{=}\#0 \wedge z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4))) =$
(
($(s \models (\$z{=}\#0 \wedge z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)) \wedge intlen\ s = 0) \vee$
($0 < intlen\ s \wedge (s \models \$z{=}\#0 \wedge z\ gets\ \$z{+}\#1 \wedge fin(\$z{=}\#4)) \wedge$
$(\forall\ ia < intlen\ s.\ (prefix\ ia\ s \models \neg((\$z{=}\#0 \wedge z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)))))$
)

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

20 Filter on Intervals

theory *IntervalFilter*
imports
Interval
begin

The filter operator on intervals is defined. The definition of filter is slightly more complicated than the one for lists as an interval has at least one state and one needs to ensure that the filter operator always returns an interval. The lemmas involving the filter on intervals are similar to those for the filter operator on lists only a bit more complicated.

20.1 Definitions

definition *opfx* :: *'a interval* ⇒ *'a interval* ⇒ *bool*
where *opfx xs ys* = ($\exists$ *zs*. *ys* = *xs* ⊖ *zs* ∨ *ys* = *xs*)

definition *sopfx* :: *'a interval* ⇒ *'a interval* ⇒ *bool*
where *sopfx xs ys* ⟷ *opfx xs ys* ∧ *xs* ≠ *ys*

interpretation *opfx-order*: *order opfx sopfx*
proof *standard*
show ⋀*x y*. *sopfx x y* = (*opfx x y* ∧ ¬ *opfx y x*)
by (*auto simp add*: *opfx-def sopfx-def*)
(*metis One-nat-def add-diff-cancel-left' diff-is-0-eq interval-intlen-intapp le-add1 nat.simps*(*3*))
show ⋀*x*. *opfx x x*
by (*auto simp add*: *opfx-def sopfx-def*)
show ⋀*x y z*. *opfx x y* ⟹ *opfx y z* ⟹ *opfx x z*
by (*auto simp add*: *opfx-def sopfx-def*)
show ⋀*x y*. *opfx x y* ⟹ *opfx y x* ⟹ *x* = *y*
by (*auto simp add*: *opfx-def sopfx-def*)
(*metis One-nat-def add-diff-cancel-left' diff-is-0-eq interval-intlen-intapp le-add1 nat.simps*(*3*))
qed

definition *osfx* :: *'a interval* ⇒ *'a interval* ⇒ *bool*
where *osfx xs ys* = ($\exists$ *zs*. *ys* = *zs* ⊖ *xs* ∨ *ys* =*xs*)

definition *sosfx* :: *'a interval* ⇒ *'a interval* ⇒ *bool*
where *sosfx xs ys* ⟷ *osfx xs ys* ∧ *xs* ≠ *ys*

interpretation *osfx-order*: *order osfx sosfx*
proof *standard*
 show ⋀*x y*. *sosfx x y* = (*osfx x y* ∧ ¬ *osfx y x*)
  by (*auto simp add*: *osfx-def sosfx-def*)
   (*metis add-diff-cancel-right' add-eq-0-iff-both-eq-0 diff-is-0-eq' interval-intapp-assoc interval-intlen-intapp le-add1 not-one-le-zero*)
 show ⋀*x*. *osfx x x*
  by (*auto simp add*: *osfx-def sosfx-def*)
 show ⋀*x y z*. *osfx x y* ⟹ *osfx y z* ⟹ *osfx x z*
  by (*auto simp add*: *osfx-def sosfx-def*)
   (*metis interval-intapp-assoc*)
 show ⋀*x y*. *osfx x y* ⟹ *osfx y x* ⟹ *x* = *y*
  by (*auto simp add*: *osfx-def sosfx-def*)
   (*metis add-diff-cancel-right' add-eq-0-iff-both-eq-0 diff-is-0-eq' interval-intapp-assoc interval-intlen-intapp le-add1 not-one-le-zero*)
qed

primrec *distinct* :: *'a interval* ⇒ *bool*
 where *distinct* ⟨*x*⟩ ⟷ *True*
  | *distinct* (*x*⊙*xs*) ⟷ *x* ∉ *set xs* ∧ *distinct xs*

primrec *remdups* :: *'a interval* ⇒ *'a interval*
 where *remdups* ⟨*x*⟩ = ⟨*x*⟩
  | *remdups* (*x*⊙*xs*) = (*if x* ∈ *set xs then remdups xs else x* ⊙ *remdups xs*)

primrec *filter*:: (*'a* ⇒ *bool*) ⇒ *'a interval* ⇒ *'a interval*
 where *filter P* ⟨*x*⟩ = ⟨*x*⟩
  | *filter P* (*x*⊙*xs*) = (*if* (∃ *y*∈*set xs*. *P y*) *then*
     (*if P x then x* ⊙ *filter P xs else filter P xs*)
    else ⟨*x*⟩)

primrec *nfilter*:: (*'a* ⇒ *bool*) ⇒ *'a interval* ⇒ *nat* ⇒ *nat interval*
 where *nfilter P* ⟨*x*⟩ *n* = ⟨*n*⟩
  | *nfilter P* (*x*⊙*xs*) *n* = (*if* (∃ *y*∈*set xs*. *P y*) *then*
     (*if P x then n* ⊙ (*nfilter P xs* (*Suc n*))
      else nfilter P xs (*Suc n*))
    else ⟨*n*⟩)

primrec *prefixes* :: *'a interval* ⇒ *'a interval interval*
 where
  prefixes ⟨*x*⟩ = ⟨⟨*x*⟩⟩
 | *prefixes* (*x*⊙*xs*) = ⟨*x*⟩ ⊙ (*map* ((⊙) *x*) (*prefixes xs*))

primrec *suffixes* :: *'a interval* ⇒ *'a interval interval*
where

suffixes ⟨*x*⟩ = ⟨⟨*x*⟩⟩
| *suffixes* (*x*⊙*xs*) = (*x*⊙*xs*) ⊙ (*suffixes xs*)

20.2 Lemmas

20.2.1 opfx and sopfx

lemma *opfxI* [*intro?*]:
assumes *ys* = *xs* ⊖ *zs* ∨ *ys* = *xs*
shows *opfx xs ys*
using *assms* **unfolding** *opfx-def* **by** *blast*

lemma *opfxE* [*elim?*]:
assumes *opfx xs ys*
obtains *zs* **where** *ys* = *xs* ⊖ *zs* ∨ *ys* = *xs*
using *assms* **unfolding** *opfx-def* **by** *blast*

lemma *sopfxI′* [*intro?*]:
ys = *xs* ⊖ (*zs*) ⟹ *sopfx xs ys*
unfolding *sopfx-def opfx-def*
by (*metis add-le-same-cancel1 interval-intlen-intapp le-add1 not-one-le-zero*)

lemma *sopfxE′* [*elim?*]:
assumes *sopfx xs ys*
obtains *zs* **where** *ys* = *xs* ⊖ *zs*
using *assms* **unfolding** *sopfx-def opfx-def* **by** *blast*

lemma *opfx-state* [*simp*]:
opfx ⟨*intfirst xs*⟩ *xs*
unfolding *opfx-def intfirst-def*
by (*metis intapp-St interval-hd-tail interval-intfirst-suffix interval-suffix-intlen interval-suffix-zero le0 order.order-iff-strict*)

lemma *opfx-snoc* [*simp*]:
opfx xs (*ys* ⊖ ⟨*y*⟩) ⟷ *xs* = *ys*⊖⟨*y*⟩ ∨ *opfx xs ys*
unfolding *opfx-def*
by (*metis interval-intapp-eq-intapp-conv2 interval-intapp-not-state*)

lemma *cons-pfx-cons* [*simp*]:
opfx (*x*⊙*xs*) (*y*⊙*ys*) = (*x* =*y* ∧ *opfx xs ys*)
by (*auto simp add*: *opfx-def*)

lemma *opfx-code* [*code*]:
opfx ⟨*intfirst xs*⟩ *xs* ⟷ *True*
opfx (*x*⊙*xs*) ⟨*y*⟩ ⟷ *False*
opfx (*x*⊙*xs*) (*y*⊙*ys*) ⟷ (*x*=*y* ∧ *opfx xs ys*)
proof *simp-all*
show *opfx* ⟨*nth xs 0*⟩ *xs*
using *opfx-state* **by** *auto*
show ¬ *opfx* (*x* ⊙ *xs*) ⟨*y*⟩
by (*simp add*: *opfx-def*)

qed

lemma *same-opfx-opfx* [*simp*]:
opfx (*xs* $\ominus$ *ys*) (*xs*$\ominus$*zs*) = *opfx ys zs*
by (*induct xs*) *simp-all*

lemma *same-opfx-state* [*simp*]:
opfx (*xs* $\ominus$ *ys*) (*xs* $\ominus$ $\langle y\rangle$) = (*ys* = $\langle y\rangle$)
by (*meson interval-same-intapp-eq opfx-order.less-le-not-le opfx-snoc sopfxI'*)

lemma *opfx-opfx* [*simp*]:
assumes *opfx xs ys*
shows *opfx xs* (*ys*$\ominus$*zs*)
using *assms* **unfolding** *opfx-def* **by** *fastforce*

lemma *intapp-opfxD*:
assumes *opfx* (*xs* $\ominus$ *ys*) *zs*
shows *opfx xs zs*
using *assms* **by** (*auto simp add*: *opfx-def*)

lemma *opfx-cons*:
opfx xs (*y*$\odot$*ys*) = (*xs* = $\langle y\rangle$ $\vee$ ($\exists$ *zs*. *xs* = *y*$\odot$*zs* $\wedge$ *opfx zs ys*))
by (*case-tac xs*) (*auto simp add*: *opfx-def*)

lemma *opfx-intapp*:
opfx xs (*ys*$\ominus$*zs*) = (*opfx xs ys* $\vee$ ($\exists$ *us*. *xs* = *ys* $\ominus$ *us* $\wedge$ *opfx us zs*))
proof (*induct zs rule*: *interval-rev-induct*)
case (*St y*)
then show *?case* **by** (*meson opfx-snoc same-opfx-opfx*)
next
case (*snoc x xs*)
then show *?case* **by** (*metis interval-intapp-assoc opfx-snoc*)
qed

lemma *intapp-one-opfx*:
assumes *opfx xs ys*
intlen xs < *intlen ys*
shows *opfx* (*xs*$\ominus$ $\langle$*nth ys* ((*intlen xs*)+*1*)$\rangle$) *ys*
using *assms*
proof (*unfold opfx-def*)
assume *1*: $\exists$*zs*. *ys* = *xs* $\ominus$ *zs* $\vee$ *ys* = *xs*
then obtain *sk* **where** *2*: *ys* = *xs* $\ominus$ *sk* $\vee$ *ys* = *xs*
by *fastforce*
assume *3*: *intlen xs* < *intlen ys*
have *4*: *intlen sk* $\geq$ *0*
by *auto*
have *5*: $\exists$ *v*. *ys* = *xs* $\ominus$ ((*intfirst sk*) $\odot$ *v*) $\vee$ *ys* = *xs* $\ominus$ $\langle$*intfirst sk*$\rangle$
by (*metis 2 3 4 interval-hd-tail interval-nth-zero-intfirst interval-suffix-intlen interval-suffix-zero le-eq-less-or-eq nat-neq-iff*)
have *6*: *ys* $\neq$ *xs*

using *3* **by** *blast*
have *7*: (*intfirst sk*) = *nth* (*xs* ⊖ *sk*) ((*intlen xs*) + *1*)
by (*simp add*: *interval-intapp-nth*)
have *8*: *ys* = *xs* ⊖ *sk*
using *2 6* **by** *auto*
have *9*: *nth* (*xs* ⊖ *sk*) ((*intlen xs*) + *1*) = *nth* (*ys*) ((*intlen xs*) + *1*)
using *8* **by** *blast*
thus ∃ *zs*. *ys* = (*xs* ⊖ ⟨*nth ys* ((*intlen xs*) + *1*)⟩) ⊖ *zs* ∨ *ys* = *xs* ⊖ ⟨*nth ys* ((*intlen xs*) + *1*)⟩
using *5 7 9*
by *simp*
qed

lemma *opfx-intlen-le*:
assumes *opfx xs ys*
shows *intlen xs* ≤ *intlen ys*
using *assms* **by** (*auto simp add*: *opfx-def*)

lemma *opfx-same-cases*:
assumes *opfx xs1 ys*
opfx xs2 ys
shows *opfx xs1 xs2* ∨ *opfx xs2 xs1*
using *assms* **unfolding** *opfx-def* **using** *interval-intapp-eq-intapp-conv2*
by *metis*

lemma *opfx-intlen-opfx*:
assumes *opfx ps xs*
opfx qs xs
intlen ps ≤ *intlen qs*
shows *opfx ps qs*
using *assms*
by (*auto simp*: *opfx-def*)
(*metis assms*(*2*) *dual-order.antisym interval-intapp-eq-intapp-conv opfx-intapp opfx-intlen-le*)

lemma *set-mono-opfx*:
assumes *opfx xs ys*
shows *set xs* ⊆ *set ys*
using *assms* **by** (*auto simp*: *opfx-def*)

lemma *prefix-is-opfx*:
opfx (*prefix n xs*) *xs*
by (*auto simp*: *opfx-def*)
(*metis Suc-eq-plus1 add-diff-inverse-nat interval-intapp-prefix-suffix interval-prefix-intlen-gr-1 less-Suc-eq-0-disj less-Suc-eq-le not-less*)

lemma *map-mono-opfx*:
assumes *opfx xs ys*
shows *opfx* (*map f xs*) (*map f ys*)
using *assms* **by** (*auto simp*: *opfx-def*)

lemma *opfx-intlen-less*:

```
 assumes sopfx xs ys
 shows   intlen xs < intlen ys
using assms by (auto simp: sopfx-def opfx-def)

lemma opfx-snocD:
 assumes opfx (xs⊖⟨x⟩) ys
 shows   sopfx xs ys
using assms intapp-opfxD opfx-order.antisym sopfxI' sopfx-def by blast

lemma sopfx-simps [simp, code]:
 sopfx xs ⟨y⟩ ⟷ False
 sopfx ⟨x⟩ (x⊙xs) ⟷ True
 sopfx (x⊙xs) (y⊙ys) ⟷ x=y ∧ sopfx xs ys
proof (simp-all add: sopfx-def)
 show opfx xs ⟨y⟩ ⟶ xs = ⟨y⟩
   by (metis interval-intapp-not-state opfxE)
 show opfx ⟨x⟩ (x ⊙ xs)
   by (simp add: opfx-cons)
 show (x = y ∧ opfx xs ys ∧ (x = y ⟶ xs ≠ ys)) = (x = y ∧ opfx xs ys ∧ xs ≠ ys)
   by auto
qed

lemma prefix-sopfx:
 assumes sopfx xs ys
 shows   sopfx (prefix n xs) ys
using assms
proof (induct n arbitrary: xs ys)
case 0
then show ?case
  proof (cases ys)
  case (St x1)
  then show ?thesis
   using 0.prems by auto
  next
  case (Cons x21 x22)
  then show ?thesis
   using 0.prems opfx-order.le-less-trans prefix-is-opfx by blast
  qed
next
case (Suc n)
then show ?case
 using opfx-order.order.strict-trans1 prefix-is-opfx by blast
qed

lemma osfxI [intro?]:
 assumes ys = zs ⊖ xs ∨ ys =xs
 shows     osfx xs ys
using assms unfolding osfx-def by blast

lemma osfxE [elim?]:
```

assumes *osfx xs ys*
obtains *zs* **where** *ys*= *zs* ⊖ *xs* ∨ *ys* = *xs*
using *assms* **unfolding** *osfx-def* **by** *blast*

lemma *osfx-tl* [*simp*]:
assumes *intlen xs* >*0*
shows *osfx* (*suffix 1 xs*) *xs*
using *assms*
proof (*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case* **by** (*auto simp*: *osfx-def*) (*metis intapp.simps*(*1*))
qed

lemma *osfx-suffix* [*simp*]:
assumes *i*≤ *intlen xs*
shows *osfx* (*suffix i xs*) *xs*
using *assms*
proof (*cases i*)
case *0*
then show *?thesis* **by** *simp*
next
case (*Suc nat*)
then show *?thesis*
proof (*cases xs*)
case (*St x1*)
then show *?thesis* **unfolding** *osfx-def* **by** *simp*
next
case (*Cons x21 x22*)
then show *?thesis* **unfolding** *osfx-def*
by (*metis Suc Suc-eq-plus1 assms interval-intapp-prefix-suffix less-le-trans zero-less-Suc*)
qed
qed

lemma *osfx-prefix* [*simp*]:
assumes *osfx xs ys*
shows *ys* = (*prefix* (*intlen ys* − *intlen xs* −*1*) *ys*) ⊖ *xs* ∨ *ys* =*xs*
using *assms*
by (*auto simp add*: *osfx-def*)
(*metis diff-zero interval-prefix-intapp interval-prefix-intlen*)

lemma *osfx-snoc* [*simp*]:
osfx xs (*ys*⊖ ⟨*y*⟩) ⟷
xs = ⟨*y*⟩ ∨ (∃ *zs*. *xs* = *zs* ⊖ ⟨*y*⟩ ∧ *osfx zs ys*)
proof (*cases xs*)
case (*St x1*)

then show *?thesis* **by** (*metis interval-intlast-intapp interval-intlast-intapp2 osfxI osfx-prefix*)
next
case (*Cons x21 x22*)
then show *?thesis*
 proof *auto*
  show $xs = x21 \odot x22 \Longrightarrow osfx\ (x21 \odot x22)\ (ys \ominus \langle y\rangle) \Longrightarrow \exists zs.\ x21 \odot x22 = zs \ominus \langle y\rangle \wedge osfx\ zs\ ys$
   using *interval-intapp-eq-intapp-conv2 interval-intapp-not-state* **unfolding** *osfx-def*
    by (*metis interval.distinct*(*1*))
  show $\bigwedge zs.\ xs = zs \ominus \langle y\rangle \Longrightarrow x21 \odot x22 = zs \ominus \langle y\rangle \Longrightarrow osfx\ zs\ ys \Longrightarrow osfx\ (zs \ominus \langle y\rangle)\ (ys \ominus \langle y\rangle)$
   unfolding *osfx-def* **by** (*metis interval-intapp-assoc*)
   qed
qed

lemma *snoc-osfx-snoc* [*simp*]:
$osfx\ (xs \ominus \langle x\rangle)\ (ys \ominus \langle y\rangle) = (x = y \wedge osfx\ xs\ ys)$
by (*simp add*: *osfx-def*)
 (*metis interval-intapp-assoc interval-intapp-eq-conv*)

lemma *same-osfx-osfx* [*simp*]:
$osfx\ (ys \ominus xs)\ (zs \ominus xs) = osfx\ ys\ zs$
unfolding *osfx-def*
by (*metis interval-intapp-assoc interval-intapp-same-eq*)

lemma *same-suffix-state* [*simp*]:
$osfx\ (ys \ominus xs)\ (x \odot xs) = (ys = \langle x\rangle)$
unfolding *osfx-def*
by (*metis intapp-St interval-intapp-assoc interval-intapp-not-state interval-intapp-same-eq*)

lemma *osfx-cons*:
 $osfx\ xs\ (y \odot ys) \longleftrightarrow xs = y \odot ys \vee osfx\ xs\ ys$
unfolding *osfx-def*
by (*auto simp*: *interval-cons-eq-intapp-conv*)

lemma *osfx-intapp*:
 $osfx\ xs\ (ys \ominus zs) \longleftrightarrow$
  $osfx\ xs\ zs\ \vee (\exists xs'.\ xs = xs' \ominus zs \wedge osfx\ xs'\ ys)$
by (*auto simp*: *osfx-def interval-intapp-eq-intapp-conv2*)

lemma *osfx-intlen*:
assumes *osfx xs ys*
 shows $intlen\ xs \leq intlen\ ys$
using *assms* **by** (*auto simp add*: *osfx-def*)

lemma *osfx-same-cases*:
assumes $osfx\ xs_1\ ys$
  $osfx\ xs_2\ ys$
shows $osfx\ xs_1\ xs_2 \vee osfx\ xs_2\ xs_1$
using *assms* **unfolding** *osfx-def* **by** (*metis interval-intapp-eq-intapp-conv2*)

lemma *osfx-intlen-osfx*:

assumes *osfx ps xs*
osfx qs xs
intlen ps $\leq$ *intlen qs*
shows *osfx ps qs*
using *assms*
by (*auto simp*: *osfx-def interval-intapp-eq-intapp-conv2*)

lemma *osfx-intlen-less*:
assumes *sosfx xs ys*
shows *intlen xs* $<$ *intlen ys*
using *assms* **by** (*auto simp*: *sosfx-def osfx-def*)

lemma *osfx-ConsD'*:
assumes *osfx* (*x*⊙*xs*) *ys*
shows *sosfx xs ys*
using *assms*
by (*simp add*: *sosfx-def osfx-def*)
(*metis interval-intapp-assoc interval-rev-eq-cons-iff interval-rev-intapp interval-rev-rev-ident opfx-order.dual-order.strict-iff-order sopfxI'*)

lemma *suffix-sosfx*:
assumes *sosfx xs ys*
shows *sosfx* (*suffix n xs*) *ys*
using *assms*
proof (*induct n arbitrary*: *xs ys*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases xs*)
case (*St x1*)
then show *?thesis* **using** *Suc.prems* **by** *auto*
next
case (*Cons x21 x22*)
then show *?thesis*
using *Suc.hyps Suc.prems osfx-ConsD' osfx-order.less-imp-le* **by** *fastforce*
qed
qed

lemma *sosfx-tl* [*simp*]:
assumes *intlen xs* $>$*0*
shows *sosfx* (*suffix 1 xs*) *xs*
using *assms*
proof (*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)

then show *?case* **using** *osfx-ConsD'* **by** *force*
qed

lemma *state-osfx* [*simp*]:
  osfx ⟨*intlast xs*⟩ *xs*
using *osfx-suffix* **by** *fastforce*

lemma *osfx-state* [*simp*]:
  (*osfx xs* ⟨*intlast xs*⟩) = (*xs* = ⟨*intlast xs*⟩)
using *osfx-order.antisym state-osfx* **by** *auto*

lemma *osfx-ConsI*:
assumes *osfx xs ys*
shows *osfx xs* (*x*⊙ *ys*)
using *assms*
by (*metis interval.inject*(*2*) *interval-hd-tail intlen.simps*(*2*) *le-add1*
   osfx-suffix plus-1-eq-Suc osfx-order.order.trans zero-less-Suc)

lemma *osfx-ConsD*:
assumes *osfx* (*x*⊙*xs*) *ys*
shows *osfx xs ys*
using *assms*
by (*meson osfx-ConsD' osfx-order.less-imp-le*)

lemma *osfx-intappI*:
assumes *osfx xs ys*
shows *osfx xs* (*zs* ⊖ *ys*)
using *assms* **by** (*metis interval-intapp-assoc osfx-def*)

lemma *osfx-intappD*:
assumes *osfx* (*zs* ⊖ *xs*) *ys*
shows *osfx xs ys*
using *assms osfxI osfx-order.dual-order.trans* **by** *blast*

lemma *sosfx-set-subset*:
assumes *sosfx xs ys*
shows *set xs* ⊆ *set ys*
using *assms* **by** (*auto simp*: *sosfx-def osfx-def*)

lemma *set-mono-osfx*:
assumes *osfx xs ys*
shows *set xs* ⊆ *set ys*
using *assms* **by** (*auto simp*: *osfx-def*)

lemma *osfx-ConsD2*:
assumes *osfx* (*x*⊙*xs*) (*y*⊙*ys*)
shows *osfx xs ys*
using *assms*
proof −
 assume *osfx* (*x*⊙*xs*) (*y*⊙*ys*)

then obtain *zs* **where** $y \odot ys = zs \ominus (x \odot xs) \lor y \odot ys = x \odot xs$
  using *osfxE* **by** *blast*
then show *?thesis*
by (*metis assms interval.inject*(*2*) *osfx-ConsD osfx-cons*)
qed

lemma *osfx-to-opfx* [*code*]:
osfx xs ys $\longleftrightarrow$ *opfx* (*intrev xs*) (*intrev ys*)
unfolding *opfx-def*
by (*metis interval-rev-intapp interval-rev-rev-ident osfx-def*)

lemma *sosfx-to-sopfx* [*code*]:
sosfx xs ys $\longleftrightarrow$ *sopfx* (*intrev xs*) (*intrev ys*)
by (*auto simp*: *osfx-to-opfx sosfx-def sopfx-def*)

lemma *map-mono-osfx*:
assumes *osfx xs ys*
shows *osfx* (*map f xs*) (*map f ys*)
using *assms* **by** (*auto elim*!: *osfxE intro*: *osfxI*)

lemma *prefix-subset*:
assumes $k \leq$ *intlen xs*
shows *set* (*prefix k xs*) $\leq$ *set xs*
using *assms* **by** (*simp add*: *prefix-is-opfx set-mono-opfx*)

lemma *suffix-subset*:
assumes $k \leq$ *intlen xs*
shows *set* (*suffix k xs*) $\leq$ *set xs*
using *assms* **by** (*simp add*: *set-mono-osfx*)

20.2.2 distinct and remdups

lemma *distinct-intapp* [*simp*]:
distinct ($xs \ominus ys$) = (*distinct xs* $\land$ *distinct ys* $\land$ *set xs* $\cap$ *set ys* = {})
by (*induct xs*) *auto*

lemma *distinct-osfx*:
assumes *distinct ys*
  osfx xs ys
shows *distinct xs*
using *assms*
proof (*clarsimp elim*!: *osfxE*)
show $\bigwedge zs$. *distinct ys* $\Longrightarrow$ $ys = zs \ominus xs \lor ys = xs$ $\Longrightarrow$ *distinct xs*
using *distinct-intapp* **by** *blast*
qed

lemma *distinct-tl*:
assumes *distinct xs*
  intlen xs > *0*

shows *distinct* (*suffix 1 xs*)
using *assms*
by (*cases xs*) *auto*

lemma *distinct-intrev* [*simp*]:
distinct (*intrev xs*) = *distinct xs*
by (*induct xs*) *auto*

lemma *set-remdups* [*simp*]:
set (*remdups xs*) = *set xs*
by (*induct xs*) *auto*

lemma *distinct-remdups* [*iff*]:
distinct (*remdups xs*)
by (*induct xs*) *auto*

lemma *distinct-remdups-id*:
assumes *distinct xs*
shows *remdups xs* = *xs*
using *assms*
by (*induct xs*) *auto*

lemma *remdups-id-iff-distinct* [*simp*]:
remdups xs = *xs* ⟷ *distinct xs*
by (*metis distinct-remdups distinct-remdups-id*)

lemma *distinct-map*:
distinct (*map f xs*) = (*distinct xs* ∧ *inj-on f* (*set xs*))
by (*induct xs*) *auto*

lemma *distinct-prefix* [*simp*]:
assumes *distinct xs*
shows *distinct* (*prefix i xs*)
using *assms*
proof (*induct xs arbitrary*: *i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
  proof (*cases i*)
  case *0*
  then show *?thesis* **by** *auto*
  next
  case (*Suc nat*)
  then show *?thesis*
  using *Cons.hyps Cons.prems prefix-is-opfx set-mono-opfx* **by** *fastforce*
  qed
qed

lemma *distinct-suffix* [*simp*]:
assumes *distinct xs*
shows *distinct* (*suffix i xs*)
using *assms*
proof
(*induct xs arbitrary*: *i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases i*)
case *0*
then show *?thesis* **using** *Cons.prems* **by** *auto*
next
case (*Suc nat*)
then show *?thesis* **using** *interval-intapp-prefix-suffix Cons.hyps Cons.prems* **by** *auto*
qed
qed

lemma *distinct-conv-nth*:
$distinct\ xs =$
$(\forall\ i \le intlen\ xs.\ (\forall\ j \le intlen\ xs.\ i \neq j \longrightarrow nth\ xs\ i \neq nth\ xs\ j))$
proof
(*induction xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
by (*auto simp add*: *Cons nth-Cons interval-nth-and-set split*: *nat.split-asm*)
blast+
qed

lemma *distinct-Ex1*:
assumes *distinct xs*
$x \in set\ xs$
shows $(\exists!i.\ i \le intlen\ xs \wedge (nth\ xs\ i) = x)$
using *assms*
by (*metis distinct-conv-nth interval-nth-and-set*)

lemma *inj-on-nth*:
assumes *distinct xs*
shows $(\forall\ i \in I.\ i \le intlen\ xs \Longrightarrow inj\text{-}on\ (nth\ xs)\ I)$
using *assms*
by (*meson distinct-conv-nth inj-onI*)

lemma *bij-betw-nth*:

assumes *distinct xs*
A = {..< *intlen xs*+*1*}
B = *set xs*
shows *bij-betw* ((*nth*) *xs*) *A B*
using *assms* **unfolding** *bij-betw-def*
proof (*auto intro*!: *inj-on-nth simp*: *set-conv-nth*)
show ⋀*xa*. *distinct xs* ⟹
A = {..<*Suc* (*intlen xs*)} ⟹
B = *set xs* ⟹ *xa* < *Suc* (*intlen xs*) ⟹ *nth xs xa* ∈ *set xs*
by (*meson interval-nth-and-set less-Suc-eq-le*)
show ⋀*x*. *distinct xs* ⟹
A = {..<*Suc* (*intlen xs*)} ⟹
B = *set xs* ⟹ *x* ∈ *set xs* ⟹ *x* ∈ *nth xs* ‘ {..<*Suc* (*intlen xs*)}
using *interval-nth-and-set* **by** (*metis image-iff lessThan-iff less-Suc-eq-le*)
qed

lemma *card-distinct*:
assumes *card* (*set xs*) = *intlen xs* +*1*
shows *distinct xs*
using *assms*
proof
(*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases x1a* ∈ *set xs*)
case *True*
then show *?thesis*
by (*metis Cons*.*prems remdups*.*simps*(*2*) *set-remdups add-Suc interval-card-intlen intlen*.*simps*(*2*)
not-less-eq-eq plus-1-eq-Suc)
next
case *False*
then show *?thesis* **using** *Cons*.*hyps Cons*.*prems* **by** *auto*
qed
qed

lemma *finite-interval*:
assumes *finite A*
shows (*A*≠ {} ⟶ (∃*xs*. *set xs* = *A*))
using *assms*
proof (*induct rule*:*finite-induct*)
case *empty*
then show *?case* **by** *simp*
next
case (*insert x F*)
then show *?case* **by** (*metis insert-is-Un interval*.*set*(*1*) *interval-set-intapp*)
qed

lemma *finite-distinct-interval*:
assumes *finite A*
$A \neq \{\}$
shows $(\exists\ xs.\ set\ xs = A \wedge distinct\ xs)$
using *assms* **by** (*metis distinct-remdups finite-interval set-remdups*)

lemma *remdups-eq-state-iff* [*simp*]:
$(remdups\ xs = \langle x\rangle) = (\forall\ i \leq intlen\ xs.\ (nth\ xs\ i) = x\ )$
proof
(*induct xs*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a xs*)
then show *?case*
proof $-$
have *1*: $(\forall\, i{\leq}intlen\ (x1a \odot xs).\ nth\ (x1a \odot xs)\ i = x) =$
$(\ x1a = x \wedge (\forall\, i{\leq}intlen\ (xs).\ nth\ (xs)\ i = x))$
by *auto*
(*metis Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv*)
have *2*: $(remdups\ (x1a \odot xs) = \langle x\rangle) =$
$((if\ x1a \in set\ xs\ then\ remdups\ xs\ else\ x1a \odot remdups\ xs) = \langle x\rangle)$
by *simp*
have *3*: $((if\ x1a \in set\ xs\ then\ remdups\ xs\ else\ x1a \odot remdups\ xs) = \langle x\rangle) =$
$(remdups\ xs\ = \langle x\rangle \wedge x1a \in set\ xs)$
by *auto*
have *4*: $(remdups\ xs\ = \langle x\rangle \wedge x1a \in set\ xs) =$
$((\forall\, i{\leq}intlen\ (xs).\ nth\ (xs)\ i = x) \wedge x1a \in set\ xs)$
using *Cons.hyps* **by** *blast*
have *5*: $(\forall\, i{\leq}intlen\ (xs).\ nth\ (xs)\ i = x) \wedge x1a \in set\ xs \longrightarrow x1a = x$
by (*metis interval-nth-and-set*)
show *?thesis*
by (*metis 1 2 3 5 Cons.hyps set-remdups interval.set-intros*(*1*))
qed
qed

lemma *remdups-eq-state-right-iff* [*simp*]:
$(\langle x\rangle = remdups\ xs) = (\forall\ i \leq intlen\ xs.\ (nth\ xs\ i) = x\ )$
by (*metis remdups-eq-state-iff*)

lemma *length-remdups-leq* [*iff*]:
$intlen(remdups\ xs) \leq intlen\ xs$
by (*induct xs*) *auto*

lemma *length-remdups-eq*[*iff*]:
$(intlen\ (remdups\ xs) = intlen\ xs) = (remdups\ xs = xs)$
proof
(*induct xs*)
case (*St x*)

then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
by (*metis length-remdups-leq remdups.simps*(*2*) *add-left-cancel intlen.simps*(*2*) *not-less-eq-eq plus-1-eq-Suc*)
qed

lemma *distinct-card*:
assumes *distinct xs*
shows *card*(*set xs*) = *Suc*(*intlen xs*)
using *assms*
by (*induct xs*) *simp-all*

20.2.3 prefixes and suffixes

lemma *in-set-prefixes* [*simp*]:
xs ∈ *set* (*prefixes ys*) ⟷ *opfx xs ys*
proof
(*induct xs arbitrary*: *ys*)
case (*St x*)
then show *?case*
proof (*cases ys*)
case (*St x1*)
then show *?thesis* **using** *opfxE* **by** *force*
next
case (*Cons x21 x22*)
then show *?thesis* **unfolding** *opfx-def* **by** *auto*
qed
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases ys*)
case (*St x1*)
then show *?thesis* **by** (*simp add*: *opfx-code*(*2*))
next
case (*Cons x21 x22*)
then show *?thesis*
proof (*auto simp add*: *opfx-def*)
show *ys* = *x21* ⊙ *x22* ⟹
xs ∈ *interval.set* (*prefixes x22*) ⟹
x1a = *x21* ⟹
x22 ≠ *xs* ⟹
∃ *zs*. *x22* = *xs* ⊖ *zs*
using *Cons.hyps opfxE* **by** *blast*
show ⋀*zs*. *ys* = *x1a* ⊙ *xs* ⊖ *zs* ⟹
x21 = *x1a* ⟹ *x22* = *xs* ⊖ *zs* ⟹
x1a ⊙ *xs* ∈ (⊙) *x1a* ‘ *interval.set* (*prefixes* (*xs* ⊖ *zs*))
by (*simp add*: *Cons.hyps*)
show *ys* = *x1a* ⊙ *xs* ⟹

$x22 = xs \Longrightarrow$
$x21 = x1a \Longrightarrow$
$x1a \odot xs \in (\odot)\ x1a\ `\ interval.set\ (prefixes\ xs)$
using *Cons.hyps* **by** *blast*
qed
qed
qed

lemma *intlen-prefixes* [*simp*]:
intlen (*prefixes xs*) = *intlen xs*
by (*induction xs*) *auto*

lemma *distinct-prefixes* [*intro*]:
distinct (*prefixes xs*)
proof (*induction xs*)
case (*St x*)
then show *?case* **by** (*auto simp*: *distinct-map*)
next
case (*Cons x1a xs*)
then show *?case* **by** (*auto simp*: *distinct-map*)
(*meson inj-onI interval.inject*(*2*))
qed

lemma *prefixes-snoc* [*simp*]:
$prefixes\ (xs \ominus \langle x \rangle) = (prefixes\ xs) \ominus \langle xs \ominus \langle x \rangle \rangle$
by (*induction xs*) *auto*

lemma *intfirst-prefixes* [*simp*]:
$intfirst\ (prefixes\ xs) = \langle intfirst\ xs \rangle$
by (*cases xs*) *auto*

lemma *intlast-prefixes* [*simp*]:
intlast (*prefixes xs*) = *xs*
by (*induction xs*)
(*simp-all add*: *intlast-map interval-nth-map*)

lemma *prefixes-intapp*:
$prefixes\ (xs \ominus ys) =$
$prefixes\ xs \ominus map\ (\lambda ys'.\ xs \ominus ys')\ (prefixes\ ys)$
proof
(*induction xs arbitrary*: *ys*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof −
have *1*: $prefixes\ ((x1a \odot xs) \ominus ys) = \langle x1a \rangle \odot (map\ ((\odot)\ x1a)\ (prefixes\ (xs \ominus ys)))$

by *simp*
have *2*: *prefixes* (*xs* ⊖ *ys*) = *prefixes xs* ⊖ *map* ((⊖) *xs*) (*prefixes ys*)
by (*simp add*: *Cons.IH*)
have *3*: ⟨*x1a*⟩ ⊙ (*map* ((⊙) *x1a*) (*prefixes* (*xs* ⊖ *ys*))) =
⟨*x1a*⟩ ⊙ (*map* ((⊙) *x1a*) (*prefixes xs* ⊖ *map* ((⊖) *xs*) (*prefixes ys*)))

using *2* **by** *auto*
show *?thesis* **by** (*simp add*: *3*)
qed
qed

lemma *prefixes-eq-snoc*:
prefixes ys = *xs* ⊖ ⟨*x*⟩ ⟷
(∃ *z zs*. *ys* = *zs*⊖⟨*z*⟩ ∧ *xs* = *prefixes zs*) ∧ *x* = *ys*
proof (*cases ys rule*: *interval-rev-cases*)
case (*St x*)
then show *?thesis* **using** *interval-intapp-not-state* **by** (*metis prefixes.simps*(*1*))
next
case (*snoc ys y*)
then show *?thesis* **by** *auto*
qed

lemma *set-prefixes-eq*:
set (*prefixes xs*) = {*ys*. *opfx ys xs*}
by *auto*

lemma *card-set-prefixes* [*simp*]:
card (*set* (*prefixes xs*)) = *Suc* (*intlen xs*)
by (*simp add*: *distinct-card distinct-prefixes*)

lemma *set-prefixes-append*:
set (*prefixes* (*xs*⊖*ys*)) = *set* (*prefixes xs*) ∪ {*xs* ⊖ *ys′* | *ys′*. *ys′* ∈ *set* (*prefixes ys*)}
by (*subst prefixes-intapp*) *auto*

lemma *in-set-suffixes* [*simp*]:
xs ∈ *set*(*suffixes ys*) ⟷ *osfx xs ys*
proof (*induct ys*)
case (*St x*)
then show *?case* **using** *osfx-prefix* **by** *fastforce*
next
case (*Cons x1a ys*)
then show *?case* **by** (*simp add*: *osfx-cons*)
qed

lemma *interval-sfx-state*:
set(*suffixes* ⟨*x*⟩) = {⟨*x*⟩}
by *simp*

lemma *interval-sfx-cons*:

$set(suffixes\ (x\odot xs)) = \{x\odot xs\} \cup set(suffixes\ xs)$
by *auto*

lemma *set-suffixes-sfx*:
$set\ (suffixes\ xs) = \{suffix\ i\ xs|\ i.\ i\leq intlen\ xs\}$
proof
(*induction xs*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a xs*)
then show *?case*
proof *auto*
show $set\ (suffixes\ xs) = \{suffix\ i\ xs\ |i.\ i \leq intlen\ xs\} \Longrightarrow$
$\exists\, i.\ x1a \odot xs = (case\ i\ of\ 0 \Rightarrow x1a \odot xs \mid Suc\ m \Rightarrow suffix\ m\ xs) \wedge i \leq Suc\ (intlen\ xs)$
by *force*
show $\bigwedge i.\ set\ (suffixes\ xs) = \{suffix\ i\ xs\ |i.\ i \leq intlen\ xs\} \Longrightarrow$
$i \leq intlen\ xs \Longrightarrow \exists\, ia.\ suffix\ i\ xs = (case\ ia\ of\ 0 \Rightarrow x1a \odot xs \mid Suc\ m \Rightarrow suffix\ m\ xs) \wedge$
$ia \leq Suc\ (intlen\ xs)$
by *force*
show $\bigwedge i.\ set\ (suffixes\ xs) = \{suffix\ i\ xs\ |i.\ i \leq intlen\ xs\} \Longrightarrow$
$(case\ i\ of\ 0 \Rightarrow x1a \odot xs \mid Suc\ m \Rightarrow suffix\ m\ xs) \neq x1a \odot xs \Longrightarrow$
$i \leq Suc\ (intlen\ xs) \Longrightarrow \exists\, ia.\ (case\ i\ of\ 0 \Rightarrow x1a \odot xs \mid Suc\ m \Rightarrow$
$suffix\ m\ xs) = suffix\ ia\ xs \wedge ia \leq intlen\ xs$
by (*metis Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv*)
qed
qed

lemma *set-sfx-exists*:
$(\exists\, i\leq intlen\ xs.\ f\ (suffix\ i\ xs)) = (set\ (suffixes\ xs) \cap \{ys.\ f(ys)\} \neq \{\})$
proof *rule+*
show $\exists\, i\leq intlen\ xs.\ f\ (suffix\ i\ xs) \Longrightarrow set\ (suffixes\ xs) \cap \{ys.\ f\ ys\} = \{\} \Longrightarrow False$
by (*meson disjoint-iff-not-equal in-set-suffixes mem-Collect-eq osfx-suffix*)
show $set\ (suffixes\ xs) \cap \{ys.\ f\ ys\} \neq \{\} \Longrightarrow \exists\, i\leq intlen\ xs.\ f\ (suffix\ i\ xs)$
using *in-set-suffixes mem-Collect-eq set-suffixes-sfx* **by** *auto force*
qed

lemma *set-suffixes-osfx*:
$set(suffixes\ xs) = \{ys.\ osfx\ ys\ xs\}$
by *auto*

lemma *distinct-suffixes* [*intro*]:
$distinct(suffixes\ xs)$
proof (*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*

by *auto* (*metis Suc-n-not-n intlen.simps*(*2*) *osfx-ConsD osfx-cons osfx-order.antisym plus-1-eq-Suc*)
qed

lemma *intlen-suffixes* [*simp*]:
intlen (*suffixes xs*) = *intlen xs*
by (*induct xs*) *simp-all*

lemma *suffixes-snoc* [*simp*]:
suffixes (*xs* ⊖ ⟨*x*⟩) = (*map* (λ *ys*. *ys* ⊖ ⟨*x*⟩) (*suffixes xs*)) ⊖ ⟨⟨*x*⟩⟩
by (*induct xs*) *simp-all*

lemma *intfirst-suffixes* [*simp*]:
intfirst (*suffixes xs*) = *xs*
by (*induct xs*) *simp-all*

lemma *intlast-suffixes* [*simp*]:
intlast (*suffixes xs*) = ⟨*intlast xs*⟩
by (*induct xs*) *simp-all*

lemma *suffixes-intapp*:
suffixes (*xs* ⊖ *ys*) = *map* (λ *xs*′. *xs*′ ⊖ *ys*) (*suffixes xs*) ⊖ (*suffixes ys*)
proof
(*induction xs arbitrary*: *ys*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a ys*)
then show *?case*
  proof (*cases ys*)
  case (*St x1*)
  then show *?thesis* **by** *simp*
  next
  case (*Cons x21 x22*)
  then show *?thesis* **by** (*auto simp add*: *Cons.IH*)
  qed
qed

lemma *card-set-suffixes* [*simp*]:
card (*set* (*suffixes xs*)) = *Suc* (*intlen xs*)
by (*simp add*: *distinct-card distinct-suffixes*)

lemma *set-suffixes-intapp*:
set (*suffixes* (*xs*⊖*ys*)) = {*xs*′ ⊖ *ys* | *xs*′. *xs*′ ∈ *set* (*suffixes xs*)} ∪ *set* (*suffixes ys*)
proof (*subst suffixes-intapp*)
show *set* (*map* (λ*xs*′. *xs*′ ⊖ *ys*) (*suffixes xs*) ⊖ *suffixes ys*) =
    {*xs*′ ⊖ *ys* |*xs*′. *xs*′ ∈ *set* (*suffixes xs*)} ∪ *set* (*suffixes ys*)
proof (*cases xs*)
case (*St x1*)
then show *?thesis* **by** *simp*

next
case (*Cons x21 x22*)
then show *?thesis* **by** *force*
qed
qed

lemma *map-first-suffixes* [*simp*]:
map (λ *xs*. *nth xs 0*) (*suffixes xs*) = *xs*
by (*induct xs*) *auto*

lemma *suffixes-conv-prefixes*:
(*suffixes xs*) = *intrev* (*map intrev* (*prefixes* (*intrev xs*)))
by (*induction xs*) *auto*

lemma *prefixes-conv-suffixes*:
(*prefixes xs*) = *intrev* (*map intrev* (*suffixes* (*intrev xs*)))
by (*induction xs*) (*auto simp add*: *intrev-map*)

lemma *prefixes-intrev*:
prefixes (*intrev xs*) = *intrev* (*map intrev* (*suffixes xs*))
by (*induction xs*) *auto*

lemma *suffixes-intrev*:
suffixes (*intrev xs*) = *intrev* (*map intrev* (*prefixes xs*))
by (*induction xs*) (*auto simp add*: *intrev-map*)

lemma *nth-suffixes*:
assumes *i*$\leq$ *intlen*(*suffixes xs*)
shows *nth* (*suffixes xs*) *i* = (*suffix i xs*)
using *assms*
proof (*induct xs arbitrary*:*i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case* **by** (*simp add*: *Nitpick*.*case-nat-unfold*)
qed

lemma *suffix-suffixes*:
assumes *i* $\leq$ *intlen* (*suffixes xs*)
shows *suffix i* (*suffixes xs*) = *suffixes* (*suffix i xs*)
using *assms*
proof (*induct xs arbitrary*:*i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case* **by** (*simp add*: *Nitpick*.*case-nat-unfold*)
qed

20.2.4 filter and nfilter

lemma *sfxfilter-intlen-a*:
assumes $\exists\ ys \in set\ (suffixes\ \langle x\rangle).\ P\ ys$
shows $intlen\ (filter\ P\ (suffixes\ \langle x\rangle)) = 0$
using *assms* **by** *simp*

lemma *sfxfilter-intlen-b*:
assumes $(\exists\ ys \in set\ (suffixes\ (x \odot xs)).\ P\ ys)$
$(\exists\ ys \in set\ (suffixes\ xs).\ P\ ys)$
$P\ (x \odot xs)$
shows $intlen\ (filter\ P\ (suffixes\ (x \odot xs))) = intlen(filter\ P\ (suffixes\ xs))+1$
using *assms* **by** *simp*

lemma *sfxfilter-intlen-c*:
assumes $(\exists\ ys \in set\ (suffixes\ (x \odot xs)).\ P\ ys)$
$(\exists\ ys \in set\ (suffixes\ xs).\ P\ ys)$
$\neg\ P\ (x \odot xs)$
shows $intlen\ (filter\ P\ (suffixes\ (x \odot xs))) = intlen(filter\ P\ (suffixes\ xs))$
using *assms* **by** *simp*

lemma *sfxfilter-intlen-d*:
assumes $(\exists\ ys \in set\ (suffixes\ (x \odot xs)).\ P\ ys)$
$\neg(\exists\ ys \in set\ (suffixes\ xs).\ P\ ys)$
shows $intlen\ (filter\ P\ (suffixes\ (x \odot xs))) = 0$
using *assms* **by** *simp*

lemma *filter-intlen-a*:
assumes $\exists\ ys \in set\ \langle x\rangle.\ P\ ys$
shows $intlen\ (filter\ P\ \langle x\rangle) = 0$
using *assms* **by** *simp*

lemma *nfilter-intlen-a*:
assumes $\exists\ ys \in set\ \langle x\rangle.\ P\ ys$
shows $intlen\ (nfilter\ P\ \langle x\rangle\ n) = 0$
using *assms* **by** *simp*

lemma *filter-intlen-b*:
assumes $(\exists\ ys \in set\ (x \odot xs).\ P\ ys)$
$(\exists\ ys \in set\ (xs).\ P\ ys)$
$P\ (x)$
shows $intlen\ (filter\ P\ (x \odot xs)) = intlen(filter\ P\ (xs))+1$
using *assms* **by** *simp*

lemma *nfilter-intlen-b*:
assumes $(\exists\ ys \in set\ (x \odot xs).\ P\ ys)$
$(\exists\ ys \in set\ (xs).\ P\ ys)$
$P\ (x)$
shows $intlen\ (nfilter\ P\ (x \odot xs)\ n) = intlen(nfilter\ P\ (xs)\ (Suc\ n))+1$
using *assms* **by** *simp*

lemma *filter-intlen-c*:
assumes $(\exists\ ys \in set\ (x \odot xs).\ P\ ys)$
$(\exists\ ys \in set\ (xs).\ P\ ys)$
$\neg\ P\ (x)$
shows $intlen\ (filter\ P\ (x \odot xs)) = intlen(filter\ P\ (xs))$
using *assms* **by** *simp*

lemma *nfilter-intlen-c*:
assumes $(\exists\ ys \in set\ (x \odot xs).\ P\ ys)$
$(\exists\ ys \in set\ (xs).\ P\ ys)$
$\neg\ P\ (x)$
shows $intlen\ (nfilter\ P\ (x \odot xs)\ n) = intlen(nfilter\ P\ (xs)\ (Suc\ n))$
using *assms* **by** *simp*

lemma *filter-intlen-d*:
assumes $(\exists\ ys \in set\ (\ (x \odot xs)).\ P\ ys)$
$\neg(\exists\ ys \in set\ (\ xs).\ P\ ys)$
shows $intlen\ (filter\ P\ (\ (x \odot xs))) = 0$
using *assms* **by** *simp*

lemma *nfilter-intlen-d*:
assumes $(\exists\ ys \in set\ (\ (x \odot xs)).\ P\ ys)$
$\neg(\exists\ ys \in set\ (\ xs).\ P\ ys)$
shows $intlen\ (nfilter\ P\ (x \odot xs)\ n) = 0$
using *assms* **by** *simp*

lemma *sfxfilter-nth-a*:
assumes $(\exists\ ys \in set\ (suffixes\ \langle x\rangle).\ P\ ys)$
$j \leq intlen(filter\ P\ (suffixes\ \langle x\rangle))$
shows $nth\ (filter\ P\ (suffixes\ \langle x\rangle))\ j = \langle x\rangle$
using *assms* **by** *simp*

lemma *sfxfilter-nth-b1*:
assumes $(\exists\ ys \in set\ (suffixes\ (x \odot xs)).\ P\ ys)$
$(\exists\ ys \in set\ (suffixes\ xs).\ P\ ys)$
$P\ (x \odot xs)$
shows $nth\ (filter\ P\ (suffixes\ (x \odot xs)))\ 0 = x \odot xs$
using *assms* **by** *simp*

lemma *sfxfilter-nth-b2*:
assumes $(\exists\ ys \in set\ (suffixes\ (x \odot xs)).\ P\ ys)$
$(\exists\ ys \in set\ (suffixes\ xs).\ P\ ys)$
$P\ (x \odot xs)$
$(Suc\ j) \leq intlen\ (filter\ P\ (suffixes\ (x \odot xs)))$
shows $nth\ (filter\ P\ (suffixes\ (x \odot xs)))\ (Suc\ j) = nth\ (filter\ P\ (suffixes\ (xs)))\ j$
using *assms* **by** *auto*

lemma *sfxfilter-nth-c*:
assumes $(\exists\ ys \in set\ (suffixes\ (x \odot xs)).\ P\ ys)$
$(\exists\ ys \in set\ (suffixes\ xs).\ P\ ys)$

$\neg$ *P* (*x*⊙*xs*)
j $\leq$ *intlen* (*filter P* (*suffixes* (*x*⊙*xs*)))
shows *nth* (*filter P* (*suffixes* (*x*⊙*xs*))) *j* = *nth* (*filter P* (*suffixes* (*xs*))) *j*
using *assms* **by** *auto*

lemma *sfxfilter-nth-d*:
assumes ($\exists$ *ys* $\in$ *set* (*suffixes* (*x*⊙*xs*)). *P ys*)
$\neg$($\exists$ *ys* $\in$ *set* (*suffixes xs*). *P ys*)
j$\leq$ *intlen* (*filter P* (*suffixes* (*x*⊙*xs*)))
shows *nth* (*filter P* (*suffixes* (*x*⊙*xs*))) *j* = *x*⊙*xs*
using *assms* **by** *auto*

lemma *nfilter-nth-a*:
assumes ($\exists$ *ys* $\in$ *set* (⟨*x*⟩). *P ys*)
j$\leq$ *intlen*(*nfilter P* (⟨*x*⟩) *n*)
shows *nth* (*nfilter P* (⟨*x*⟩) *n*) *j* = *n*
using *assms* **by** *auto*

lemma *filter-nth-a*:
assumes ($\exists$ *ys* $\in$ *set* (⟨*x*⟩). *P ys*)
j$\leq$ *intlen*(*filter P* (⟨*x*⟩))
shows *nth* (*filter P* (⟨*x*⟩)) *j* = *x*
using *assms* **by** *simp*

lemma *nfilter-nth-b1*:
assumes ($\exists$ *ys* $\in$ *set* ((*x*⊙*xs*)). *P ys*)
($\exists$ *ys* $\in$ *set* (*xs*). *P ys*)
P (*x*)
shows *nth* (*nfilter P* ((*x*⊙*xs*)) *n*) *0* = *n*
using *assms* **by** *simp*

lemma *filter-nth-b1*:
assumes ($\exists$ *ys* $\in$ *set* ((*x*⊙*xs*)). *P ys*)
($\exists$ *ys* $\in$ *set* (*xs*). *P ys*)
P (*x*)
shows *nth* (*filter P* ((*x*⊙*xs*))) *0* = *x*
using *assms* **by** *simp*

lemma *nfilter-nth-b2*:
assumes ($\exists$ *ys* $\in$ *set* ((*x*⊙*xs*)). *P ys*)
($\exists$ *ys* $\in$ *set* (*xs*). *P ys*)
P (*x*)
(*Suc j*) $\leq$ *intlen* (*nfilter P* ((*x*⊙*xs*)) *n*)
shows *nth* (*nfilter P* ((*x*⊙*xs*)) *n*) (*Suc j*) = *nth* (*nfilter P* ((*xs*)) (*Suc n*)) *j*
using *assms* **by** *auto*

lemma *filter-nth-b2*:
assumes ($\exists$ *ys* $\in$ *set* ((*x*⊙*xs*)). *P ys*)
($\exists$ *ys* $\in$ *set* (*xs*). *P ys*)
P (*x*)

$(Suc\ j) \leq intlen\ (filter\ P\ (\ (x \odot xs)))$
shows $nth\ (filter\ P\ (\ (x \odot xs)))\ (Suc\ j) = nth\ (filter\ P\ (\ (xs)))\ j$
using *assms* **by** *auto*

lemma *nfilter-nth-c*:
assumes $(\exists\ ys \in set\ (\ (x \odot xs)).\ P\ ys)$
$(\exists\ ys \in set\ (\ xs).\ P\ ys)$
$\neg\ P\ (x)$
$j \leq intlen\ (nfilter\ P\ (\ (x \odot xs))\ n)$
shows $nth\ (nfilter\ P\ (\ (x \odot xs))\ n)\ j = \ nth\ (nfilter\ P\ (\ (xs))\ (Suc\ n))\ j$
using *assms* **by** *auto*

lemma *filter-nth-c*:
assumes $(\exists\ ys \in set\ (\ (x \odot xs)).\ P\ ys)$
$(\exists\ ys \in set\ (\ xs).\ P\ ys)$
$\neg\ P\ (x)$
$j \leq intlen\ (filter\ P\ (\ (x \odot xs)))$
shows $nth\ (filter\ P\ (\ (x \odot xs)))\ j = \ nth\ (filter\ P\ (\ (xs)))\ j$
using *assms* **by** *auto*

lemma *nfilter-nth-d*:
assumes $(\exists\ ys \in set\ (\ (x \odot xs)).\ P\ ys)$
$\neg(\exists\ ys \in set\ (\ xs).\ P\ ys)$
$j \leq intlen\ (nfilter\ P\ (\ (x \odot xs))\ n)$
shows $nth\ (nfilter\ P\ (\ (x \odot xs))\ n)\ j\ = n$
using *assms* **by** *auto*

lemma *filter-nth-d*:
assumes $(\exists\ ys \in set\ (\ (x \odot xs)).\ P\ ys)$
$\neg(\exists\ ys \in set\ (\ xs).\ P\ ys)$
$j \leq intlen\ (filter\ P\ (\ (x \odot xs)))$
shows $nth\ (filter\ P\ (\ (x \odot xs)))\ j\ = x$
using *assms* **by** *auto*

lemma *sfxfilter-nth-cons*:
$nth\ (filter\ P\ (suffixes\ (x \odot xs)))\ j =$
$(if\ (\exists\ ys \in set\ (suffixes\ xs).\ P\ ys)\ then$
$(if\ P\ (x \odot xs)\ then$
$(if\ j{=}0\ then\ (x \odot xs)\ else\ nth\ (filter\ P\ (suffixes\ xs))\ (j{-}1))$
$else\ nth\ (filter\ P\ (suffixes\ xs))\ j)$
$else\ (x \odot xs))$
by (*simp add*: *Nitpick.case-nat-unfold*)

lemma *sfxfilter-nth-cons-a*:
$nth\ (filter\ P\ (suffixes\ (x \odot xs)))\ j =$
$(if\ (\exists\ ys \in set\ (suffixes\ xs).\ P\ ys)\ then$
$(if\ P\ (x \odot xs)\ then$
$(case\ j\ of\ 0 \Rightarrow x \odot xs\ |\ Suc\ m \Rightarrow nth\ (filter\ P\ (suffixes\ xs))\ m)$

else nth (filter P (suffixes xs)) j)
else (x⊙xs))
by (*simp add*: *Nitpick.case-nat-unfold*)

lemma *nfilter-nth-cons*:
nth (nfilter P ((x⊙xs)) n) j =
(if (∃ ys ∈ set (xs). P ys) then
(if P (x) then
(if j=0 then (n) else nth (nfilter P (xs) (Suc n)) (j−1))
else nth (nfilter P (xs) (Suc n)) j)
else (n))
by (*simp add*: *Nitpick.case-nat-unfold*)

lemma *nfilter-nth-cons-a*:
nth (nfilter P ((x⊙xs)) n) j =
(if (∃ ys ∈ set (xs). P ys) then
(if P (x) then
(case j of 0 ⇒ n | Suc m ⇒ nth (nfilter P (xs) (Suc n)) m)
else nth (nfilter P (xs) (Suc n)) j)
else (n))
by (*simp add*: *Nitpick.case-nat-unfold*)

lemma *filter-nth-cons*:
nth (filter P ((x⊙xs))) j =
(if (∃ ys ∈ set (xs). P ys) then
(if P (x) then
(if j=0 then (x) else nth (filter P (xs)) (j−1))
else nth (filter P (xs)) j)
else (x))
by (*simp add*: *Nitpick.case-nat-unfold*)

lemma *filter-nth-cons-a*:
nth (filter P ((x⊙xs))) j =
(if (∃ ys ∈ set (xs). P ys) then
(if P (x) then
(case j of 0 ⇒ x | (Suc m) ⇒ nth (filter P (xs)) m)
else nth (filter P (xs)) j)
else (x))
by (*simp add*: *Nitpick.case-nat-unfold*)

lemma *sfxfilter-nth*:
assumes (∃ *ys* ∈ *set* (*suffixes xs*). *P ys*)
i ≤ *intlen* (*filter P* (*suffixes xs*))
shows *P* (*nth* (*filter P* (*suffixes xs*)) *i*)
using *assms*
proof
(*induction xs arbitrary*: *i*)
case (*St x*)
then show *?case* **by** *simp*
next

case (*Cons x1a xs*)
then show *?case*
proof (*cases* $\exists$ *a*$\in$*set* (*suffixes xs*). *P a*)
show ($\bigwedge$*i*. $\exists$ *a*$\in$*set* (*suffixes xs*). *P a* $\Longrightarrow$
i $\leq$ *intlen* (*filter P* (*suffixes xs*)) $\Longrightarrow$
P (*nth* (*filter P* (*suffixes xs*)) *i*)) $\Longrightarrow$
$\exists$ *a*$\in$*set* (*suffixes* (*x1a* $\odot$ *xs*)). *P a* $\Longrightarrow$
i $\leq$ *intlen* (*filter P* (*suffixes* (*x1a* $\odot$ *xs*))) $\Longrightarrow$
$\exists$ *a*$\in$*set* (*suffixes xs*). *P a* $\Longrightarrow$
P (*nth* (*filter P* (*suffixes* (*x1a* $\odot$ *xs*))) *i*)
by (*auto simp add*: *nat.split-sels*(*2*))
show ($\bigwedge$*i*. $\exists$ *a*$\in$*set* (*suffixes xs*). *P a* $\Longrightarrow$
i $\leq$ *intlen* (*filter P* (*suffixes xs*)) $\Longrightarrow$
P (*nth* (*filter P* (*suffixes xs*)) *i*)) $\Longrightarrow$
$\exists$ *a*$\in$*set* (*suffixes* (*x1a* $\odot$ *xs*)). *P a* $\Longrightarrow$
i $\leq$ *intlen* (*filter P* (*suffixes* (*x1a* $\odot$ *xs*))) $\Longrightarrow$
$\neg$ ($\exists$ *a*$\in$*set* (*suffixes xs*). *P a*) $\Longrightarrow$
P (*nth* (*filter P* (*suffixes* (*x1a* $\odot$ *xs*))) *i*)
by *simp*
qed
qed

lemma *nfilter-intlen*:
assumes ($\exists$ *x* $\in$ *set xs*. *P x*)
shows *intlen*(*nfilter P xs n*) = *intlen* (*filter P xs*)
using *assms*
proof
(*induction xs arbitrary*: *n*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case* **by** *simp*
qed

lemma *nfilter-upper-bound*:
assumes ($\exists$ *x* $\in$ *set xs*. *P x*)
i$\leq$*intlen* (*nfilter P xs n*)
shows (*nth* (*nfilter P xs n*) *i*) $\leq$ *n*+*intlen xs*
using *assms*
proof
(*induct xs arbitrary*: *i n*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases* $\exists$ *x* $\in$ *set xs*. *P x*)

show $(\bigwedge i\ n.\ \exists\, a{\in}set\ xs.\ P\ a \Longrightarrow$
$i \le intlen\ (nfilter\ P\ xs\ n) \Longrightarrow nth\ (nfilter\ P\ xs\ n)\ i \le n + intlen\ xs) \Longrightarrow$
$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$i \le intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists\, x{\in}set\ xs.\ P\ x \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ i \le n + intlen\ (x1a \odot xs)$
proof *auto*
show $\bigwedge x.\ (\bigwedge i\ n.\ i \le intlen\ (nfilter\ P\ xs\ n) \Longrightarrow nth\ (nfilter\ P\ xs\ n)\ i \le n + intlen\ xs) \Longrightarrow$
$i \le Suc\ (intlen\ (nfilter\ P\ xs\ (Suc\ n))) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$(case\ i\ of\ 0 \Rightarrow n \mid Suc\ k \Rightarrow nth\ (nfilter\ P\ xs\ (Suc\ n))\ k) \le Suc\ (n + intlen\ xs)$
by (*cases i*, *simp*, *fastforce*)
show $\bigwedge x.\ (\bigwedge i\ n.\ i \le intlen\ (nfilter\ P\ xs\ n) \Longrightarrow nth\ (nfilter\ P\ xs\ n)\ i \le n + intlen\ xs) \Longrightarrow$
$i \le intlen\ (nfilter\ P\ xs\ (Suc\ n)) \Longrightarrow$
$x \in set\ xs \Longrightarrow P\ x \Longrightarrow \neg\ P\ x1a \Longrightarrow nth\ (nfilter\ P\ xs\ (Suc\ n))\ i \le Suc\ (n + intlen\ xs)$
by *force*
qed
show $(\bigwedge i\ n.\ \exists\, a{\in}set\ xs.\ P\ a \Longrightarrow$
$i \le intlen\ (nfilter\ P\ xs\ n) \Longrightarrow nth\ (nfilter\ P\ xs\ n)\ i \le n + intlen\ xs) \Longrightarrow$
$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$i \le intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\neg\ (\exists\, x{\in}set\ xs.\ P\ x) \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ i \le n + intlen\ (x1a \odot xs)$
by *simp*
qed
qed

lemma *nfilter-lower-bound*:
assumes $(\exists\ x \in set\ xs.\ P\ x)$
$i {\le} intlen\ (nfilter\ P\ xs\ n)$
shows $n \le (nth\ (nfilter\ P\ xs\ n)\ i)$
using *assms*
proof
(*induction xs arbitrary*: *i n*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases* $\exists\ x \in set\ xs.\ P\ x$)
show $(\bigwedge i\ n.\ \exists\, a{\in}set\ xs.\ P\ a \Longrightarrow i \le intlen\ (nfilter\ P\ xs\ n) \Longrightarrow n \le nth\ (nfilter\ P\ xs\ n)\ i) \Longrightarrow$
$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$i \le intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists\, x{\in}set\ xs.\ P\ x \Longrightarrow$
$n \le nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ i$
proof *auto*
show $\bigwedge x.\ (\bigwedge i\ n.\ i \le intlen\ (nfilter\ P\ xs\ n) \Longrightarrow n \le nth\ (nfilter\ P\ xs\ n)\ i) \Longrightarrow$
$i \le Suc\ (intlen\ (nfilter\ P\ xs\ (Suc\ n))) \Longrightarrow$

$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$n \leq (case\ i\ of\ 0 \Rightarrow n \mid Suc\ k \Rightarrow nth\ (nfilter\ P\ xs\ (Suc\ n))\ k)$
by (*metis Suc-leD add-le-imp-le-left nat.split-sels*(1) *order-refl plus-1-eq-Suc*)
show $\bigwedge x. (\bigwedge i\ n.\ i \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow n \leq nth\ (nfilter\ P\ xs\ n)\ i) \Longrightarrow$
$i \leq intlen\ (nfilter\ P\ xs\ (Suc\ n)) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$n \leq nth\ (nfilter\ P\ xs\ (Suc\ n))\ i$
using *Suc-leD* **by** *blast*
qed
show $(\bigwedge i\ n.\ \exists a \in set\ xs.\ P\ a \Longrightarrow i \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow n \leq nth\ (nfilter\ P\ xs\ n)\ i) \Longrightarrow$
$\exists a \in set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$i \leq intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\neg\ (\exists x \in set\ xs.\ P\ x) \Longrightarrow$
$n \leq nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ i$
by *auto*
qed
qed

lemma *nfilter-filter*:
assumes $(\exists\ x \in set\ xs.\ P\ x)$
$i \leq intlen\ (nfilter\ P\ xs\ n)$
shows $(nth\ xs\ ((nth\ (nfilter\ P\ xs\ n)\ i) - n)\ ) = (nth\ (filter\ P\ xs)\ i)$
using *assms*
proof
(*induct xs arbitrary*: *i n*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases* $\exists\ x \in set\ xs.\ P\ x$)
show $(\bigwedge i\ n.\ \exists a \in set\ xs.$
$P\ a \Longrightarrow$
$i \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ xs\ (nth\ (nfilter\ P\ xs\ n)\ i - n) = nth\ (filter\ P\ xs)\ i) \Longrightarrow$
$\exists a \in set\ (x1a \odot xs).$
$P\ a \Longrightarrow$
$i \leq intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x \in set\ xs.$
$P\ x \Longrightarrow$
$nth\ (x1a \odot xs)\ (nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ i - n) =$
$nth\ (filter\ P\ (x1a \odot xs))\ i$
proof *auto*
show $\bigwedge x. (\bigwedge i\ n.$
$i \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$

$nth\ xs\ (nth\ (nfilter\ P\ xs\ n)\ i - n) = nth\ (filter\ P\ xs)\ i) \Longrightarrow$
$i \leq Suc\ (intlen\ (nfilter\ P\ xs\ (Suc\ n))) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$(case\ (case\ i\ of\ 0 \Rightarrow n \mid Suc\ k \Rightarrow nth\ (nfilter\ P\ xs\ (Suc\ n))\ k) - n\ of\ 0 \Rightarrow x1a$
$\mid Suc\ x \Rightarrow nth\ xs\ x) =$
$(case\ i\ of\ 0 \Rightarrow x1a \mid Suc\ k \Rightarrow nth\ (filter\ P\ xs)\ k)$
proof (*cases i*)
show $\bigwedge x.\ (\bigwedge i\ n.\ i \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ xs\ (nth\ (nfilter\ P\ xs\ n)\ i - n) = nth\ (filter\ P\ xs)\ i) \Longrightarrow$
$i \leq Suc\ (intlen\ (nfilter\ P\ xs\ (Suc\ n))) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$i = 0 \Longrightarrow$
$(case\ (case\ i\ of\ 0 \Rightarrow n \mid Suc\ k \Rightarrow nth\ (nfilter\ P\ xs\ (Suc\ n))\ k) - n\ of\ 0 \Rightarrow x1a$
$\mid Suc\ x \Rightarrow nth\ xs\ x) =$
$(case\ i\ of\ 0 \Rightarrow x1a \mid Suc\ k \Rightarrow nth\ (filter\ P\ xs)\ k)$
by *simp*
show $\bigwedge x\ nat.$
$(\bigwedge i\ n.\ i \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ xs\ (nth\ (nfilter\ P\ xs\ n)\ i - n) = nth\ (filter\ P\ xs)\ i) \Longrightarrow$
$i \leq Suc\ (intlen\ (nfilter\ P\ xs\ (Suc\ n))) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$i = Suc\ nat \Longrightarrow$
$(case\ (case\ i\ of\ 0 \Rightarrow n \mid Suc\ k \Rightarrow nth\ (nfilter\ P\ xs\ (Suc\ n))\ k) - n\ of\ 0 \Rightarrow x1a$
$\mid Suc\ x \Rightarrow nth\ xs\ x) =$
$(case\ i\ of\ 0 \Rightarrow x1a \mid Suc\ k \Rightarrow nth\ (filter\ P\ xs)\ k)$
by *simp*
(*metis* (*full-types*) *Suc-diff-le diff-Suc-Suc nfilter-lower-bound old.nat.simps*(*5*))
qed
show $\bigwedge x.$
$(\bigwedge i\ n.\ i \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ xs\ (nth\ (nfilter\ P\ xs\ n)\ i - n) = nth\ (filter\ P\ xs)\ i) \Longrightarrow$
$i \leq intlen\ (nfilter\ P\ xs\ (Suc\ n)) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$(case\ nth\ (nfilter\ P\ xs\ (Suc\ n))\ i - n\ of\ 0 \Rightarrow x1a \mid Suc\ x \Rightarrow nth\ xs\ x) =$
$nth\ (filter\ P\ xs)\ i$
by (*metis Nitpick.case-nat-unfold Suc-eq-plus1 Suc-le-lessD diff-diff-left neq0-conv nfilter-lower-bound zero-less-diff*)
qed
show $(\bigwedge i\ n.$
$\exists a \in set\ xs.\ P\ a \Longrightarrow$
$i \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ xs\ (nth\ (nfilter\ P\ xs\ n)\ i - n) = nth\ (filter\ P\ xs)\ i) \Longrightarrow$

```
        ∃ a∈set (x1a ⊙ xs).
         P a ⟹
         i ≤ intlen (nfilter P (x1a ⊙ xs) n) ⟹
        ¬ (∃ x∈set xs. P x) ⟹
         nth (x1a ⊙ xs) (nth (nfilter P (x1a ⊙ xs) n) i − n) =
         nth (filter P (x1a ⊙ xs)) i
    by auto
  qed
qed


lemma set-filter [simp]:
assumes (∃ x ∈ set xs. P x)
shows set (filter P xs) = {x. x ∈ set xs ∧ P x}
using assms
proof
 (induct xs)
 case (St x)
 then show ?case by (simp add: Collect-conv-if)
 next
 case (Cons x1a xs)
 then show ?case by auto
qed

lemma set-nfilter [simp]:
assumes (∃ x ∈ set xs. P x)
shows set (nfilter P xs n) = {n+k|k . k≤ intlen xs ∧ P (nth xs k)}
using assms
proof
 (induction xs arbitrary: n)
 case (St x)
 then show ?case by auto
 next
 case (Cons x1a xs)
 then show ?case
  proof (auto simp add: nat.split)
    show ⋀y k. (⋀n. ∃ x∈set xs. P x ⟹
               set (nfilter P xs n) = {n + k |k. k ≤ intlen xs ∧ P (nth xs k)}) ⟹
        P x1a ⟹
        y ∈ set xs ⟹
        P y ⟹
        0 < k ⟹
        k ≤ Suc (intlen xs) ⟹
        ∀ x2. k = Suc x2 ⟶ P (nth xs x2) ⟹
        ∃ ka. k = Suc ka ∧ ka ≤ intlen xs ∧ P (nth xs ka)
     by (metis Suc-le-mono gr0-implies-Suc)
   show ⋀k. P x1a ⟹
         ∀ x∈set xs. ¬ P x ⟹
         k ≤ Suc (intlen xs) ⟹
         ∀ x2. k = Suc x2 ⟶ P (nth xs x2) ⟹
```

```
          k = 0
    by (metis Suc-le-mono le-SucE le-zero-eq nth-set zero-induct)
  show ⋀x y k.
      (⋀n. ∃x∈set xs. P x ⟹
            set (nfilter P xs n) = {n + k |k. k ≤ intlen xs ∧ P (nth xs k)}) ⟹
      x ∈ set xs ⟹
      P x ⟹
      P x1a ⟹
      y ∈ set xs ⟹
      P y ⟹
      0 < k ⟹
      k ≤ Suc (intlen xs) ⟹
      ∀x2. k = Suc x2 ⟶ P (nth xs x2) ⟹
      ∃ka. k = Suc ka ∧ ka ≤ intlen xs ∧ P (nth xs ka)
    by (metis Suc-le-mono gr0-implies-Suc)
  show ⋀x y k.
      (⋀n. ∃x∈set xs. P x ⟹
            set (nfilter P xs n) = {n + k |k. k ≤ intlen xs ∧ P (nth xs k)}) ⟹
      x ∈ set xs ⟹
      P x ⟹
      ¬ P x1a ⟹
      y ∈ set xs ⟹
      P y ⟹
      k ≤ Suc (intlen xs) ⟹
      0 < k ⟹
      ∀x2. k = Suc x2 ⟶ P (nth xs x2) ⟹
      ∃ka. k = Suc ka ∧ ka ≤ intlen xs ∧ P (nth xs ka)
     by (metis Suc-less-eq gr0-implies-Suc not-less)
  qed
qed

lemma set-minus-filter-out:
assumes (∃ z ∈ set xs. (λ x. ¬(x =y)) z)
 shows  set xs − {y} = set(filter (λ x. ¬(x =y)) xs)
using assms
by (induct xs) auto

lemma filter-filter [simp]:
assumes ∃ x ∈ set (filter Q xs). P x
        ∃ x ∈ set xs. Q x
        ∃ x ∈ set xs. P x ∧ Q x
shows filter P (filter Q xs) = filter (λx. P x ∧ Q x) xs
using assms
by (induct xs) auto

lemma length-nfilter-le [simp]:
 intlen (nfilter P xs n) ≤ intlen xs
by (induct xs arbitrary: n) (auto simp add: le-SucI)

lemma length-filter-le [simp]:
```

intlen (*filter P xs*) ≤ *intlen xs*
by (*induct xs*) (*auto simp add*: *le-SucI*)

lemma *sfxfilter-bound*:
assumes (∃ *ys* ∈ *set* (*suffixes xs*). *P ys*)
shows *intlen* (*filter P* (*suffixes xs*)) ≤ *intlen xs*
using *assms* **by** (*metis length-filter-le intlen-suffixes*)

lemma *filter-bound*:
assumes (∃ *ys* ∈ *set* (*xs*). *P ys*)
shows *intlen* (*filter P* (*xs*)) ≤ *intlen xs*
using *assms* **by** *auto*

lemma *sfxfilter-nth-bound*:
assumes (∃ *ys* ∈ *set* (*suffixes xs*). *P ys*)
j ≤ *intlen* (*filter P* (*suffixes xs*))
shows *intlen* ((*nth* (*filter P* (*suffixes xs*)) *j*)) ≤ *intlen xs*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *set-filter in-set-suffixes mem-Collect-eq nth-set osfx-intlen*)

lemma *sfxfilter-nth-suffix*:
assumes (∃ *ys* ∈ *set* (*suffixes xs*). *P ys*)
j≤ *intlen* (*filter P* (*suffixes xs*))
shows *nth* (*filter P* (*suffixes xs*)) *j* =
suffix (*intlen xs* − *intlen*(*nth*(*filter P* (*suffixes xs*)) *j*)) *xs*
using *assms*
proof
(*induction xs arbitrary*: *j*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases j*)
show (⋀*j*. ∃ *a*∈*set* (*suffixes xs*). *P a* ⟹
j ≤ *intlen* (*filter P* (*suffixes xs*)) ⟹
nth (*filter P* (*suffixes xs*)) *j* =
suffix (*intlen xs* − *intlen* (*nth* (*filter P* (*suffixes xs*)) *j*)) *xs*) ⟹
∃ *a*∈*set* (*suffixes* (*x1a* ⊙ *xs*)). *P a* ⟹
j ≤ *intlen* (*filter P* (*suffixes* (*x1a* ⊙ *xs*))) ⟹
j = *0* ⟹
nth (*filter P* (*suffixes* (*x1a* ⊙ *xs*))) *j* =
suffix (*intlen* (*x1a* ⊙ *xs*) − *intlen* (*nth* (*filter P* (*suffixes* (*x1a* ⊙ *xs*))) *j*)) (*x1a* ⊙ *xs*)
by (*simp-all add*: *Suc-diff-le sfxfilter-nth-bound*)
show ⋀*nat*. (⋀*j*. ∃ *a*∈*set* (*suffixes xs*). *P a* ⟹
j ≤ *intlen* (*filter P* (*suffixes xs*)) ⟹
nth (*filter P* (*suffixes xs*)) *j* =
suffix (*intlen xs* − *intlen* (*nth* (*filter P* (*suffixes xs*)) *j*)) *xs*) ⟹
∃ *a*∈*set* (*suffixes* (*x1a* ⊙ *xs*)). *P a* ⟹
j ≤ *intlen* (*filter P* (*suffixes* (*x1a* ⊙ *xs*))) ⟹

$j = Suc\ nat \Longrightarrow$
$nth\ (filter\ P\ (suffixes\ (x1a \odot xs)))\ j =$
$suffix\ (intlen\ (x1a \odot xs) - intlen\ (nth\ (filter\ P\ (suffixes\ (x1a \odot xs)))\ j))\ (x1a \odot xs)$
proof $(cases\ (\exists\ ys \in set\ (suffixes\ xs).\ P\ ys))$
show $\bigwedge nat.\ (\bigwedge j.\ \exists a{\in}set\ (suffixes\ xs).\ P\ a \Longrightarrow$
$j \le intlen\ (filter\ P\ (suffixes\ xs)) \Longrightarrow$
$nth\ (filter\ P\ (suffixes\ xs))\ j =$
$suffix\ (intlen\ xs - intlen\ (nth\ (filter\ P\ (suffixes\ xs))\ j))\ xs) \Longrightarrow$
$\exists a{\in}set\ (suffixes\ (x1a \odot xs)).\ P\ a \Longrightarrow$
$j \le intlen\ (filter\ P\ (suffixes\ (x1a \odot xs))) \Longrightarrow$
$j = Suc\ nat \Longrightarrow$
$\exists ys{\in}set\ (suffixes\ xs).\ P\ ys \Longrightarrow$
$nth\ (filter\ P\ (suffixes\ (x1a \odot xs)))\ j =$
$suffix\ (intlen\ (x1a \odot xs) - intlen\ (nth\ (filter\ P\ (suffixes\ (x1a \odot xs)))\ j))\ (x1a \odot xs)$
proof *auto*
show $\bigwedge nat\ x.$
$(\bigwedge j.\ j \le intlen\ (filter\ P\ (suffixes\ xs)) \Longrightarrow$
$nth\ (filter\ P\ (suffixes\ xs))\ j =$
$suffix\ (intlen\ xs - intlen\ (nth\ (filter\ P\ (suffixes\ xs))\ j))\ xs) \Longrightarrow$
$nat \le intlen\ (filter\ P\ (suffixes\ xs)) \Longrightarrow$
$j = Suc\ nat \Longrightarrow$
$osfx\ x\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$P\ (x1a \odot xs) \Longrightarrow$
$nth\ (filter\ P\ (suffixes\ xs))\ nat =$
$(case\ Suc\ (intlen\ xs) - intlen\ (nth\ (filter\ P\ (suffixes\ xs))\ nat)\ of\ 0 \Rightarrow x1a \odot xs$
$\mid Suc\ m \Rightarrow suffix\ m\ xs)$
by (*metis* (*full-types*) *Suc-diff-le diff-le-self interval-suffix-suc osfx-intlen osfx-suffix suffix.simps*(2))
show $\bigwedge nat\ x.$
$(\bigwedge j.\ j \le intlen\ (filter\ P\ (suffixes\ xs)) \Longrightarrow$
$nth\ (filter\ P\ (suffixes\ xs))\ j =$
$suffix\ (intlen\ xs - intlen\ (nth\ (filter\ P\ (suffixes\ xs))\ j))\ xs) \Longrightarrow$
$Suc\ nat \le intlen\ (filter\ P\ (suffixes\ xs)) \Longrightarrow$
$j = Suc\ nat \Longrightarrow$
$osfx\ x\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$\neg\ P\ (x1a \odot xs) \Longrightarrow$
$nth\ (filter\ P\ (suffixes\ xs))\ (Suc\ nat) =$
$(case\ Suc\ (intlen\ xs) - intlen\ (nth\ (filter\ P\ (suffixes\ xs))\ (Suc\ nat))\ of\ 0 \Rightarrow x1a \odot xs$
$\mid Suc\ m \Rightarrow suffix\ m\ xs)$
by (*metis* (*full-types*) *Suc-diff-le diff-le-self interval-suffix-suc osfx-intlen osfx-suffix suffix.simps*(2))
qed
show $\bigwedge nat.$
$(\bigwedge j.\ \exists a{\in}set\ (suffixes\ xs).\ P\ a \Longrightarrow$
$j \le intlen\ (filter\ P\ (suffixes\ xs)) \Longrightarrow$
$nth\ (filter\ P\ (suffixes\ xs))\ j =$
$suffix\ (intlen\ xs - intlen\ (nth\ (filter\ P\ (suffixes\ xs))\ j))\ xs) \Longrightarrow$
$\exists a{\in}set\ (suffixes\ (x1a \odot xs)).\ P\ a \Longrightarrow$

$j \le$ *intlen* (*filter P* (*suffixes* (*x1a* $\odot$ *xs*))) $\Longrightarrow$
$j =$ *Suc nat* $\Longrightarrow$
$\neg$ ($\exists$ *ys*$\in$*set* (*suffixes xs*). *P ys*) $\Longrightarrow$
nth (*filter P* (*suffixes* (*x1a* $\odot$ *xs*))) *j* $=$
suffix (*intlen* (*x1a* $\odot$ *xs*) $-$ *intlen* (*nth* (*filter P* (*suffixes* (*x1a* $\odot$ *xs*))) *j*)) (*x1a* $\odot$ *xs*)
by *simp*
qed
qed
qed

lemma *initfilter-sfxfilter-exists*:
($\exists$ *ys* $\in$ *set* (*suffixes xs*). *P* ((*nth ys 0*))) $=$ ($\exists$ *x* $\in$ *set xs*. *P x*)
by (*metis interval-nth-and-set interval-nth-map intlen-suffixes map-first-suffixes*)

lemma *initfilter-sfxfilter*:
assumes $\exists$ *ys* $\in$ *set* (*suffixes xs*). *P* ((*nth ys 0*))
shows *filter P xs* $=$ *map* (λ*s*. (*nth s 0*)) (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*))

using *assms*
proof
(*induction xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof *auto*
show $\bigwedge$*y*. ($\exists$*ys*$\in$*set* (*suffixes xs*). *P* (*nth ys 0*) $\Longrightarrow$
filter P xs $=$
map (λ*s*. *nth s 0*) (*filter* (λ*ys*. *P* (*nth ys 0*)) (*suffixes xs*))) $\Longrightarrow$
osfx y xs $\Longrightarrow$
P (*nth y 0*) $\Longrightarrow$
P x1a $\Longrightarrow$
$\exists$*x*$\in$*set xs*. *P x*
by (*metis in-set-suffixes interval-nth-and-set interval-nth-map intlen-suffixes map-first-suffixes*)
show $\bigwedge$*x*. $\forall$*y*$\in$*set* (*suffixes xs*). $\neg$ *P* (*nth y 0*) $\Longrightarrow$ *P x1a* $\Longrightarrow$ *x* $\in$ *set xs* $\Longrightarrow$ *P x* $\Longrightarrow$ *False*
by (*metis interval-nth-and-set interval-nth-map intlen-suffixes map-first-suffixes*)
show $\bigwedge$*ys y*.
($\exists$*ys*$\in$*set* (*suffixes xs*). *P* (*nth ys 0*) $\Longrightarrow$
filter P xs $=$
map (λ*s*. *nth s 0*) (*filter* (λ*ys*. *P* (*nth ys 0*)) (*suffixes xs*))) $\Longrightarrow$
osfx ys xs $\Longrightarrow$
P (*nth ys 0*) $\Longrightarrow$
osfx y xs $\Longrightarrow$
P (*nth y 0*) $\Longrightarrow$
P x1a $\Longrightarrow$
$\exists$*x*$\in$*set xs*. *P x*
by (*metis in-set-suffixes interval-nth-and-set interval-nth-map intlen-suffixes map-first-suffixes*)
show $\bigwedge$*ys y*.
($\exists$*ys*$\in$*set* (*suffixes xs*). *P* (*nth ys 0*) $\Longrightarrow$

filter P xs =
map (λ*s*. *nth s 0*) (*filter* (λ*ys*. *P* (*nth ys 0*)) (*suffixes xs*))) ⟹
osfx ys xs ⟹
P (*nth ys 0*) ⟹
osfx y xs ⟹
P (*nth y 0*) ⟹
¬ *P x1a* ⟹
∀ *x*∈*set xs*. ¬ *P x* ⟹
∃ *y*. ⟨*y*⟩ = *filter* (λ*ys*. *P* (*nth ys 0*)) (*suffixes xs*) ∧ *nth y 0* = *x1a*
by (*metis in-set-suffixes interval-nth-and-set interval-nth-map intlen-suffixes map-first-suffixes*)
qed
qed

lemma *filter-nth*:
assumes (∃ *x* ∈ *set xs*. *P x*)
i≤ *intlen* (*filter P xs*)
shows (∃ *k* ≤ *intlen xs*. *nth*(*filter P xs*) *i* = *nth xs k*)
proof −
have *1*: ∃ *ys* ∈ *set* (*suffixes xs*). *P* ((*nth ys 0*))
using *assms*
by (*simp add*: *initfilter-sfxfilter-exists*)
have *2*: ⋀*i*. *i*≤ *intlen* (*filter P xs*) ⟶
nth(*filter P xs*) *i* = *nth* (*map* (λ*s*. (*nth s 0*)) (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*))) *i*
using *1 initfilter-sfxfilter* **by** *force*
have *3*: ⋀*i*. *i* ≤ *intlen* (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*)) ⟶
nth (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*)) *i* =
suffix (*intlen xs* − *intlen*(*nth*(*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*)) *i*)) *xs*

by (*meson 1 sfxfilter-nth-suffix*)
have *4*: ⋀*i*. *i* ≤ *intlen* (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*)) ⟶
nth (*map* (λ*s*. (*nth s 0*)) (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*))) *i* =
(λ*s*. (*nth s 0*)) (*nth* (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*)) *i*)

using *interval-nth-map* **by** *blast*
have *5*: ⋀*i*. *i* ≤ *intlen* (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*)) ⟶
(λ*s*. (*nth s 0*)) (*nth* (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*)) *i*) =
(λ*s*. (*nth s 0*)) (*suffix* (*intlen xs* − *intlen*(*nth*(*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*)) *i*)) *xs*)

using *3* **by** *auto*
have *6*: (∃ *k* ≤ *intlen xs*.
nth (*map* (λ*s*. (*nth s 0*)) (*filter* (λ*ys*. *P*(*nth ys 0*)) (*suffixes xs*))) *i* = *nth xs k*)
by (*metis* (*no-types*, *lifting*) *2 set-filter assms interval-nth-and-set*
mem-Collect-eq nth-set)
show *?thesis*
by (*simp add*: *2 6 assms*)
qed

lemma *interval-sfx-nth-zero*:
set xs = {(*nth ys 0*)| *ys*. *ys* ∈ *set*(*suffixes xs*) }
proof

(*induct xs*)
case (*St x*)
then show *?case*
by *auto*
next
case (*Cons x1a xs*)
then show *?case*
by *auto* (*metis interval-nth-zero*)
qed

lemma *interval-sfx-1*:
assumes *ys* ∈ *set*(*suffixes xs*)
shows (*nth ys 0*) ∈ *set xs*
using *assms interval-sfx-nth-zero* **by** *fastforce*

lemma *sum-length-filter-compl-help*:
assumes ∃ *x* ∈ *set xs*. *P x*
∃ *x* ∈ *set xs*. ¬ *P x*
shows *intlen xs* > *0*
using *assms*
proof
(*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
using *interval-intlen-cons-1* **by** *blast*
qed

lemma *filter-id-conv*:
assumes ∃ *x* ∈ *set xs*. *P x*
shows (*filter P xs* = *xs*) = (∀ *x*∈*set xs*. *P x*)
using *assms*
proof
(*induct xs*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a xs*)
then show *?case*
by (*auto simp add*: *interval-set-nonempty*)
(*metis length-filter-le Suc-n-not-le-n intlen.simps*(*2*) *plus-1-eq-Suc*)
qed

lemma *sum-length-filter-compl*:
assumes ∃ *x* ∈ *set xs*. *P x*
∃ *x* ∈ *set xs*. ¬ *P x*
shows *intlen*(*filter P xs*) + *intlen*(*filter* (λ*x*. ¬*P x*) *xs*) +*1* = *intlen xs*
using *assms*

```
proof
 (induct xs)
 case (St x)
 then show ?case by auto
 next
 case (Cons x1a xs)
 then show ?case
  proof (cases (∃y∈set xs. P y))
  case True
  then show ?thesis
   proof (cases (∃y∈set xs. ¬ P y))
   case True
   then show ?thesis
     proof auto
      show ⋀y x. y ∈ set xs ⟹
           ¬ P y ⟹
           x ∈ set xs ⟹
           P x ⟹
           ¬ P x1a ⟹
           Suc (intlen (filter P xs) + intlen (filter (λx. ¬ P x) xs)) = intlen xs
       using Cons.hyps by fastforce
      show ⋀y x. y ∈ set xs ⟹
           ¬ P y ⟹
           x ∈ set xs ⟹
           P x ⟹
           P x1a ⟹ Suc (intlen (filter P xs) + intlen (filter (λx. ¬ P x) xs)) = intlen xs
       using Cons.hyps by fastforce
      show ⋀y. y ∈ set xs ⟹
         ∀x∈set xs. ¬ P x ⟹ ¬ P x1a ⟹ Suc (intlen (filter (λx. ¬ P x) xs)) = intlen xs
       using Cons.prems(1) by auto
      show ⋀y. y ∈ set xs ⟹
         ∀x∈set xs. ¬ P x ⟹ P x1a ⟹ intlen (filter (λx. ¬ P x) xs) = intlen xs
      by (metis filter-id-conv)
     qed
   next
   case False
   then show ?thesis
     proof auto
      show ⋀y. ∀y∈set xs. P y ⟹ P x1a ⟹ y ∈ set xs ⟹
            Suc (intlen (filter P xs)) = intlen xs
       using Cons.prems by auto
      show P x1a ⟹ set xs = {} ⟹ intlen xs = 0
       using True by blast
      show ⋀y. ∀y∈set xs. P y ⟹ ¬ P x1a ⟹ y ∈ set xs ⟹
            intlen (filter P xs) = intlen xs
       by (metis filter-id-conv)
      show ¬ P x1a ⟹ set xs = {} ⟹ intlen xs = 0
       using interval-set-nonempty by blast
      qed
   qed
```

```
 next
 case False
 then show ?thesis
     proof auto
      show ⋀y. ∀x∈set xs. ¬ P x ⟹
         ¬ P x1a ⟹ y ∈ set xs ⟹
            Suc (intlen (filter (λx. ¬ P x) xs)) = intlen xs
       using Cons.prems(1) by auto
      show ¬ P x1a ⟹ set xs = {} ⟹ intlen xs = 0
       using interval-set-nonempty by blast
      show ⋀y. ∀x∈set xs. ¬ P x ⟹
           P x1a ⟹ y ∈ set xs ⟹
            intlen (filter (λx. ¬ P x) xs) = intlen xs
       by (metis filter-id-conv)
      show P x1a ⟹ set xs = {} ⟹ intlen xs = 0
      using interval-set-nonempty by blast
    qed
 qed
qed

lemma filter-intlen-imp:
assumes ∃ x ∈ set xs. P x ∧ Q x
shows   intlen (filter (λx. P x ∧ Q x) xs) ≤ intlen (filter P xs)
using assms
proof (induct xs)
case (St x)
then show ?case by simp
next
case (Cons x1a xs)
then show ?case by force
qed

lemma subset-filter:
assumes (∃ x ∈ set xs. P x)
shows    set (filter P xs) ≤ set (filter (λ x. P x ∨ Q x) xs)
proof −
 have 1: (∃ x ∈ set xs. P x ∨ Q x)
   using assms by blast
 have 2:  set (filter (λ x. P x ∨ Q x) xs) = {x ∈ set xs. P x ∨ Q x}
   using assms set-filter[of xs (λ x. P x ∨ Q x)] by blast
 have 3: set (filter P xs) = {x ∈ set xs. P x}
   using assms set-filter by auto
 have 4: {x ∈ set xs. P x} ≤ {x ∈ set xs. P x ∨ Q x}
    by auto
 show ?thesis by (simp add: 2 3 4)
qed

lemma set-filter-not:
 assumes  ∃ x ∈ set xs. P x
          ∃ x ∈ set xs. ¬ (P x)
```

```
 shows  set (filter (λx. ¬ (P x)) xs) = set xs − set (filter P xs)
using assms
 proof (induct xs)
 case (St x)
 then show ?case by auto
 next
 case (Cons x1a xs)
 then show ?case
  proof −
   have 1: set (filter (λx. ¬ P x) (x1a ⊙ xs)) =
         set (if (∃ y ∈ set xs. ¬ P y) then
               (if ¬ P x1a then x1a ⊙ filter (λ x. ¬ P x) xs else filter (λ x. ¬ P x) xs)
              else ⟨x1a⟩)

    by simp
   have 2: set (if (∃ y ∈ set xs. ¬ P y) then
               (if ¬ P x1a then x1a ⊙ filter (λ x. ¬ P x) xs else filter (λ x. ¬ P x) xs)
              else ⟨x1a⟩) =
         (if (∃ y ∈ set xs. ¬ P y) then
               (if ¬ P x1a then set (x1a ⊙ filter (λ x. ¬ P x) xs)
                           else set (filter (λ x. ¬ P x) xs))
              else set (⟨x1a⟩))
     by simp
   have 3: (if (∃ y ∈ set xs. ¬ P y) then
               (if ¬ P x1a then set (x1a ⊙ filter (λ x. ¬ P x) xs)
                            else set (filter (λ x. ¬ P x) xs))
              else set (⟨x1a⟩)) =
         (if (∃ y ∈ set xs. ¬ P y) then
               (if ¬ P x1a then set (⟨x1a⟩) ∪ set( filter (λ x. ¬ P x) xs)
                           else set (filter (λ x. ¬ P x) xs))
              else set (⟨x1a⟩))

      by simp
   have 4: set (filter P (x1a ⊙ xs))  =
         set (if (∃ y ∈ set xs.  P y) then
               (if  P x1a then x1a ⊙ filter P xs else filter P xs)
              else ⟨x1a⟩)

     by simp
   have 5: set (if (∃ y ∈ set xs.  P y) then
               (if  P x1a then x1a ⊙ filter P xs else filter P xs)
              else ⟨x1a⟩) =
          (if (∃ y ∈ set xs.  P y) then
               (if  P x1a then set (x1a ⊙ filter P xs) else set (filter P xs))
              else set ⟨x1a⟩)

     by simp
   have 6: (if (∃ y ∈ set xs.  P y) then
               (if  P x1a then set (x1a ⊙ filter P xs) else set (filter P xs))
              else set ⟨x1a⟩) =
```

(*if* $(\exists y \in$ *set xs*. $P\ y)$ *then*
(*if* P *x1a then set* $(\langle x1a\rangle) \cup$ *set*(*filter P xs*) *else set* (*filter P xs*))
else set $\langle x1a\rangle$)

by *simp*
have *7*: (*if* $(\exists y \in$ *set xs*. $\neg P\ y)$ *then*
(*if* $\neg P$ *x1a then set* $(\langle x1a\rangle) \cup$ *set*(*filter* $(\lambda\ x.\ \neg P\ x)$ *xs*)
else set (*filter* $(\lambda\ x.\ \neg P\ x)$ *xs*))
else set $(\langle x1a\rangle)) =$
set (*x1a* $\odot$ *xs*) $-$
(*if* $(\exists y \in$ *set xs*. $P\ y)$ *then*
(*if* P *x1a then set* $(\langle x1a\rangle) \cup$ *set*(*filter P xs*) *else set* (*filter P xs*))
else set $\langle x1a\rangle$)

using *Cons.prems* **by** *auto*
show *?thesis* **using** *1 3 4 6 7* **by** *presburger*
qed
qed

lemma *set-filter-cap*:
assumes $\exists\ x \in$ *set xs*. *f x*
$\exists\ x \in$ *set xs*. $\neg$ *f x*
shows *set* (*filter f xs*) $\cap$ *set* (*filter* $(\lambda\ x.\ \neg f\ x)$ *xs*) $= \{\}$
using *assms* **by** *auto*

lemma *filter-nth-or*:
assumes $\exists\ x \in$ *set xs*. $P\ x$
shows $\exists\ x \in$ *set* (*filter* $(\lambda x.\ P\ x \lor Q\ x)$ *xs*). $P\ x$
proof $-$
have *1*: $\exists\ i \leq$ *intlen*(*filter* $(\lambda x.\ P\ x \lor Q\ x)$ *xs*). P (*nth* (*filter* $(\lambda x.\ P\ x)$ *xs*) *i*)
using *assms* **by** (*metis* (*mono-tags*, *lifting*) *set-filter interval-intlen-gr-zero*
mem-Collect-eq nth-set)
obtain *i* **where** *2*: $i \leq$ *intlen*(*filter* $(\lambda x.\ P\ x \lor Q\ x)$ *xs*) $\land$ P (*nth* (*filter* $(\lambda x.\ P\ x)$ *xs*) *i*) $\land$
(*nth* (*filter* $(\lambda x.\ P\ x)$ *xs*) *i*) $\in$ *set* (*filter P xs*)
using *1*
by (*metis* (*mono-tags*, *lifting*) *set-filter assms interval-intlen-gr-zero*
mem-Collect-eq nth-set)
have *3*: *set* (*filter P xs*) $\leq$ *set* (*filter* $(\lambda x.\ P\ x \lor Q\ x)$ *xs*)
using *assms subset-filter*[*of xs P*] **by** *auto*
have *4*: (*nth* (*filter* $(\lambda x.\ P\ x)$ *xs*) *i*) $\in$ *set* (*filter P xs*)
using *2* **by** *auto*
have *5*: (*nth* (*filter* $(\lambda x.\ P\ x)$ *xs*) *i*) $\in$ *set* (*filter* $(\lambda x.\ P\ x \lor Q\ x)$ *xs*)
using *3 4* **by** *blast*
from *2 5* **show** *?thesis* **by** *blast*
qed

lemma *filter-intapp1*:
assumes $\forall x \in$ *set xs*. $\neg P\ x$
P x1a
P y

y ∈ *set ys*
shows *filter P* (*xs* ⊖ *ys*) = *filter P ys*
using *assms*
by (*induct xs*) *auto*

lemma *filter-intapp* [*simp*]:
assumes (∃ *x* ∈ *set* (*xs* ⊖ *ys*). *P x*)
(∃ *x* ∈ *set xs*. *P x*)
(∃ *x* ∈ *set ys*. *P x*)
shows *filter P* (*xs*⊖*ys*) = (*filter P xs*) ⊖ (*filter P ys*)
using *assms*
proof
(*induct xs arbitrary*: *ys*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof
(*cases* (∃ *x* ∈ *set xs*. *P x*))
case *True*
then show *?thesis* **using** *Cons.hyps Cons.prems*(*3*) **by** *auto*
next
case *False*
then show *?thesis*
proof *auto*
show ⋀*y*. ∀*x*∈*set xs*. ¬ *P x* ⟹
P x1a ⟹ *P y* ⟹ *y* ∈ *set ys* ⟹
filter P (*xs* ⊖ *ys*) = *filter P ys*
by (*simp add*: *filter-intapp1*)
show ∀*x*∈*set xs*. ¬ *P x* ⟹ *P x1a* ⟹ ∃*x*∈*set xs* ∪ *set ys*. *P x*
using *Cons.prems*(*3*) **by** *blast*
show ⋀*y*. ∀*x*∈*set xs*. ¬ *P x* ⟹
¬ *P x1a* ⟹ *P y* ⟹ *y* ∈ *set ys* ⟹
filter P (*xs* ⊖ *ys*) = *x1a* ⊙ *filter P ys*
using *Cons.prems*(*2*) **by** *auto*
show ∀*x*∈*set xs*. ¬ *P x* ⟹ ¬ *P x1a* ⟹ ∃*x*∈*set xs* ∪ *set ys*. *P x*
using *Cons.prems*(*2*) **by** *auto*
qed
qed
qed

lemma *filter-True*:
assumes ∀*x* ∈ *set xs*. *P x*
shows *filter P xs* = *xs*
using *assms*
by (*meson filter-id-conv length-filter-le nth-set*)

lemma *nfilter-map*:
assumes ∃ *x* ∈ *set* (*map f xs*). *P x*

shows *nfilter P* (*map f xs*) *n* = (*nfilter* (*P*∘*f*) *xs n*)
using *assms*
by (*induct xs arbitrary*: *n*) *auto*

lemma *filter-map*:
assumes ∃ *x* ∈ *set* (*map f xs*). *P x*
shows *filter P* (*map f xs*) = *map f* (*filter* (*P*∘*f*) *xs*)
using *assms*
by (*induct xs*) *auto*

lemma *length-nfilter-map*[*simp*]:
assumes ∃ *x* ∈ *set* (*map f xs*). *P x*
shows *intlen* (*nfilter P* (*map f xs*) *n*) = *intlen*(*nfilter* (*P* ∘ *f*) *xs n*)
using *assms* **by** (*simp add*:*nfilter-map*)

lemma *length-filter-map*[*simp*]:
assumes ∃ *x* ∈ *set* (*map f xs*). *P x*
shows *intlen* (*filter P* (*map f xs*)) = *intlen*(*filter* (*P* ∘ *f*) *xs*)
using *assms* **by** (*simp add*:*filter-map*)

lemma *nfilter-is-subset* [*simp*]:
assumes ∃ *x* ∈ *set xs*. *P x*
shows *set* (*nfilter P xs n*) ≤ {*n*+*k*|*k*. *k*≤*intlen xs*}
using *assms* **by** *auto*

lemma *filter-is-subset* [*simp*]:
assumes ∃ *x* ∈ *set xs*. *P x*
shows *set* (*filter P xs*) ≤ *set xs*
using *assms* **by** *auto*

lemma *length-nfilter-less*:
assumes ∃ *x* ∈ *set xs*. *P x*
x ∈ *set xs*
¬ *P x*
shows *intlen*(*nfilter P xs n*) < *intlen xs*
using *assms*
proof
(*induct xs arbitrary*: *n*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case* **using** *le-imp-less-Suc length-nfilter-le* **by** (*simp-all*, *blast*)
qed

lemma *length-filter-less*:
assumes ∃ *x* ∈ *set xs*. *P x*
x ∈ *set xs*
¬ *P x*

shows *intlen*(*filter P xs*) < *intlen xs*
using *assms*
proof
(*induct xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
using *length-filter-le le-imp-less-Suc* **by** (*simp-all*, *blast*)
qed

lemma *nfilter-set*:
assumes $\exists\ x \in set\ xs.\ P\ x$
shows $set\ (nfilter\ P\ xs\ n) = \{n+i|i.\ i \leq intlen\ xs \wedge P(nth\ xs\ i)\}$
using *assms* **by** *auto*

lemma *State-eq-filterD*:
assumes $\exists\ x \in set\ ys.\ P\ x$
$\langle x \rangle = filter\ P\ ys$
shows ($\exists$ *us vs* . ($ys = \langle x \rangle \vee$
$(ys = x \odot vs \wedge (\forall\ v \in set\ vs.\ \neg\ P\ v)) \vee$
$(ys = us \ominus \langle x \rangle \wedge (\forall\ u \in set\ us.\ \neg\ P\ u)) \vee$
$(ys = us \ominus (x \odot vs) \wedge (\forall\ u \in set\ us.\ \neg\ P\ u) \wedge (\forall\ v \in set\ vs.\ \neg\ P\ v))$
$) \wedge P\ x$
)
using *assms*
proof
(*induct ys*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a ys*)
then show *?case*
proof
(*cases P x1a*)
case *True*
then show *?thesis*
by (*metis Cons.prems*(*2*) *filter.simps*(*2*) *interval.distinct*(*1*) *interval.inject*(*1*))
next
case *False*
then show *?thesis*
proof
(*cases x = x1a*)
case *True*
then show *?thesis*
using *Cons.hyps Cons.prems*(*1*) *Cons.prems*(*2*) *False* **by** *auto*
next
case *False*

then show *?thesis*
proof −
have *1*: $\exists\ x \in set\ ys.\ \ P\ x$
using *Cons.prems*(*2*) *False interval.simps*(*15*) **by** *fastforce*
have *2*: $\neg\ P\ x1a$
using *1 Cons.prems*(*2*) **by** *auto*
have *3*: *P x*
using *1 2 Cons.hyps Cons.prems*(*2*) **by** *auto*
have *4*: $(\langle x\rangle = filter\ P\ (x1a \odot ys)) =$
$(if\ (\exists y{\in}set\ ys.\ P\ y)\ then$
$(if\ P\ x1a\ then\ \langle x\rangle = x1a \odot filter\ P\ ys\ else\ \langle x\rangle = filter\ P\ ys)$
$else\ \langle x\rangle = \langle x1a\rangle)$

using *2* **by** *auto*
have *5*: $\langle x\rangle = filter\ P\ ys$
by (*simp add*: *1 2 Cons.prems*(*2*))
have *6*: $(\exists\ \ us\ vs\ .$
$(ys = \langle x\rangle\ \vee$
$(ys = x \odot vs \wedge (\forall\ \ v \in set\ vs.\ \neg\ P\ v))\ \vee$
$(ys = us \ominus \langle x\rangle \wedge (\forall\ \ u \in set\ us.\ \neg\ P\ u))\ \vee$
$(ys = us \ominus (x \odot vs) \wedge (\forall\ \ u \in set\ us.\ \neg\ P\ u) \wedge (\forall\ \ v \in set\ vs.\ \neg\ P\ v))$
$) \wedge P\ x)$
using *1 Cons.hyps Cons.prems*(*2*) **by** *auto*
obtain *us vs* **where** *61*: $(ys = \langle x\rangle\ \vee$
$(ys = x \odot vs \wedge (\forall\ \ v \in set\ vs.\ \neg\ P\ v))\ \vee$
$(ys = us \ominus \langle x\rangle \wedge (\forall\ \ u \in set\ us.\ \neg\ P\ u))\ \vee$
$(ys = us \ominus (x \odot vs) \wedge (\forall\ \ u \in set\ us.\ \neg\ P\ u) \wedge (\forall\ \ v \in set\ vs.\ \neg\ P\ v))$
$) \wedge P\ x$
using *6* **by** *auto*
have *7*: $ys{=}\langle x\rangle \longrightarrow$
$(x1a \odot ys = \langle x\rangle\ \vee$
$(x1a \odot ys = x \odot vs \wedge (\forall v{\in}set\ vs.\ \neg\ P\ v))\ \vee$
$(x1a \odot ys = \langle x1a\rangle \ominus \langle x\rangle \wedge (\forall u{\in}set\ \langle x1a\rangle.\ \neg\ P\ u))\ \vee$
$(x1a \odot ys = \langle x1a\rangle \ominus x \odot vs \wedge (\forall u{\in}set\ \langle x1a\rangle.\ \neg\ P\ u) \wedge (\forall v{\in}set\ vs.\ \neg\ P\ v)))$
$\wedge\ P\ x$

using *1 Cons.prems*(*2*) **by** *auto*
have *8*: $(ys = x \odot vs \wedge (\forall\ \ v \in set\ vs.\ \neg\ P\ v)) \longrightarrow$
$(x1a \odot ys = \langle x\rangle\ \vee$
$x1a \odot ys = x \odot vs \wedge (\forall v{\in}set\ vs.\ \neg\ P\ v)\ \vee$
$x1a \odot ys = \langle x1a\rangle \ominus \langle x\rangle \wedge (\forall u{\in}set\ \langle x1a\rangle.\ \neg\ P\ u)\ \vee$
$x1a \odot ys = \langle x1a\rangle \ominus x \odot vs \wedge (\forall u{\in}set\ \langle x1a\rangle.\ \neg\ P\ u) \wedge (\forall v{\in}set\ vs.\ \neg\ P\ v))$
$\wedge\ P\ x$

using *1 Cons.prems*(*2*) **by** *fastforce*
have *9*: $(ys = us \ominus \langle x\rangle \wedge (\forall\ \ u \in set\ us.\ \neg\ P\ u)) \longrightarrow$
$(x1a \odot ys = \langle x\rangle\ \vee$
$x1a \odot ys = x \odot vs \wedge (\forall v{\in}set\ vs.\ \neg\ P\ v)\ \vee$
$x1a \odot ys = (x1a \odot us) \ominus \langle x\rangle \wedge (\forall u{\in}set\ (x1a \odot us).\ \neg\ P\ u)\ \vee$
$x1a \odot ys = (x1a \odot us) \ominus x \odot vs \wedge (\forall u{\in}set\ (x1a{\odot}us).\ \neg\ P\ u)\ \wedge$

```
              (∀ v∈set vs. ¬ P v))
            ∧ P x
        using 2 3 by auto
      have 10: (ys = us ⊖ (x ⊙ vs) ∧ (∀ u ∈ set us. ¬ P u) ∧ (∀ v ∈ set vs. ¬ P v)) ⟶
              (x1a ⊙ ys = ⟨x⟩ ∨
               x1a ⊙ ys = x ⊙ vs ∧ (∀ v∈set vs. ¬ P v) ∨
               x1a ⊙ ys = (x1a⊙us) ⊖ ⟨x⟩ ∧ (∀ u∈set (x1a⊙us). ¬ P u) ∨
               x1a ⊙ ys = (x1a ⊙us) ⊖ x ⊙ vs ∧ (∀ u∈set (x1a⊙us). ¬ P u) ∧
              (∀ v∈set vs. ¬ P v))
              ∧ P x
         using 2 3 by auto
       have 11: ( (ys = ⟨x⟩ ∨
           (ys = x ⊙ vs ∧ (∀ v ∈ set vs. ¬ P v)) ∨
           (ys = us ⊖ ⟨x⟩ ∧ (∀ u ∈ set us. ¬ P u)) ∨
           (ys = us ⊖ (x ⊙ vs) ∧ (∀ u ∈ set us. ¬ P u) ∧ (∀ v ∈ set vs. ¬ P v))
           ) ∧ P x) ⟶ (∃ us vs.
             (x1a ⊙ ys = ⟨x⟩ ∨
              x1a ⊙ ys = x ⊙ vs ∧ (∀ v∈set vs. ¬ P v) ∨
              x1a ⊙ ys = us ⊖ ⟨x⟩ ∧ (∀ u∈set us. ¬ P u) ∨
              x1a ⊙ ys = us ⊖ x ⊙ vs ∧ (∀ u∈set us. ¬ P u) ∧ (∀ v∈set vs. ¬ P v))
             ∧ P x
           )
      using 3
      using 10 7 8 9 False by blast
    show ?thesis using 11 3 61 by blast
   qed
  qed
 qed
qed


lemma filter-eq-StateD:
 assumes ∃ x ∈ set ys. P x
          filter P ys = ⟨x⟩
 shows (∃ us vs . (ys = ⟨x⟩ ∨
        (ys = x ⊙ vs ∧ (∀ v ∈ set vs. ¬ P v)) ∨
        (ys = us ⊖ ⟨x⟩ ∧ (∀ u ∈ set us. ¬ P u)) ∨
        (ys = us ⊖ (x ⊙ vs) ∧ (∀ u ∈ set us. ¬ P u) ∧ (∀ v ∈ set vs. ¬ P v))
       ) ∧ P x)
using assms State-eq-filterD[of ys P x ] by simp

lemma filter-eq-State-iff :
 assumes ∃ x ∈ set ys. P x
 shows (filter P ys = ⟨x⟩) =
       (∃ us vs . (ys = ⟨x⟩ ∨
        (ys = x ⊙ vs ∧ (∀ v ∈ set vs. ¬ P v)) ∨
        (ys = us ⊖ ⟨x⟩ ∧ (∀ u ∈ set us. ¬ P u)) ∨
        (ys = us ⊖ (x ⊙ vs) ∧ (∀ u ∈ set us. ¬ P u) ∧ (∀ v ∈ set vs. ¬ P v))
       ) ∧ P x)
proof −
```

have *1*: $(filter\ P\ ys = \langle x\rangle) \Longrightarrow$
$(\exists\ us\ vs\ .\ (ys = \langle x\rangle \vee$
$(ys = x \odot vs \wedge (\forall\ v \in set\ vs.\ \neg\ P\ v)) \vee$
$(ys = us \ominus \langle x\rangle \wedge (\forall\ u \in set\ us.\ \neg\ P\ u)) \vee$
$(ys = us \ominus (x \odot vs) \wedge (\forall\ u \in set\ us.\ \neg\ P\ u) \wedge (\forall\ v \in set\ vs.\ \neg\ P\ v))$
$) \wedge P\ x)$

using *assms* **by** (*rule State-eq-filterD*) *simp*
have *2*: $(\exists\ us\ vs\ .\ (ys = \langle x\rangle \vee$
$(ys = x \odot vs \wedge (\forall\ v \in set\ vs.\ \neg\ P\ v)) \vee$
$(ys = us \ominus \langle x\rangle \wedge (\forall\ u \in set\ us.\ \neg\ P\ u)) \vee$
$(ys = us \ominus (x \odot vs) \wedge (\forall\ u \in set\ us.\ \neg\ P\ u) \wedge (\forall\ v \in set\ vs.\ \neg\ P\ v))$
$) \wedge P\ x) \Longrightarrow (filter\ P\ ys = \langle x\rangle)$
by (*auto simp add*: *filter-intapp1*)
from *1 2* **show** *?thesis* **by** *metis*
qed

lemma *Cons-eq-filterD*:
assumes $\exists\ x \in set\ ys.\ P\ x$
$(x\odot\ xs = filter\ P\ ys)$
shows $(\exists\ us\ vs.\ (ys = x\odot\ vs \vee$
$ys = us \ominus (x \odot vs) \wedge (\forall\ u\ \in set\ us.\ \neg\ P\ u))$
$\wedge (\exists\ x \in set\ vs.\ P\ x)$
$\wedge\ P\ x \wedge xs = filter\ P\ vs)$
using *assms*
proof
(*induct ys*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a ys*)
then show *?case*
proof (*cases P x1a*)
case *True*
then show *?thesis*
by (*metis Cons.prems*(2) *filter.simps*(2) *interval.inject*(2) *interval.simps*(4))
next
case *False*
then show *?thesis*
proof (*cases* $x = x1a$)
case *True*
then show *?thesis*
using *Cons.hyps Cons.prems*(2) *False* **by** *force*
next
case *False*
then show *?thesis*
proof −
have *1*: $\exists\ x \in set\ ys.\ \ P\ x$
by (*metis Cons.prems*(2) *filter.simps*(2) *interval.simps*(4))
have *2*: $\neg\ P\ x1a$

```
          using 1 Cons.prems(2) False by auto
        have 4: (x⊙xs = filter P (x1a ⊙ ys)) =
              (if (∃y∈set ys. P y) then
                        (if P x1a then x⊙xs = x1a ⊙ filter P ys else x⊙xs =filter P ys)
                       else x⊙xs =⟨x1a⟩)
          by simp
        have 5: x⊙xs = filter P ys
          by (simp add: 1 2 Cons.prems(2))
        have 6: (∃ us vs. (ys = x⊙ vs  ∨
                  ys = us ⊖ (x ⊙ vs) ∧ (∀ u ∈ set us. ¬ P u))
                  ∧ (∃ x ∈ set vs. P x)
                  ∧ P x ∧ xs = filter P vs)
          by (simp add: 1 2 Cons.hyps Cons.prems(2))
        obtain us vs where 61: (ys = x⊙ vs  ∨
                  ys = us ⊖ (x ⊙ vs) ∧ (∀ u ∈ set us. ¬ P u))
                 ∧ (∃ x ∈ set vs. P x)
                 ∧ P x ∧ xs = filter P vs
           using 6 by auto
        have 62: P x ∧ (∃ x ∈ set vs. P x)
           using 61 by blast
        have 7: ys = x⊙ vs ∧ P x ∧ xs = filter P vs ⟶
             (x1a ⊙ ys = x ⊙ vs  ∨
             x1a ⊙ ys = ⟨x1a⟩ ⊖ x ⊙ vs ∧ (∀u∈set ⟨x1a⟩. ¬ P u))
              ∧ (∃ x ∈ set vs. P x)
            ∧ P x ∧ xs = filter P vs

           by (simp add: 2 62)
         have 8: ys = us ⊖ (x ⊙ vs) ∧ (∀ u ∈ set us. ¬ P u) ∧ P x ∧ xs = filter P vs ⟶
                (x1a ⊙ ys = x ⊙ vs  ∨
                 x1a ⊙ ys = (x1a ⊙us) ⊖ x ⊙ vs ∧ (∀u∈set (x1a ⊙us). ¬ P u))
                 ∧ (∃ x ∈ set vs. P x)
                 ∧ P x ∧ xs = filter P vs

           using 2 62 by auto
         have 9: (ys = x⊙ vs  ∨
                 ys = us ⊖ (x ⊙ vs) ∧ (∀ u ∈ set us. ¬ P u)) ∧ P x ∧ xs = filter P vs ⟶
            (∃us vs.
             (x1a ⊙ ys = x ⊙ vs  ∨
              x1a ⊙ ys = us ⊖ x ⊙ vs ∧ (∀u∈set us. ¬ P u))
             ∧ (∃ x ∈ set vs. P x)
            ∧ P x ∧ xs = filter P vs)

         using 7 8 by blast
        show ?thesis using 61 9 by blast
       qed
    qed
   qed
qed
```

lemma *filter-eq-ConsD*:
assumes $\exists\ x \in set\ ys.\ P\ x$
$(filter\ P\ ys = x \odot xs)$
shows $(\exists\ us\ vs.\ (ys = x \odot\ vs\ \vee$
$ys = us \ominus (x \odot vs) \wedge (\forall\ u\ \in set\ us.\ \neg\ P\ u))$
$\wedge (\exists\ x \in set\ vs.\ P\ x)$
$\wedge\ P\ x \wedge xs = filter\ P\ vs)$
using *assms Cons-eq-filterD*[*of ys*] **by** *simp*

lemma *filter-eq-Cons-iff*:
assumes $(\exists\ x \in set\ ys.\ P\ x)$
shows $(filter\ P\ ys = x \odot xs)\ =$
(
$(\exists\ us\ vs.\ (ys = x \odot\ vs\ \vee$
$ys = us \ominus (x \odot vs) \wedge (\forall\ u\ \in set\ us.\ \neg\ P\ u))$
$\wedge (\exists\ x \in set\ vs.\ P\ x)$
$\wedge\ P\ x \wedge xs = filter\ P\ vs))$
proof −
have *1*: $(filter\ P\ ys = x \odot xs) \Longrightarrow$
(
$(\exists\ us\ vs.\ (ys = x \odot\ vs\ \vee$
$ys = us \ominus (x \odot vs) \wedge (\forall\ u\ \in set\ us.\ \neg\ P\ u))$
$\wedge (\exists\ x \in set\ vs.\ P\ x)$
$\wedge\ P\ x \wedge xs = filter\ P\ vs))$

using *assms* **by** (*rule Cons-eq-filterD*) *simp*
have *2*: (
$(\exists\ us\ vs.\ (ys = x \odot\ vs\ \vee$
$ys = us \ominus (x \odot vs) \wedge (\forall\ u\ \in set\ us.\ \neg\ P\ u))$
$\wedge (\exists\ x \in set\ vs.\ P\ x)$
$\wedge\ P\ x \wedge xs = filter\ P\ vs)) \Longrightarrow$
$(filter\ P\ ys = x \odot xs)$

by (*auto simp add*: *filter-intapp1*)
from *1 2* **show** *?thesis* **by** *metis*
qed

lemma *Cons-eq-filter-iff*:
assumes $(\exists\ x \in set\ ys.\ P\ x)$
shows $(x \odot xs = filter\ P\ ys\ )\ =$
(
$(\exists\ us\ vs.\ (ys = x \odot\ vs\ \vee$
$ys = us \ominus (x \odot vs) \wedge (\forall\ u\ \in set\ us.\ \neg\ P\ u))$
$\wedge (\exists\ x \in set\ vs.\ P\ x)$
$\wedge\ P\ x \wedge xs = filter\ P\ vs))$
proof −
have *1*: $(x \odot xs = filter\ P\ ys\ ) = (filter\ P\ ys = x \odot xs)$
by *auto*

have *2*: (*filter P ys* = *x* ⊙ *xs*) =
(
(∃ *us vs*. (*ys* = *x*⊙ *vs* ∨
ys = *us* ⊖ (*x* ⊙ *vs*) ∧ (∀ *u* ∈ *set us*. ¬ *P u*))
∧ (∃ *x* ∈ *set vs*. *P x*)
∧ *P x* ∧ *xs* = *filter P vs*))

using *assms*
by (*simp add*: *filter-eq-Cons-iff*)
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *nfilter-cong*[*fundef-cong*]:
assumes *xs* = *ys*
(⋀*x*. *x* ∈ *set ys* ⟹ *P x* = *Q x*)
shows *nfilter P xs n* = *nfilter Q ys n*
using *assms* **by** (*induction xs arbitrary*: *ys n*) *auto*

lemma *filter-cong*[*fundef-cong*]:
assumes *xs* = *ys*
(⋀*x*. *x* ∈ *set ys* ⟹ *P x* = *Q x*)
shows *filter P xs* = *filter Q ys*
using *assms* **by** (*induct ys arbitrary*: *xs*) *auto*

lemma *remdups-filter*:
assumes ∃ *x* ∈ *set xs* . *P x*
shows *remdups*(*filter P xs*) = *filter P* (*remdups xs*)
using *assms*
by (*induct xs*) *auto*

lemma *distinct-map-filter*:
assumes ∃ *x* ∈ *set xs* . *P x*
distinct (*map f xs*)
shows *distinct* (*map f* (*filter P xs*))
using *assms* **by** (*induct xs*) *auto*

lemma *distinct-nfilter* [*simp*]:
distinct (*nfilter P xs n*)
by (*induction xs arbitrary*: *n*) *auto*

lemma *distinct-filter* [*simp*]:
assumes *distinct xs*
shows *distinct* (*filter P xs*)
using *assms* **by** (*induct xs*) *auto*

lemma *distinct-length-filter*:
assumes ∃ *x* ∈ *set xs* . *P x*
distinct xs
shows *intlen* (*filter P xs*) +*1* = *card* ({*x*. *P x*} *Int set xs*)

using *assms* **by** (*induct xs*) *auto*

lemma *filter-mono-osfx*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
osfx xs ys
shows *osfx* (*filter P xs*) (*filter P ys*)
using *assms*
proof (*auto simp*: *osfx-def*)
fix $x :: 'a$ **and** $zs :: 'a\ interval$
assume *a1*: *P x*
assume *a2*: $x \in set\ xs$
assume *a3*: $filter\ P\ (zs \ominus xs) \neq filter\ P\ xs$
assume *a4*: $ys = zs \ominus xs$
have *f5*: $\forall\, i\ p\ a\ aa\ ia.$
$((\exists\, a.\ (a::'a) \in set\ i \wedge p\ a) \vee \neg\ p\ a \vee \neg\ p\ aa \vee aa \notin set\ ia) \vee$
$filter\ p\ (i \ominus ia) = filter\ p\ ia$
by (*metis* (*no-types*) *filter-intapp1*)
obtain $aa :: ('a \Rightarrow bool) \Rightarrow 'a\ interval \Rightarrow 'a$ **where**
$\forall\, x3\ x4.\ (\exists\, v5.\ v5 \in set\ x4 \wedge x3\ v5) = (aa\ x3\ x4 \in set\ x4 \wedge x3\ (aa\ x3\ x4))$
by *moura*
then have $\forall\, i\ p\ a\ ab\ ia.\ aa\ p\ i \in set\ i \wedge p\ (aa\ p\ i) \vee \neg\ p\ a \vee \neg\ p\ ab \vee$
$ab \notin set\ ia \vee filter\ p\ (i \ominus ia) = filter\ p\ ia$
using *f5* **by** *presburger*
then have *f6*: $\exists\, a.\ a \in set\ zs \wedge P\ a$
using *a3 a2 a1* **by** *blast*
have *osfx xs* ($zs \ominus xs$)
using *a4 assms*(*2*) **by** *blast*
then show $\exists\, i.\ filter\ P\ (zs \ominus xs) = i \ominus filter\ P\ xs$
using *f6 a2 a1* **by** (*meson filter-intapp set-mono-osfx subsetD*)
qed

lemma *idx-nfilter-mono*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
$na < intlen\ (nfilter\ P\ xs\ n)$
shows $nth\ (nfilter\ P\ xs\ n)\ na < nth\ (nfilter\ P\ xs\ n)\ (Suc\ na)$
using *assms*
proof (*induct xs arbitrary*: *n na*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases* $\exists\ x \in set\ xs\ .\ P\ x$)
show ($\bigwedge na\ n.$
$\exists\, a{\in}set\ xs.\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ na < nth\ (nfilter\ P\ xs\ n)\ (Suc\ na)) \Longrightarrow$
$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$

$na < intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ na < nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ (Suc\ na)$
proof (*cases na*)
show $(\bigwedge na\ n.$
$\exists a{\in}set\ xs.\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ na < nth\ (nfilter\ P\ xs\ n)\ (Suc\ na)) \Longrightarrow$
$\exists a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$
$na = 0 \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ na < nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ (Suc\ na)$
proof (*cases P x1a*)
show $(\bigwedge na\ n.$
$\exists a{\in}set\ xs.\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ na < nth\ (nfilter\ P\ xs\ n)\ (Suc\ na)) \Longrightarrow$
$\exists a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$
$na = 0 \Longrightarrow$
$P\ x1a \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ na < nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ (Suc\ na)$
by (*simp add*: *Suc-le-lessD nfilter-lower-bound*)
show $(\bigwedge na\ n.$
$\exists a{\in}set\ xs.\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ na < nth\ (nfilter\ P\ xs\ n)\ (Suc\ na)) \Longrightarrow$
$\exists a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$
$na = 0 \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ na < nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ (Suc\ na)$
by *simp*
qed
show $\bigwedge nat.$
$(\bigwedge na\ n.$
$\exists a{\in}set\ xs.\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ na < nth\ (nfilter\ P\ xs\ n)\ (Suc\ na)) \Longrightarrow$
$\exists a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$
$na = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ na < nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ (Suc\ na)$
by *auto*
qed
show $(\bigwedge na\ n.$

$\exists\, a\in set\ xs.\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ na < nth\ (nfilter\ P\ xs\ n)\ (Suc\ na)) \Longrightarrow$
$\exists\, a\in set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$na < intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\neg\ (\exists\, x\in set\ xs.\ P\ x) \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ na < nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ (Suc\ na)$
by *auto*
qed
qed

lemma *idx-nfilter*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
shows $index\text{-}sequence\ (intfirst((nfilter\ P\ xs\ n)))\ (nfilter\ P\ xs\ n)$
using *assms* **by** (*simp add*: *index-sequence-def idx-nfilter-mono*)

lemma *idx-nfilter-expand*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
shows $\forall\, na<intlen\ (nfilter\ P\ xs\ n).\ nth\ (nfilter\ P\ xs\ n)\ na < nth\ (nfilter\ P\ xs\ n)\ (Suc\ na)$
using *assms idx-nfilter* **by** (*simp add*: *index-sequence-def*)

lemma *idx-nfilter-gr-eq*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
$k \le j$
$j \le intlen(nfilter\ P\ xs\ n)$
shows $nth\ (nfilter\ P\ xs\ n)\ k \le\ nth\ (nfilter\ P\ xs\ n)\ j$
using *assms* **by** (*meson idx-nfilter interval-idx-less-eq*)

lemma *idx-nfilter-gr*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
shows $(\forall\ j\ .\ \ k< j \wedge j \le intlen(nfilter\ P\ xs\ n)\longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ k <\ nth\ (nfilter\ P\ xs\ n)\ j)$
using *assms*
by (*meson Suc-leI dual-order.strict-trans1 idx-nfilter-expand idx-nfilter-gr-eq*)

lemma *idx-nfilter-less-eq*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
$k\le\ intlen(nfilter\ P\ xs\ n)$
shows $(\forall\ j \le k.\ \ nth\ (nfilter\ P\ xs\ n)\ j \le\ nth\ (nfilter\ P\ xs\ n)\ k)$
using *assms* **by** (*simp add*: *idx-nfilter-gr-eq*)

lemma *idx-nfilter-less*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
$k\le\ intlen(nfilter\ P\ xs\ n)$
shows $(\forall\ j < k.\ \ nth\ (nfilter\ P\ xs\ n)\ j <\ nth\ (nfilter\ P\ xs\ n)\ k)$
using *assms*

by (*simp add*: *idx-nfilter-gr*)

lemma *nfilter-prefix-set-0*:
assumes $k \leq$ *intlen xs*
$\exists\ x \in$ *set* (*prefix k xs*) . *P x*
shows *intlen* (*nfilter P* (*prefix k xs*) *n*) $\leq$ *intlen* (*nfilter P xs n*)
using *assms*
proof (*induct xs arbitrary*: *n k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases k*)
show ($\bigwedge k\ n.$
$k \leq$ *intlen xs* $\Longrightarrow$
$\exists a \in$*set* (*prefix k xs*). *P a* $\Longrightarrow$
intlen (*nfilter P* (*prefix k xs*) *n*) $\leq$ *intlen* (*nfilter P xs n*)) $\Longrightarrow$
$k \leq$ *intlen* (*x1a* $\odot$ *xs*) $\Longrightarrow$
$\exists a \in$*set* (*prefix k* (*x1a* $\odot$ *xs*)). *P a* $\Longrightarrow$
$k = 0 \Longrightarrow$
intlen (*nfilter P* (*prefix k* (*x1a* $\odot$ *xs*)) *n*) $\leq$ *intlen* (*nfilter P* (*x1a* $\odot$ *xs*) *n*)
by *simp*
show $\bigwedge$*nat*.
($\bigwedge k\ n.$
$k \leq$ *intlen xs* $\Longrightarrow$
$\exists a \in$*set* (*prefix k xs*). *P a* $\Longrightarrow$
intlen (*nfilter P* (*prefix k xs*) *n*) $\leq$ *intlen* (*nfilter P xs n*)) $\Longrightarrow$
$k \leq$ *intlen* (*x1a* $\odot$ *xs*) $\Longrightarrow$
$\exists a \in$*set* (*prefix k* (*x1a* $\odot$ *xs*)). *P a* $\Longrightarrow$
$k =$ *Suc nat* $\Longrightarrow$
intlen (*nfilter P* (*prefix k* (*x1a* $\odot$ *xs*)) *n*) $\leq$ *intlen* (*nfilter P* (*x1a* $\odot$ *xs*) *n*)
using *prefix-subset* **by** *simp blast*
qed
qed

lemma *nfilter-subset*:
assumes $\exists\ x \in$ *set* (*prefix k xs*) . *P x*
$k \leq$ *intlen xs*
shows *set* (*nfilter P* (*prefix k xs*) *n*) $\leq$ *set* (*nfilter P xs n*)
using *assms*
proof (*induct xs arbitrary*:*k n*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a xs*)
then show *?case*

proof (*cases k*)
show ($\bigwedge$*k n*.
$\exists\, a{\in}set$ (*prefix k xs*). *P a* $\Longrightarrow$
$k \leq$ *intlen xs* $\Longrightarrow$
set (*nfilter P* (*prefix k xs*) *n*) $\subseteq$ *set* (*nfilter P xs n*)) $\Longrightarrow$
$\exists\, a{\in}set$ (*prefix k* (*x1a* $\odot$ *xs*)). *P a* $\Longrightarrow$
$k \leq$ *intlen* (*x1a* $\odot$ *xs*) $\Longrightarrow$
$k = 0 \Longrightarrow$
set (*nfilter P* (*prefix k* (*x1a* $\odot$ *xs*)) *n*) $\subseteq$ *set* (*nfilter P* (*x1a* $\odot$ *xs*) *n*)
by *simp*
show $\bigwedge$*nat*.
($\bigwedge$*k n*.
$\exists\, a{\in}set$ (*prefix k xs*). *P a* $\Longrightarrow$
$k \leq$ *intlen xs* $\Longrightarrow$
set (*nfilter P* (*prefix k xs*) *n*) $\subseteq$ *set* (*nfilter P xs n*)) $\Longrightarrow$
$\exists\, a{\in}set$ (*prefix k* (*x1a* $\odot$ *xs*)). *P a* $\Longrightarrow$
$k \leq$ *intlen* (*x1a* $\odot$ *xs*) $\Longrightarrow$
$k =$ *Suc nat* $\Longrightarrow$
set (*nfilter P* (*prefix k* (*x1a* $\odot$ *xs*)) *n*) $\subseteq$ *set* (*nfilter P* (*x1a* $\odot$ *xs*) *n*)
by (*auto simp add*: *nth-set*)
qed
qed

lemma *nfilter-prefix-set*:
assumes $\exists\ x \in$ *set* (*prefix k xs*) . *P x*
$k\leq$ *intlen xs*
shows *set* (*nfilter P* (*prefix k xs*) *n*) $= \{n{+}i|i.\ i{\leq}\ k \wedge P(nth\ xs\ i)\}$
using *assms*
by *auto*

lemma *nfilter-suffix-set*:
assumes $\exists\ x \in$ *set* (*suffix k xs*) . *P x*
$k\leq$ *intlen xs*
shows *set* (*nfilter P* (*suffix k xs*) *n*) $= \{n{+}i|i.\ i \leq intlen\ xs{-}k \wedge P(nth\ xs\ (k{+}i))\}$
using *assms* **by** *auto*

lemma *nfilter-suffix-set-a*:
assumes $\exists\ x \in$ *set* (*suffix k xs*) . *P x*
$k\leq$ *intlen xs*
shows *set* (*nfilter P* (*suffix k xs*) *n*) $= \{n{+}(j{-}k)|j.\ k{\leq}j \wedge\ j \leq intlen\ xs \wedge P(nth\ xs\ j)\}$
using *assms Nat.le-diff-conv2* **by** *auto fastforce*

lemma *nfilter-suffix-set-b*:
assumes $\exists\ x \in$ *set* (*suffix k xs*) . *P x*
$k\leq$ *intlen xs*
shows *set* (*nfilter P* (*suffix k xs*) *n*) $= \{n{+}i|i.\ i \leq intlen\ xs{-}k \wedge P(nth\ xs\ (i{+}k))\}$
using *assms nfilter-suffix-set* **by** (*auto simp add*: *add.commute*)

lemma *nfilter-card*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
shows $intlen\ (nfilter\ P\ xs\ n)\ +1 = card\ (set\ (nfilter\ P\ xs\ n))$
using *assms*
by (*induct xs arbitrary*: *n*) *auto*

lemma *nfilter-prefix-set-1*:
assumes $k \leq intlen\ xs$
$\exists\ x \in set\ (prefix\ k\ xs)\ .\ P\ x$
shows $set\ (prefix\ (intlen(nfilter\ P\ (prefix\ k\ xs)\ n))\ (nfilter\ P\ xs\ n)) =$
$\{(nth\ (nfilter\ P\ xs\ n)\ i) \mid i.\ i \leq intlen(nfilter\ P\ (prefix\ k\ xs)\ n)\}$
using *prefix-set*[*of* $(intlen(nfilter\ P\ (prefix\ k\ xs)\ n))\ (nfilter\ P\ xs\ n)$]
by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-prefix-set-0*)

lemma *nfilter-prefix-subset*:
assumes $\exists\ x \in set\ xs\ .\ P\ x$
$k \leq intlen\ (nfilter\ P\ xs\ n)$
shows $set\ (prefix\ k\ (nfilter\ P\ xs\ n)) \leq set\ (nfilter\ P\ xs\ n)$
using *assms*
using *prefix-subset* **by** *blast*

lemma *intlen-filter-conv-card*:
assumes $\exists\ x \in set\ xs.\ P\ x$
shows $intlen\ (filter\ P\ xs) + 1 = card\ \{i.\ i \leq intlen\ xs \wedge P\ (nth\ xs\ i)\}$
proof −
have *1*: $intlen\ (filter\ P\ xs) = intlen(nfilter\ P\ xs\ 0)$
by (*simp add*: *assms nfilter-intlen*)
have *2*: $intlen\ (nfilter\ P\ xs\ 0)\ +1 = card\ (set\ (nfilter\ P\ xs\ 0))$
by (*meson assms nfilter-card*)
have *3*: $set\ (nfilter\ P\ xs\ 0) = \{i|i.\ i \leq intlen\ xs \wedge P(nth\ xs\ i)\}$
using *assms* **by** *auto*
show *?thesis*
using *1 2 3* **by** *auto*
qed

lemma *nfilter-all*:
assumes $\exists\ x \in set\ xs.\ P\ x$
shows $((nfilter\ P\ xs\ n) = [n..\leq n+intlen\ xs]) = (\forall\ x \in set\ xs.\ P\ x)$
using *assms*
proof (*induction xs arbitrary*: *n*)
case (*St x*)
then show *?case* **by** (*simp add*: *upt-same*)
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases* $\exists\ x \in set\ xs.\ P\ x$)
show ($\bigwedge n.$
$\exists\ a{\in}set\ xs.\ P\ a \Longrightarrow$
$(nfilter\ P\ xs\ n = [n..\leq n + intlen\ xs]) = (\forall\ a{\in}set\ xs.\ P\ a)) \Longrightarrow$

$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$\exists\, x{\in}set\ xs.\ P\ x \Longrightarrow$
$(nfilter\ P\ (x1a \odot xs)\ n = [n..{\le}n + intlen\ (x1a \odot xs)]) = (\forall\, a{\in}set\ (x1a \odot xs).\ P\ a)$

proof *auto*
show $\bigwedge x\ xa.$
$(\bigwedge n.\ (nfilter\ P\ xs\ n = [n..{\le}n + intlen\ xs]) = (\forall\, x{\in}set\ xs.\ P\ x)) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$n \odot nfilter\ P\ xs\ (Suc\ n) = [n..{\le}n + intlen\ xs] \ominus \langle Suc\ (n + intlen\ xs)\rangle \Longrightarrow$
$xa \in set\ xs \Longrightarrow$
$P\ xa$
by (*metis add.right-neutral add-diff-cancel-left′ add-right-imp-eq interval-intlen-cons interval-intlen-intapp intlen.simps*(*1*) *length-nfilter-less nat-less-le upt-length*)
show $\bigwedge x.$
$(\bigwedge n.\ nfilter\ P\ xs\ n = [n..{\le}n + intlen\ xs]) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x1a \Longrightarrow$
$\forall\, x{\in}set\ xs.\ P\ x \Longrightarrow$
$Suc\ 0 \le intlen\ xs \Longrightarrow$
$n \odot [Suc\ n..{\le}n + intlen\ xs] \ominus \langle Suc\ (n + intlen\ xs)\rangle =$
$[n..{\le}n + intlen\ xs] \ominus \langle Suc\ (n + intlen\ xs)\rangle$
using *upt-rec* **by** *auto*
show $\bigwedge x.$
$(\bigwedge n.\ nfilter\ P\ xs\ n = [n..{\le}n + intlen\ xs]) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x1a \Longrightarrow$
$\forall\, x{\in}set\ xs.\ P\ x \Longrightarrow$
$\neg\ Suc\ 0 \le intlen\ xs \Longrightarrow$
$\langle n,\ Suc\ (n + intlen\ xs)\rangle = [n..{\le}n + intlen\ xs] \ominus \langle Suc\ (n + intlen\ xs)\rangle$
by (*simp add*: *upt-rec*)
show $\bigwedge x.$
$(\bigwedge n.\ (nfilter\ P\ xs\ n = [n..{\le}n + intlen\ xs]) = (\forall\, x{\in}set\ xs.\ P\ x)) \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$nfilter\ P\ xs\ (Suc\ n) = [n..{\le}n + intlen\ xs] \ominus \langle Suc\ (n + intlen\ xs)\rangle \Longrightarrow False$
by (*metis Suc-n-not-le-n interval-intfirst-intapp2 interval-intlen-gr-zero interval-nth-zero-intfirst le-add1 nfilter-lower-bound upt-intfirst*)
qed
show $(\bigwedge n.\ \exists\, a{\in}set\ xs.\ P\ a \Longrightarrow (nfilter\ P\ xs\ n = [n..{\le}n + intlen\ xs]) = (\forall\, a{\in}set\ xs.\ P\ a)) \Longrightarrow$
$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$\neg\ (\exists\, x{\in}set\ xs.\ P\ x) \Longrightarrow$
$(nfilter\ P\ (x1a \odot xs)\ n = [n..{\le}n + intlen\ (x1a \odot xs)]) = (\forall\, a{\in}set\ (x1a \odot xs).\ P\ a)$
proof *auto*
show $\bigwedge x.\ P\ x1a \Longrightarrow$
$\forall\, x{\in}set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle n\rangle = [n..{\le}n + intlen\ xs] \ominus \langle Suc\ (n + intlen\ xs)\rangle \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$False$

by (*metis interval-intapp-not-state*)
show *P x1a* $\Longrightarrow$
set xs = {} $\Longrightarrow$
$\langle n\rangle$ = [*n*..$\leq$*n* + *intlen xs*] $\ominus$ $\langle$*Suc* (*n* + *intlen xs*)$\rangle$
using *interval-set-nonempty* **by** *blast*
qed
qed
qed

lemma *filter-nfilter-prefix-intlen-0*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs n*) *k*) −*n*))
k $\leq$ *intlen* (*filter P xs*)
shows ($\exists$ *x* $\in$ *set xs*. *P x*)
using *assms*
by (*induction xs arbitrary*: *k*) *auto*

lemma *nfilter-intlen-n-zero*:
assumes ($\exists$ *x* $\in$ *set xs*. *P x*)
shows *intlen* (*nfilter P xs n*) = *intlen* (*nfilter P xs 0*)
using *assms*
proof
(*induction xs arbitrary*: *n*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases n*)
show ($\bigwedge$*n*. $\exists$ *a*$\in$*set xs*. *P a* $\Longrightarrow$ *intlen* (*nfilter P xs n*) = *intlen* (*nfilter P xs 0*)) $\Longrightarrow$
$\exists$ *a*$\in$*set* (*x1a* $\odot$ *xs*). *P a* $\Longrightarrow$
n = *0* $\Longrightarrow$
intlen (*nfilter P* (*x1a* $\odot$ *xs*) *n*) = *intlen* (*nfilter P* (*x1a* $\odot$ *xs*) *0*)
by *blast*
show $\bigwedge$*nat*.
($\bigwedge$*n*. $\exists$ *a*$\in$*set xs*. *P a* $\Longrightarrow$ *intlen* (*nfilter P xs n*) = *intlen* (*nfilter P xs 0*)) $\Longrightarrow$
$\exists$ *a*$\in$*set* (*x1a* $\odot$ *xs*). *P a* $\Longrightarrow$
n = *Suc nat* $\Longrightarrow$
intlen (*nfilter P* (*x1a* $\odot$ *xs*) *n*) = *intlen* (*nfilter P* (*x1a* $\odot$ *xs*) *0*)
by (*simp add*: *nfilter-intlen*)
qed
qed

lemma *nfilter-nth-n-zero-a*:
assumes ($\exists$ *x* $\in$ *set xs*. *P x*)
k $\leq$ *intlen* (*nfilter P xs n*)
shows *n* $\leq$ (*nth* (*nfilter P xs n*) *k*)
using *assms* **by** (*simp add*: *nfilter-lower-bound*)

lemma *nfilter-nth-n-zero*:

assumes $(\exists\ x \in set\ xs.\ P\ x)$
$k \leq intlen\ (nfilter\ P\ xs\ n)$
shows $(nth\ (nfilter\ P\ xs\ n)\ k) - n\ = nth\ (nfilter\ P\ xs\ 0)\ k$
using *assms*
proof
$(induction\ xs\ arbitrary\colon n\ k)$
case $(St\ x)$
then show *?case* **by** *simp*
next
case $(Cons\ x1a\ xs)$
then show *?case*
proof $(cases\ (\exists\ x \in set\ xs.\ P\ x))$
show $(\bigwedge k\ n.$
$\exists a{\in}set\ xs.\ P\ a \Longrightarrow$
$k \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ k - n = nth\ (nfilter\ P\ xs\ 0)\ k) \Longrightarrow$
$\exists a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$k \leq intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ k - n = nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ k$
proof $(cases\ P\ x1a)$
show $(\bigwedge k\ n.$
$\exists a{\in}set\ xs.\ P\ a \Longrightarrow$
$k \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ k - n = nth\ (nfilter\ P\ xs\ 0)\ k) \Longrightarrow$
$\exists a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$k \leq intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ k - n = nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ k$
proof $(cases\ k)$
show $(\bigwedge k\ n.$
$\exists a{\in}set\ xs.\ P\ a \Longrightarrow$
$k \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ k - n = nth\ (nfilter\ P\ xs\ 0)\ k) \Longrightarrow$
$\exists a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$k \leq intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$k = 0 \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ n)\ k - n = nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ k$
by *auto*
show $\bigwedge nat.$
$(\bigwedge k\ n.$
$\exists a{\in}set\ xs.\ P\ a \Longrightarrow$
$k \leq intlen\ (nfilter\ P\ xs\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ n)\ k - n = nth\ (nfilter\ P\ xs\ 0)\ k) \Longrightarrow$
$\exists a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$k \leq intlen\ (nfilter\ P\ (x1a \odot xs)\ n) \Longrightarrow$
$\exists x{\in}set\ xs.\ P\ x \Longrightarrow$

```
      P x1a ⟹
      k = Suc nat ⟹
      nth (nfilter P (x1a ⊙ xs) n) k − n = nth (nfilter P (x1a ⊙ xs) 0) k
  proof auto
    fix nat
    fix x
    show (⋀k n.
            k ≤ intlen (nfilter P xs n) ⟹
            nth (nfilter P xs n) k − n = nth (nfilter P xs 0) k) ⟹
          nat ≤ intlen (nfilter P xs (Suc n)) ⟹
          P x1a ⟹
          k = Suc nat ⟹
          x ∈ set xs ⟹
          P x ⟹
          nth (nfilter P xs (Suc n)) nat − n = nth (nfilter P xs (Suc 0)) nat
     proof −
      assume a0: (⋀k n.
                   k ≤ intlen (nfilter P xs n) ⟹
                   nth (nfilter P xs n) k − n = nth (nfilter P xs 0) k)
      assume a1:  nat ≤ intlen (nfilter P xs (Suc n))
      assume a2: P x1a
      assume a3:  k = Suc nat
      assume a4: x ∈ set xs
      assume a5: P x
      show nth (nfilter P xs (Suc n)) nat − n = nth (nfilter P xs (Suc 0)) nat
        proof −
         have 1: nth (nfilter P xs (Suc n)) nat − (Suc n) = nth (nfilter P xs 0) nat
               using a0 a1 by blast
         have 2: nth (nfilter P xs (Suc 0)) nat − (Suc 0) = nth (nfilter P xs 0) nat
            by (metis a0 a1 a4 a5 nfilter-intlen-n-zero)
         show ?thesis
         by (metis 1 2 One-nat-def Suc-diff-le a1 a4 a5 diff-Suc-Suc
             nfilter-intlen-n-zero nfilter-lower-bound
             ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc)
        qed
      qed
     qed
    qed
show (⋀k n.
       ∃a∈set xs. P a ⟹
       k ≤ intlen (nfilter P xs n) ⟹
       nth (nfilter P xs n) k − n = nth (nfilter P xs 0) k) ⟹
     ∃a∈set (x1a ⊙ xs). P a ⟹
     k ≤ intlen (nfilter P (x1a ⊙ xs) n) ⟹
     ∃x∈set xs. P x ⟹
     ¬ P x1a ⟹
     nth (nfilter P (x1a ⊙ xs) n) k − n = nth (nfilter P (x1a ⊙ xs) 0) k
 proof auto
  fix x
  show (⋀k n.
```

k ≤ *intlen* (*nfilter P xs n*) ⟹
nth (*nfilter P xs n*) *k* − *n* = *nth* (*nfilter P xs 0*) *k*) ⟹
k ≤ *intlen* (*nfilter P xs* (*Suc n*)) ⟹
¬ *P x1a* ⟹
x ∈ *set xs* ⟹
P x ⟹
nth (*nfilter P xs* (*Suc n*)) *k* − *n* = *nth* (*nfilter P xs* (*Suc 0*)) *k*
proof −
assume *b0*: (⋀*k n*.
k ≤ *intlen* (*nfilter P xs n*) ⟹
nth (*nfilter P xs n*) *k* − *n* = *nth* (*nfilter P xs 0*) *k*)
assume *b1*: *k* ≤ *intlen* (*nfilter P xs* (*Suc n*))
assume *b2*: ¬ *P x1a*
assume *b3*: *x* ∈ *set xs*
assume *b4*: *P x*
show *nth* (*nfilter P xs* (*Suc n*)) *k* − *n* = *nth* (*nfilter P xs* (*Suc 0*)) *k*
proof −
have *3*: *nth* (*nfilter P xs* (*Suc n*)) *k* − (*Suc n*) = *nth* (*nfilter P xs 0*) *k*
using *b0 b1* **by** *blast*
have *4*: *nth* (*nfilter P xs* (*Suc 0*)) *k* − (*Suc 0*) = *nth* (*nfilter P xs 0*) *k*
by (*metis b0 b1 b3 b4 nfilter-intlen-n-zero*)
show *?thesis*
by (*metis 3 4 One-nat-def Suc-diff-le b1 b3 b4 diff-Suc-Suc nfilter-intlen-n-zero nfilter-lower-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
qed
qed
qed
qed
show (⋀*k n*.
∃ *a*∈*set xs*. *P a* ⟹
k ≤ *intlen* (*nfilter P xs n*) ⟹
nth (*nfilter P xs n*) *k* − *n* = *nth* (*nfilter P xs 0*) *k*) ⟹
∃ *a*∈*set* (*x1a* ⊙ *xs*). *P a* ⟹
k ≤ *intlen* (*nfilter P* (*x1a* ⊙ *xs*) *n*) ⟹
¬ (∃ *x*∈*set xs*. *P x*) ⟹
nth (*nfilter P* (*x1a* ⊙ *xs*) *n*) *k* − *n* = *nth* (*nfilter P* (*x1a* ⊙ *xs*) *0*) *k*
by *auto*
qed
qed

lemma *nfilter-n-zero*:
assumes (∃ *x* ∈ *set xs*. *P x*)
shows (*nfilter P xs n*) = *map* (λ*i*. *i*+*n*) (*nfilter P xs 0*)
using *assms*
proof −
have *1*: *intlen* (*nfilter P xs n*) = *intlen* (*map* (λ*i*. *i*+*n*) (*nfilter P xs 0*))
using *assms nfilter-intlen-n-zero* **by** *fastforce*

have *2*: $\bigwedge$*k*. *k*≤ *intlen* (*nfilter P xs n*) ⟶
nth (*nfilter P xs n*) *k* = *nth* (*map* (λ*i*. *i*+*n*) (*nfilter P xs 0*)) *k*
using *assms nfilter-nth-n-zero*[*of xs P - n*]
by (*metis interval-nth-map le-add-diff-inverse2 nfilter-lower-bound*)
show *?thesis* **by** (*simp add*: *1 2 interval-eq-nth-eq*)
qed

lemma *nfilter-n-zero-a*:
assumes (∃ *x* ∈ *set xs*. *P x*)
shows (*nfilter P xs 0*) = *map* (λ*i*. *i*−*n*) (*nfilter P xs n*)
proof −
have *1*: *intlen*(*nfilter P xs 0*) = *intlen*(*map* (λ*i*. *i*−*n*) (*nfilter P xs n*))
by (*metis assms interval-intlen-map nfilter-intlen-n-zero*)
have *2*: $\bigwedge$*k*. *k*≤ *intlen* (*nfilter P xs 0*) ⟶
nth (*nfilter P xs 0*) *k* = *nth* (*map* (λ*i*. *i*−*n*) (*nfilter P xs n*)) *k*
using *assms*
by (*simp add*: *1 interval-nth-map nfilter-nth-n-zero*)
show *?thesis*
using *1 2 interval-eq-nth-eq* **by** *blast*
qed

lemma *nfilter-count*:
assumes (∃ *x* ∈ *set xs*. *P x*)
shows *card* {(*nth* (*nfilter P xs n*) *k*)| *k*. *k* ≤ *intlen* (*nfilter P xs n*)} =
intlen (*nfilter P xs n*) +*1*
using *assms nfilter-card*[*of xs P n*] *set-nfilter*[*of xs P n*]
interval-nth-and-set **by** (*simp add*: *set-nth*)

lemma *nfilter-holds*:
assumes (∃ *x* ∈ *set xs*. *P x*)
shows (∀ *x* ∈ *set* (*nfilter P xs n*). *P* (*nth xs* (*x*−*n*)))
using *assms* **by** *auto*

lemma *nfilter-holds-not*:
assumes (∃ *x* ∈ *set xs*. *P x*)
shows (∀ *x* ∈ ({*i*+*n*| *i*. *i*≤ *intlen xs*} − (*set* (*nfilter P xs n*))). ¬ *P* (*nth xs* (*x*−*n*)))
using *assms* **by** *auto*

lemma *nfilter-holds-a*:
assumes (∃ *x* ∈ *set xs*. *P x*)
shows (∀ *i*≤*intlen xs*. (*i*+*n*) ∈ *set* (*nfilter P xs n*) ⟶ *P* (*nth xs i*))
using *assms* **by** *auto*

lemma *nfilter-holds-not-a*:
assumes (∃ *x* ∈ *set xs*. *P x*)
shows (∀ *i*≤ *intlen xs*. *P* (*nth xs i*) ⟶ (*i*+*n*) ∈ *set* (*nfilter P xs n*))
using *assms* **by** *auto*

lemma *nfilter-holds-b*:
 assumes $(\exists\ x \in set\ xs.\ P\ x)$
  shows $(\forall\ i{\le}intlen\ xs.\ (i{+}n) \in set\ (nfilter\ P\ xs\ n) = P\ (nth\ xs\ i))$
using *assms* **by** *auto*

lemma *nfilter-holds-c*:
 assumes $(\exists\ x \in set\ xs.\ P\ x)$
  $i{\le}intlen\ xs$
  shows $(i{+}n) \in set\ (nfilter\ P\ xs\ n) = P\ (nth\ xs\ i)$
by (*simp add*: *assms*(*1*) *assms*(*2*))

lemma *nfilter-holds-d*:
 assumes $(\exists\ x \in set\ xs.\ P\ x)$
  $n \le i$
  $i \le intlen\ xs\ {+}n$
  shows $i \in set\ (nfilter\ P\ xs\ n) = P\ (nth\ xs\ (i{-}n))$
using *assms*
by (*metis diff-add le-diff-conv nfilter-holds-b*)

lemma *nfilter-holds-not-b*:
 assumes $(\exists\ x \in set\ xs.\ P\ x)$
  $n \le i$
  $i \le intlen\ xs\ {+}n$
  shows $i \notin set\ (nfilter\ P\ xs\ n) = (\neg\ P\ (nth\ xs\ (i{-}n)))$
using *assms* **by** *auto*

lemma *nfilter-disjoint-set-coset*:
 assumes $(\exists\ x \in set\ xs.\ P\ x)$
  shows $(\{i{+}n|\ i.\ i{\le}\ intlen\ xs\} - (set\ (nfilter\ P\ xs\ n))) \cap (set\ (nfilter\ P\ xs\ n)) = \{\}$
using *assms* **by** *auto*

lemma *nfilter-not-before*:
 assumes $(\exists\ x \in set\ xs.\ P\ x)$
  $i{<}\ (nth\ (nfilter\ P\ xs\ 0)\ 0)$
 shows $\neg\ P\ (nth\ xs\ i)$
proof −
 have *0*: $(nth\ (nfilter\ P\ xs\ 0)\ 0) \le intlen\ xs$
  by (*metis add.left-neutral assms*(*1*) *interval-intlen-gr-zero nfilter-upper-bound*)
 have *1*: $i \notin set\ (nfilter\ P\ xs\ 0)$
  using *assms*
  proof (*induction xs arbitrary*: *i*)
  case (*St x*)
  then show *?case* **by** *simp*
  next
  case (*Cons x1a xs*)
  then show *?case*
   by (*metis idx-nfilter interval-idx-greater interval-nth-and-set interval-nth-zero-intfirst leD*)
  qed
 have *2*: $i \notin set\ (nfilter\ P\ xs\ 0) \wedge i{\le}\ intlen\ xs \longrightarrow \neg\ P\ (nth\ xs\ (i))$
  by (*metis add.right-neutral nfilter-holds-not-a nth-set*)

have *3*: $i \leq$ *intlen xs*
using *0 assms*(*2*) **by** *linarith*
from *0 1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *nfilter-n-not-before*:
assumes ($\exists$ $x \in$ *set* (*suffix n xs*). *P x*)
n $\leq$ *intlen xs*
n $\leq$*i*
i$<$ (*nth* (*nfilter P* (*suffix n xs*) *n*) *0*)
shows $\neg$ *P* (*nth xs* (*i*))
proof $-$
have *0*: (*nth* (*nfilter P* (*suffix n xs*) *n*) *0*) $\leq$ *intlen xs*
by (*metis assms*(*1*) *assms*(*2*) *interval-intlen-gr-zero interval-suffix-length-good*
le-add-diff-inverse nfilter-upper-bound)
have *1*: *i* $\notin$ *set* (*nfilter P* (*suffix n xs*) *n*)
using *assms*
proof (*induction xs arbitrary*: *i*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
by (*metis idx-nfilter interval-idx-greater interval-nth-and-set interval-nth-zero-intfirst leD*)
qed
have *2*: *i* $\notin$ *set* (*nfilter P* (*suffix n xs*) *n*) $\wedge$ *n* $\leq$ *i* $\wedge$ *i*$\leq$ *intlen xs* $\longrightarrow$ $\neg$ *P* (*nth xs* (*i*))
using *assms nfilter-holds-not-a*[*of* (*suffix n xs*) *P n*]
by (*metis add-le-imp-le-right interval-nth-suffix interval-suffix-length le-add-diff-inverse2*
ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
have *3*: *n* $\leq$ *i* $\wedge$ *i*$\leq$ *intlen xs*
using *0 assms*(*3*) *assms*(*4*) **by** *linarith*
from *0 1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *nfilter-not-after*:
assumes ($\exists$ $x \in$ *set xs*. *P x*)
(*nth* (*nfilter P xs 0*) (*intlen* (*nfilter P xs 0*))) $<$ *i*
i $\leq$ *intlen xs*
shows $\neg$ *P* (*nth xs* (*i*))
proof $-$
have *1*: *i* $\notin$ *set* (*nfilter P xs 0*)
using *assms*
proof (*induction xs arbitrary*: *i*)
case (*St x*)
then show *?case* **by** *auto*
next
case (*Cons x1a xs*)
then show *?case*
by (*metis idx-nfilter-gr-eq interval-nth-and-set leD le-refl*)
qed

have *2*: $i \notin$ *set* (*nfilter P xs 0*) $\wedge$ $i \leq$ *intlen xs* $\longrightarrow\neg$ *P* (*nth xs* (*i*))
  by (*metis add.right-neutral nfilter-holds-b nth-set*)
 have *3*: $i \leq$ *intlen xs*
   by (*simp add*: *assms*(*3*))
 from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *nfilter-n-not-after*:
 assumes ($\exists$ $x \in$ *set* (*suffix n xs*). *P x*)
  $n \leq$ *intlen xs*
  (*nth* (*nfilter P* (*suffix n xs*) *n*) (*intlen* (*nfilter P* (*suffix n xs*) *n*))) $<$ *i*
  $i \leq$ *intlen xs*
 shows $\neg$ *P* (*nth xs* (*i*))
proof −
 have *1*: $i \notin$ *set* (*nfilter P* (*suffix n xs*) *n*)
  using *assms*
   proof (*induction xs arbitrary*: *i*)
   case (*St x*)
   then show *?case* **by** *auto*
   next
   case (*Cons x1a xs*)
   then show *?case*
     by (*metis idx-nfilter-gr-eq interval-nth-and-set leD le-eq-less-or-eq*)
   qed
 have *2*: $i \notin$ *set* (*nfilter P* (*suffix n xs*) *n*) $\wedge$ $n\leq$ *i* $\wedge$ $i \leq$ *intlen xs* $\longrightarrow\neg$ *P* (*nth xs* (*i*))
    using *assms nfilter-holds-not-a*[*of suffix n xs P n*]
    by (*metis add-le-imp-le-left interval-nth-suffix interval-suffix-length le-add-diff-inverse2 ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
 have *3*: $n\leq$ *i* $\wedge$ $i \leq$ *intlen xs*
   by (*meson assms*(*1*) *assms*(*3*) *assms*(*4*) *dual-order.strict-implies-order dual-order.strict-trans2 nfilter-nth-n-zero-a order-refl*)
 from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *nfilter-not-between-help-a*:
assumes ($\bigwedge$*k i*.
   $\exists$ *a*$\in$*set xs*. *P a* $\Longrightarrow$
   k $<$ *intlen* (*nfilter P xs 0*) $\Longrightarrow$
   nth (*nfilter P xs 0*) *k* $<$ *i* $\Longrightarrow$
   i $<$ *nth* (*nfilter P xs 0*) (*Suc k*) $\Longrightarrow$
   nth (*nfilter P xs 0*) (*Suc k*) $\leq$ *intlen xs* $\Longrightarrow$
   $i \notin$ *set* (*nfilter P xs 0*))
  $\exists$ *a*$\in$*set* (*x1a* $\odot$ *xs*). *P a*
  k $<$ *intlen* (*nfilter P* (*x1a* $\odot$ *xs*) *0*)
  nth (*nfilter P* (*x1a* $\odot$ *xs*) *0*) *k* $<$ *i*
  i $<$ *nth* (*nfilter P* (*x1a* $\odot$ *xs*) *0*) (*Suc k*)
  nth (*nfilter P* (*x1a* $\odot$ *xs*) *0*) (*Suc k*) $\leq$ *intlen* (*x1a* $\odot$ *xs*)
  $\exists$ *x*$\in$*set xs*. *P x*
  P x1a

shows $i \notin set\ (nfilter\ P\ (x1a \odot xs)\ 0)$
proof −
have *1*: $k=0 \Longrightarrow i \notin set\ (nfilter\ P\ (x1a \odot xs)\ 0)$
using *assms* **by** *auto*
(*metis One-nat-def Suc-less-eq2 diff-Suc-1 interval-intlen-gr-zero nfilter-not-before nfilter-nth-n-zero*)
have *2*: $\bigwedge n.\ k= (Suc\ n) \Longrightarrow\ i \notin set\ (nfilter\ P\ (x1a \odot xs)\ 0)$
using *assms*
proof *auto*
fix *n*
fix *x*
fix *ka*
assume *a0*: $k= Suc\ n$
assume *a1*: $(\bigwedge k\ i.\ k < intlen\ (nfilter\ P\ xs\ 0) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ k < i \Longrightarrow$
$i < nth\ (nfilter\ P\ xs\ 0)\ (Suc\ k) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ (Suc\ k) \leq intlen\ xs \Longrightarrow$
$\neg\ P\ (nth\ xs\ i))$
assume *a2*: $n < intlen\ (nfilter\ P\ xs\ (Suc\ 0))$
assume *a3*: $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ n < Suc\ ka$
assume *a4*: $Suc\ ka < nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ n)$
assume *a5*: $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ n) \leq Suc\ (intlen\ xs)$
assume *a6*: $P\ x1a$
assume *a7*: $x \in set\ xs$
assume *a8*: $P\ x$
assume *a9*: $i = Suc\ ka$
assume *a10*: $P\ (nth\ xs\ ka)$
show *False*
proof −
have *3*: $n < intlen\ (nfilter\ P\ xs\ 0) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ n < ka \Longrightarrow$
$ka < nth\ (nfilter\ P\ xs\ 0)\ (Suc\ n) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ (Suc\ n) \leq intlen\ xs \Longrightarrow$
$\neg\ P\ (nth\ xs\ ka)$
using *a1*[*of n ka*] **by** *auto*
have *4*: $n < intlen\ (nfilter\ P\ xs\ 0)$
by (*metis a2 a7 a8 nfilter-intlen-n-zero*)
have *5*: $nth\ (nfilter\ P\ xs\ 0)\ (Suc\ n) \leq intlen\ xs$
by (*metis 4 Suc-leI add.commute add.right-neutral assms*(*7*) *nfilter-upper-bound*)
have *6*: $\exists x{\in}set\ xs.\ P\ x$
using *assms*(*7*) **by** *auto*
have *7*: $Suc\ 0 \leq intlen\ (nfilter\ P\ xs\ (Suc\ n))$
by (*metis One-nat-def Suc-le-mono Suc-mono a2 a7 a8 interval-intlen-gr-zero le-SucE nfilter-intlen-n-zero not-add-less1 plus-1-eq-Suc*)
have *8*: $ka < nth\ (nfilter\ P\ xs\ 0)\ (Suc\ n)$
using *nfilter-nth-n-zero*[*of xs P Suc n Suc 0*] *a4 6 a2* **by** *linarith*
have *9*: $nth\ (nfilter\ P\ xs\ 0)\ n < ka$
using *a2 a3 a7 a8 nfilter-nth-n-zero*[*of xs P n Suc 0*]
by (*metis One-nat-def add.commute dual-order.strict-implies-order less-diff-conv2 nfilter-lower-bound plus-1-eq-Suc*)

have *10*: $\neg$ *P* (*nth xs ka*)
using *3 4 9 8 5* **by** *auto*
from *a10 10* **show** *?thesis* **by** *auto*
qed
qed
show *?thesis* **using** *1 2*
using *gr0-implies-Suc* **by** *blast*
qed

lemma *nfilter-not-between-help-b*:
assumes ($\bigwedge$*k i*.
$\exists$ *a*$\in$*set xs*. *P a* $\Longrightarrow$
k $<$ *intlen* (*nfilter P xs 0*) $\Longrightarrow$
nth (*nfilter P xs 0*) *k* $<$ *i* $\Longrightarrow$
i $<$ *nth* (*nfilter P xs 0*) (*Suc k*) $\Longrightarrow$
nth (*nfilter P xs 0*) (*Suc k*) $\leq$ *intlen xs* $\Longrightarrow$
i $\notin$ *set* (*nfilter P xs 0*))
$\exists$ *a*$\in$*set* (*x1a* $\odot$ *xs*). *P a*
k $<$ *intlen* (*nfilter P* (*x1a* $\odot$ *xs*) *0*)
nth (*nfilter P* (*x1a* $\odot$ *xs*) *0*) *k* $<$ *i*
i $<$ *nth* (*nfilter P* (*x1a* $\odot$ *xs*) *0*) (*Suc k*)
nth (*nfilter P* (*x1a* $\odot$ *xs*) *0*) (*Suc k*) $\leq$ *intlen* (*x1a* $\odot$ *xs*)
$\exists$ *x*$\in$*set xs*. *P x*
$\neg$ *P x1a*
shows *i* $\notin$ *set* (*nfilter P* (*x1a* $\odot$ *xs*) *0*)
proof $-$
have *1*: *k=0* $\Longrightarrow$ *i* $\notin$ *set* (*nfilter P* (*x1a* $\odot$ *xs*) *0*)
using *assms*
proof *auto*
fix *x* :: *'a* **and** *ka* :: *nat*
assume *a1*: *nth* (*nfilter P xs* (*Suc 0*)) *0* $<$ *Suc ka*
assume *a2*: *x* $\in$ *set xs*
assume *a3*: *P x*
assume *a4*: $\bigwedge$*k i*. $[\![$*k* $<$ *intlen* (*nfilter P xs 0*); *nth* (*nfilter P xs 0*) *k* $<$ *i*;
i $<$ *nth* (*nfilter P xs 0*) (*Suc k*); *nth* (*nfilter P xs 0*) (*Suc k*) $\leq$ *intlen xs*$]\!]$
$\Longrightarrow$ $\neg$ *P* (*nth xs i*)
assume *a5*: *P* (*nth xs ka*)
assume *a6*: *0* $<$ *intlen* (*nfilter P xs* (*Suc 0*))
assume *a7*: *Suc ka* $<$ *nth* (*nfilter P xs* (*Suc 0*)) (*Suc 0*)
assume *a8*: *nth* (*nfilter P xs* (*Suc 0*)) (*Suc 0*) $\leq$ *Suc* (*intlen xs*)
have *f9*: $\exists$ *a*. *a* $\in$ *set xs* $\wedge$ *P a*
using *a3 a2* **by** *blast*
then have *f10*: *0* $\leq$ *intlen* (*nfilter P xs* (*Suc 0*)) $\longrightarrow$ *nth* (*nfilter P xs* (*Suc 0*)) *0* $\neq$ *0*
by (*metis le-zero-eq nfilter-lower-bound not-less-eq-eq*)
have *f11*: *intlen* (*nfilter P xs* (*Suc 0*)) = *intlen* (*nfilter P xs 0*)
using *f9* **by** (*meson nfilter-intlen-n-zero*)
obtain *nn* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* **where**
f12: *nth* (*nfilter P xs* (*Suc 0*)) (*Suc 0*) = *Suc* (*nn* (*nth* (*nfilter P xs* (*Suc 0*)) (*Suc 0*)) *ka*) $\wedge$
ka $<$ *nn* (*nth* (*nfilter P xs* (*Suc 0*)) (*Suc 0*)) *ka*
using *a7* **by** (*meson Suc-less-eq2*)

then have *nth* (*nfilter P xs 0*) (*Suc 0*) = *nn* (*nth* (*nfilter P xs* (*Suc 0*)) (*Suc 0*)) *ka*
using *f9 a6* **by** (*metis One-nat-def diff-Suc-1 le-zero-eq neq0-conv*
nfilter-nth-n-zero not-less-eq-eq)
then have $\neg$ *0* $\leq$ *intlen* (*nfilter P xs 0*)
using *f12 f11 f10 f9 a8 a6 a5 a4 a1*
by (*metis One-nat-def Suc-le-mono diff-Suc-1 less-Suc-eq-0-disj nfilter-nth-n-zero*)
then show *False*
by *blast*
qed
have *2*: $\bigwedge$*n. k*= (*Suc n*) $\Longrightarrow$ *i* $\notin$ *set* (*nfilter P* (*x1a* $\odot$ *xs*) *0*)
using *assms*
proof *auto*
fix *n*
fix *x*
fix *ka*
assume *a0*: *k* = *Suc n*
assume *a1*: ($\bigwedge$*k i. k* < *intlen* (*nfilter P xs 0*) $\Longrightarrow$
nth (*nfilter P xs 0*) *k* < *i* $\Longrightarrow$
i < *nth* (*nfilter P xs 0*) (*Suc k*) $\Longrightarrow$
nth (*nfilter P xs 0*) (*Suc k*) $\leq$ *intlen xs* $\Longrightarrow$
$\neg$ *P* (*nth xs i*))
assume *a2*: *Suc n* < *intlen* (*nfilter P xs* (*Suc 0*))
assume *a3*: *nth* (*nfilter P xs* (*Suc 0*)) (*Suc n*) < *Suc ka*
assume *a4*: *Suc ka* < *nth* (*nfilter P xs* (*Suc 0*)) (*Suc* (*Suc n*))
assume *a5*: *nth* (*nfilter P xs* (*Suc 0*)) (*Suc* (*Suc n*)) $\leq$ *Suc* (*intlen xs*)
assume *a6*: $\neg$ *P x1a*
assume *a7*: *x* $\in$ *set xs*
assume *a8*: *P x*
assume *a9*: *i* = *Suc ka*
assume *a10*: *P* (*nth xs ka*)
show *False*
proof −
have *3*: *Suc n* < *intlen* (*nfilter P xs 0*) $\Longrightarrow$
nth (*nfilter P xs 0*) (*Suc n*) < *ka* $\Longrightarrow$
ka < *nth* (*nfilter P xs 0*) (*Suc* (*Suc n*)) $\Longrightarrow$
nth (*nfilter P xs 0*) (*Suc* (*Suc n*)) $\leq$ *intlen xs* $\Longrightarrow$
$\neg$ *P* (*nth xs ka*)
using *a1*[*of Suc n ka*] **by** *auto*
have *4*: (*Suc n*) < *intlen* (*nfilter P xs 0*)
by (*metis a2 a7 a8 nfilter-intlen-n-zero*)
have *5*: *nth* (*nfilter P xs 0*) (*Suc* (*Suc n*)) $\leq$ *intlen xs*
by (*metis 4 Suc-leI a7 a8 add.left-neutral nfilter-upper-bound*)
have *6*: $\exists$*x*$\in$*set xs. P x*
by (*simp add*: *assms*(*7*))
have *7*: *Suc 0* $\leq$ *intlen* (*nfilter P xs* (*Suc n*))
by (*metis One-nat-def Suc-le-mono Suc-mono a2 a7 a8 interval-intlen-gr-zero le-SucE*
nfilter-intlen-n-zero not-add-less1 plus-1-eq-Suc)
have *8*: *ka* < *nth* (*nfilter P xs 0*) (*Suc* (*Suc n*))
using *nfilter-nth-n-zero*[*of xs P Suc* (*Suc n*) *Suc 0*] *a4 6 a2* **by** *linarith*
have *9*: *nth* (*nfilter P xs 0*) (*Suc n*) < *ka*

using *a2 a3 a7 a8 nfilter-nth-n-zero*[*of xs P Suc n Suc 0*]
by (*metis One-nat-def add.commute dual-order.strict-implies-order less-diff-conv2 nfilter-lower-bound plus-1-eq-Suc*)
have *10*: ¬ *P* (*nth xs ka*)
using *3 4 9 8 5* **by** *auto*
from *a10 10* **show** *?thesis* **by** *auto*
qed
qed
show *?thesis* **using** *1 2*
using *gr0-implies-Suc* **by** *blast*
qed

lemma *nfilter-not-between-help*:
assumes (∃ *x* ∈ *set xs*. *P x*)
k < *intlen* (*nfilter P xs 0*)
(*nth* (*nfilter P xs 0*) *k*) < *i*
i < (*nth* (*nfilter P xs 0*) (*Suc k*))
(*nth* (*nfilter P xs 0*) (*Suc k*)) ≤ *intlen xs*
shows *i* ∉ *set* (*nfilter P xs 0*)
using *assms*
proof (*induction xs arbitrary*: *i k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases* (∃ *x* ∈ *set xs*. *P x*))
show (⋀*k i*.
∃ *a*∈*set xs*. *P a* ⟹
k < *intlen* (*nfilter P xs 0*) ⟹
nth (*nfilter P xs 0*) *k* < *i* ⟹
i < *nth* (*nfilter P xs 0*) (*Suc k*) ⟹
nth (*nfilter P xs 0*) (*Suc k*) ≤ *intlen xs* ⟹
i ∉ *set* (*nfilter P xs 0*)) ⟹
∃ *a*∈*set* (*x1a* ⊙ *xs*). *P a* ⟹
k < *intlen* (*nfilter P* (*x1a* ⊙ *xs*) *0*) ⟹
nth (*nfilter P* (*x1a* ⊙ *xs*) *0*) *k* < *i* ⟹
i < *nth* (*nfilter P* (*x1a* ⊙ *xs*) *0*) (*Suc k*) ⟹
nth (*nfilter P* (*x1a* ⊙ *xs*) *0*) (*Suc k*) ≤ *intlen* (*x1a* ⊙ *xs*) ⟹
∃ *x*∈*set xs*. *P x* ⟹
i ∉ *set* (*nfilter P* (*x1a* ⊙ *xs*) *0*)
proof (*cases P x1a*)
show (⋀*k i*.
∃ *a*∈*set xs*. *P a* ⟹
k < *intlen* (*nfilter P xs 0*) ⟹
nth (*nfilter P xs 0*) *k* < *i* ⟹
i < *nth* (*nfilter P xs 0*) (*Suc k*) ⟹
nth (*nfilter P xs 0*) (*Suc k*) ≤ *intlen xs* ⟹
i ∉ *set* (*nfilter P xs 0*)) ⟹

$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$k < intlen\ (nfilter\ P\ (x1a \odot xs)\ 0) \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ k < i \Longrightarrow$
$i < nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ (Suc\ k) \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ (Suc\ k) \le intlen\ (x1a \odot xs) \Longrightarrow$
$\exists\, x{\in}set\ xs.\ P\ x \Longrightarrow$
$P\ x1a \Longrightarrow$
$i \notin set\ (nfilter\ P\ (x1a \odot xs)\ 0)$
using *nfilter-not-between-help-a*[*of xs P x1a k i*] **by** *simp*
show ($\bigwedge k\ i.$
$\exists\, a{\in}set\ xs.\ P\ a \Longrightarrow$
$k < intlen\ (nfilter\ P\ xs\ 0) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ k < i \Longrightarrow$
$i < nth\ (nfilter\ P\ xs\ 0)\ (Suc\ k) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ (Suc\ k) \le intlen\ xs \Longrightarrow$
$i \notin set\ (nfilter\ P\ xs\ 0)) \Longrightarrow$
$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$k < intlen\ (nfilter\ P\ (x1a \odot xs)\ 0) \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ k < i \Longrightarrow$
$i < nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ (Suc\ k) \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ (Suc\ k) \le intlen\ (x1a \odot xs) \Longrightarrow$
$\exists\, x{\in}set\ xs.\ P\ x \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$i \notin set\ (nfilter\ P\ (x1a \odot xs)\ 0)$
using *nfilter-not-between-help-b*[*of xs P x1a k i*] **by** *simp*
qed
show ($\bigwedge k\ i.$
$\exists\, a{\in}set\ xs.\ P\ a \Longrightarrow$
$k < intlen\ (nfilter\ P\ xs\ 0) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ k < i \Longrightarrow$
$i < nth\ (nfilter\ P\ xs\ 0)\ (Suc\ k) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ (Suc\ k) \le intlen\ xs \Longrightarrow$
$i \notin set\ (nfilter\ P\ xs\ 0)) \Longrightarrow$
$\exists\, a{\in}set\ (x1a \odot xs).\ P\ a \Longrightarrow$
$k < intlen\ (nfilter\ P\ (x1a \odot xs)\ 0) \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ k < i \Longrightarrow$
$i < nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ (Suc\ k) \Longrightarrow$
$nth\ (nfilter\ P\ (x1a \odot xs)\ 0)\ (Suc\ k) \le intlen\ (x1a \odot xs) \Longrightarrow$
$\neg\ (\exists\, x{\in}set\ xs.\ P\ x) \Longrightarrow$
$i \notin set\ (nfilter\ P\ (x1a \odot xs)\ 0)$
by *auto*
qed
qed

lemma *nfilter-not-between*:
assumes $(\exists\ x \in set\ xs.\ P\ x)$
$(nth\ (nfilter\ P\ xs\ 0)\ k\ ) < i$
$i < (nth\ (nfilter\ P\ xs\ 0)\ (Suc\ k)\ )$
$k < intlen\ (nfilter\ P\ xs\ 0)$

shows $\neg$ *P* (*nth xs* (*i*))
proof –
have *0*: (*nth* (*nfilter P xs 0*) (*Suc k*)) $\leq$ *intlen xs*
by (*metis Suc-leI add-cancel-right-left assms*(*1*) *assms*(*4*) *nfilter-upper-bound*)
have *1*: *i* $\leq$ *intlen xs*
using *0 assms*(*3*) **by** *linarith*
have *2*: *k* < *intlen* (*nfilter P xs 0*) $\wedge$ (*nth* (*nfilter P xs 0*) *k*) < *i* $\wedge$
i < (*nth* (*nfilter P xs 0*) (*Suc k*)) $\wedge$ (*nth* (*nfilter P xs 0*) (*Suc k*)) $\leq$ *intlen xs* $\longrightarrow$
i $\notin$ *set* (*nfilter P xs 0*)
using *assms*(*1*)
proof (*induction xs arbitrary*: *i k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case* **using** *assms nfilter-not-between-help*
by *metis*
qed
have *3* : *i* $\notin$ *set* (*nfilter P xs 0*) $\wedge$ *i* $\leq$ *intlen xs* $\longrightarrow \neg$ *P* (*nth xs* (*i*))
by (*metis add.right-neutral nfilter-holds-b nth-set*)
from *0 1 2 3* **show** *?thesis* **using** *assms* **by** *blast*
qed

lemma *idx-imp-distinct*:
assumes *index-sequence* (*nth xs 0*) *xs*
shows *distinct xs*
using *assms*
proof (*induction xs*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
by (*auto simp add*: *interval-idx-expand1*)
(*metis interval-idx-expand1 interval-idx-greater-first interval-nth-and-set*
interval-nth-zero-intfirst le-zero-eq not-less not-less-iff-gr-or-eq)
qed

lemma *idx-set-eq*:
assumes *index-sequence* (*nth xs 0*) *xs*
index-sequence (*nth ys 0*) *ys*
set xs = *set ys*
shows *xs* = *ys*
using *assms*
proof
(*induction xs arbitrary*: *ys*)

case (*St x*)
then show *?case*
  by (*metis Suc-leI index-sequence-def interval.simps*(*15*) *interval-suffix-intlen*
    interval-suffix-zero le0 nat-neq-iff neq0-conv nth-set singletonD)
next
case (*Cons x1a xs*)
then show *?case*
  proof (*cases ys*)
  case (*St x1*)
  then show *?thesis*
  using *Cons.prems*(*1*) *Cons.prems*(*3*) *interval-idx-less-last-1 le-eq-less-or-eq nth-set* **by** *fastforce*
  next
  case (*Cons x21 x22*)
  then show *?thesis*
   proof (*cases x1a* = *x21*)
   case *True*
   then show *?thesis*
    proof −
     have *1*: *intlen* (*x1a*⊙*xs*) = *intlen* (*x21*⊙*x22*)
        by (*metis Cons.prems*(*1*) *Cons.prems*(*2*) *Cons.prems*(*3*) *distinct-card idx-imp-distinct*
          local.Cons nat.inject)
     have *2*: *index-sequence* (*nth xs 0*) *xs*
        using *Cons.prems*(*1*) *interval-idx-expand1* **by** *auto*
     have *3*: *index-sequence* (*nth x22 0*) *x22*
        using *Cons.prems*(*2*) *interval-idx-expand1 local.Cons* **by** *auto*
     have *4*: *distinct xs*
        using *2 idx-imp-distinct* **by** *auto*
     have *5*: *distinct x22*
        by (*simp add*: *3 idx-imp-distinct*)
     have *6*: *set* (*x1a*⊙*xs*) = {*x1a*} ∪ *set xs*
        by *auto*
     have *7*: *x1a* ∉ *set xs*
        by (*meson Cons.prems*(*1*) *distinct.simps*(*2*) *idx-imp-distinct*)
     have *8*: *set* (*x21*⊙*x22*) = {*x21*} ∪ *set x22*
        by *auto*
     have *9*: *x21* ∉ *set x22*
        using *Cons.prems*(*2*) *idx-imp-distinct local.Cons* **by** *fastforce*
     have *10*: *set xs* =*set x22*
        using *7 9 Cons.prems*(*3*) *True local.Cons* **by** *fastforce*
     have *11*: *xs* = *x22*
        using *10 2 3 Cons.IH* **by** *blast*
     have *12*: *x1a*⊙ *xs* = *x21*⊙*x22*
        by (*simp add*: *11 True*)
     show *?thesis* **by** (*simp add*: *12 local.Cons*)
    qed
   next
   case *False*
   then show *?thesis*
   by (*metis Cons.prems*(*1*) *Cons.prems*(*2*) *Cons.prems*(*3*) *IntervalFilter.distinct.simps*(*2*)
     dual-order.strict-trans idx-imp-distinct interval-hd-in-set interval-idx-expand1

interval-nth-zero interval-set-ConsD local.Cons not-less-iff-gr-or-eq)
qed
qed
qed

lemma *filter-nfilter-prefix-idx-a*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k ≤ *intlen* (*filter P xs*)
shows *index-sequence* (*nth* (*prefix k* (*nfilter P xs 0*)) *0*) (*prefix k* (*nfilter P xs 0*))
using *assms* **by** (*auto simp add*: *index-sequence-def*)
(*metis diff-zero filter-nfilter-prefix-intlen-0 idx-nfilter-mono*)

lemma *filter-nfilter-prefix-idx-a-1*:
assumes ∃ *x* ∈ *set xs*. *P x*
k ≤ *intlen* (*filter P xs*)
shows *index-sequence* (*nth* (*prefix k* (*nfilter P xs 0*)) *0*) (*prefix k* (*nfilter P xs 0*))
using *assms* **by** (*auto simp add*: *index-sequence-def*)
(*meson idx-nfilter-mono*)

lemma *filter-nfilter-suffix-idx-a*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k ≤ *intlen* (*filter P xs*)
shows *index-sequence* (*nth* (*suffix k* (*nfilter P xs 0*)) *0*) (*suffix k* (*nfilter P xs 0*))
using *assms* **by** (*simp add*: *index-sequence-def*)
(*metis add.commute diff-zero filter-nfilter-prefix-intlen-0 idx-nfilter-mono less-diff-conv*)

lemma *filter-nfilter-suffix-idx-a-1*:
assumes ∃ *x* ∈ *set xs*. *P x*
k ≤ *intlen* (*filter P xs*)
shows *index-sequence* (*nth* (*suffix k* (*nfilter P xs 0*)) *0*) (*suffix k* (*nfilter P xs 0*))
using *assms*
by (*simp add*: *index-sequence-def idx-nfilter-mono nfilter-intlen*)

lemma *filter-nfilter-prefix-idx-b*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k ≤ *intlen* (*filter P xs*)
shows *index-sequence* (*nth* (*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) *0*)
(*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)
using *assms idx-nfilter*[*of* (*prefix* (*nth* (*nfilter P xs 0*) *k*) *xs*) *P 0*]
by (*metis add.right-neutral interval-intlast-intfirst interval-nth-intlen-intlast*
interval-nth-suffix interval-nth-zero-intfirst le0 le-refl nth-set)

lemma *filter-nfilter-prefix-idx-b-1*:
assumes ∃ *x* ∈ *set xs*. *P x*
k ≤ *intlen* (*filter P xs*)
shows *index-sequence* (*nth* (*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) *0*)
(*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)
using *assms*
by (*metis filter-nfilter-prefix-idx-b nfilter-holds nfilter-intlen nfilter-nth-n-zero nth-set*)

lemma *filter-nfilter-suffix-idx-b*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k ≤ *intlen* (*filter P xs*)
shows *index-sequence* (*nth* (*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)
(*nth* (*nfilter P xs 0*) *k*)) *0*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)
(*nth* (*nfilter P xs 0*) *k*))
using *assms*
by (*auto simp add*: *index-sequence-def*)
(*metis idx-nfilter-mono interval-intfirst-suffix interval-intlen-gr-zero interval-nth-zero-intfirst interval-suffix-length-code length-nfilter-le less-le-trans not-less0 nth-set*)

lemma *filter-nfilter-suffix-idx-b-1*:
assumes ∃ *x* ∈ *set xs*. *P x*
k ≤ *intlen* (*filter P xs*)
shows *index-sequence* (*nth* (*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)
(*nth* (*nfilter P xs 0*) *k*)) *0*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*))
using *assms*
by (*metis filter-nfilter-suffix-idx-b nfilter-holds nfilter-intlen nfilter-nth-n-zero nth-set*)

lemma *filter-nfilter-prefix-set-eq*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k ≤ *intlen* (*filter P xs*)
shows *set* (*prefix k* (*nfilter P xs 0*)) =
set (*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)
proof −
have *1*: (*nth* (*nfilter P xs 0*) *k*) ≤ *intlen xs*
by (*metis assms*(*1*) *assms*(*2*) *diff-zero filter-nfilter-prefix-intlen-0 interval-intlen-gr-zero nfilter-intlen nfilter-upper-bound ordered-cancel-comm-monoid-diff-class*.*add-diff-inverse*)
have *2*: ∃*x* ∈ *set xs* . *P x*
using *1 assms*(*1*) *nth-set* **by** *blast*
have *3*: { *i*. *i*≤ *intlen*(*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) ∧
P (*nth* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *i*)} =

{ *i*. *i*≤ (*nth* (*nfilter P xs 0*) *k*) ∧ *P* (*nth xs i*)}
using *1* **by** *auto*
have *4*: ∃ *x* ∈ *set*(*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*). *P x*
by (*metis 1 assms*(*1*) *interval-nth-prefix interval-prefix-length-good nth-set order-refl*)
have *5*: { *i*. *i*≤ (*nth* (*nfilter P xs 0*) *k*) ∧ *P* (*nth xs i*)} =
{ *i*. *i*≤ (*nth* (*nfilter P xs 0*) *k*) ∧ *i*∈*set*(*nfilter P xs 0*)}
using *4 1 2 nfilter-holds-b* **by** *auto*
have *6*: { *i*. *i*≤ (*nth* (*nfilter P xs 0*) *k*) ∧ *i*∈*set*(*nfilter P xs 0*)} =
{ *i*. *i*≤ (*nth* (*nfilter P xs 0*) *k*) ∧
i∈{(*nth* (*nfilter P xs 0*) *j*)| *j*. *j*≤ *intlen*(*nfilter P xs 0*)} }
by (*auto simp add*: *interval-nth-and-set*)
have *7*: { *i*. *i*≤ (*nth* (*nfilter P xs 0*) *k*) ∧
i∈{(*nth* (*nfilter P xs 0*) *j*)| *j*. *j*≤ *intlen*(*nfilter P xs 0*)} } =
{ (*nth* (*nfilter P xs 0*) *j*) | *j*. *j*≤ *k* }

```
   using assms 2 by (auto simp add: nfilter-intlen,
       metis dual-order.antisym idx-nfilter interval-idx-less-eq le-cases nfilter-intlen,
       metis idx-nfilter-less-eq nfilter-intlen)
 have 8: k≤ intlen (nfilter P xs 0)
   by (simp add: 2 assms(2) nfilter-intlen)
 have 9: { (nth (nfilter P xs 0) j) | j. j≤ k } = set (prefix k (nfilter P xs 0))
   using 8 2 using prefix-set by force
 have 10: set (nfilter P (prefix ((nth (nfilter P xs 0) k) ) xs) 0) =
          { i. i≤ ((nth (nfilter P xs 0) k)) ∧
                 P (nth (prefix ((nth (nfilter P xs 0) k)) xs) i)}
  using 4 1  nfilter-prefix-set by auto
 have 11: intlen(prefix ((nth (nfilter P xs 0) k)) xs) = ((nth (nfilter P xs 0) k))
  using 1 interval-prefix-length-good by blast
 have 12: { i. i≤ ((nth (nfilter P xs 0) k)) ∧
                 P (nth (prefix ((nth (nfilter P xs 0) k)) xs) i)} =
          { i. i≤ intlen(prefix ((nth (nfilter P xs 0) k)) xs) ∧
                 P (nth (prefix ((nth (nfilter P xs 0) k)) xs) i)}
   using 11 by auto
 show ?thesis
 using 10 12 3 5 6 7 9 by auto
qed

lemma filter-nfilter-prefix-set-eq-1:
  assumes ∃x ∈ set xs . P x
          k ≤ intlen (filter P xs)
  shows    set (prefix k (nfilter P xs 0)) =
           set (nfilter P (prefix ((nth (nfilter P xs 0) k) ) xs) 0)
proof −
 have 1: (nth (nfilter P xs 0) k)  ≤ intlen xs
   by (metis assms(1) assms(2) diff-zero  interval-intlen-gr-zero
   nfilter-intlen nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
 have 2: ∃x ∈ set xs . P x
   using 1 assms(1) nth-set by blast
 have 3: { i. i≤ intlen(prefix ((nth (nfilter P xs 0) k)) xs) ∧
                 P (nth (prefix ((nth (nfilter P xs 0) k)) xs) i)} =

         { i. i≤ (nth (nfilter P xs 0) k) ∧ P (nth xs i)}
  using 1 by auto
 have 4: ∃ x ∈ set(prefix ((nth (nfilter P xs 0) k))    xs). P x
   using assms 1 2 nfilter-holds[of xs P 0]
   by (metis interval-intlast-prefix interval-nth-intlen-intlast le-refl nfilter-intlen
       nfilter-nth-n-zero nth-set)
 have 5: { i. i≤ (nth (nfilter P xs 0) k) ∧ P (nth xs i)} =
         { i. i≤ (nth (nfilter P xs 0) k) ∧ i∈set(nfilter P xs 0)}
   using 4 1 2 nfilter-holds-b by auto
 have 6: { i. i≤ (nth (nfilter P xs 0) k) ∧ i∈set(nfilter P xs 0)} =
         { i. i≤ (nth (nfilter P xs 0) k) ∧
              i∈{(nth (nfilter P xs 0) j)| j. j≤ intlen(nfilter P xs 0)}   }
   by (auto simp add: interval-nth-and-set)
 have 7: { i. i≤ (nth (nfilter P xs 0) k) ∧
```

$i \in \{(nth\ (nfilter\ P\ xs\ 0)\ j) \mid j.\ j \leq intlen(nfilter\ P\ xs\ 0)\}\ \ \} =$
$\{\ (nth\ (nfilter\ P\ xs\ 0)\ j)\ \mid j.\ j \leq\ k\ \}$
using *assms 2* **by** (*auto simp add*: *nfilter-intlen*,
metis dual-order.antisym idx-nfilter interval-idx-less-eq le-cases nfilter-intlen,
metis idx-nfilter-less-eq nfilter-intlen)
have *8*: $k \leq$ *intlen* (*nfilter P xs 0*)
by (*simp add*: *2 assms*(*2*) *nfilter-intlen*)
have *9*: $\{\ (nth\ (nfilter\ P\ xs\ 0)\ j)\ \mid j.\ j \leq\ k\ \} = set\ (prefix\ k\ (nfilter\ P\ xs\ 0))$
using *8 2* **using** *prefix-set* **by** *force*
have *10*: $set\ (nfilter\ P\ (prefix\ ((nth\ (nfilter\ P\ xs\ 0)\ k)\ )\ xs)\ 0) =$
$\{\ i.\ i \leq ((nth\ (nfilter\ P\ xs\ 0)\ k)) \wedge$
$P\ (nth\ (prefix\ ((nth\ (nfilter\ P\ xs\ 0)\ k))\ xs)\ i)\}$
using *4 1 nfilter-prefix-set* **by** *auto*
have *11*: $intlen(prefix\ ((nth\ (nfilter\ P\ xs\ 0)\ k))\ xs) = ((nth\ (nfilter\ P\ xs\ 0)\ k))$
using *1 interval-prefix-length-good* **by** *blast*
have *12*: $\{\ i.\ i \leq ((nth\ (nfilter\ P\ xs\ 0)\ k)) \wedge$
$P\ (nth\ (prefix\ ((nth\ (nfilter\ P\ xs\ 0)\ k))\ xs)\ i)\} =$
$\{\ i.\ i \leq intlen(prefix\ ((nth\ (nfilter\ P\ xs\ 0)\ k))\ xs) \wedge$
$P\ (nth\ (prefix\ ((nth\ (nfilter\ P\ xs\ 0)\ k))\ xs)\ i)\}$
using *11* **by** *auto*
show *?thesis*
using *10 12 3 5 6 7 9* **by** *auto*
qed

lemma *filter-nfilter-suffix-set-eq*:
assumes $P\ (nth\ xs\ \ (\ (nth\ (nfilter\ P\ xs\ 0)\ k))\ )$
$k \leq intlen\ (filter\ P\ xs)$
shows $set\ (nfilter\ P\ (suffix\ (nth\ (nfilter\ P\ xs\ 0)\ k)\ \ xs)\ (nth\ (nfilter\ P\ xs\ 0)\ k)) =$
$set\ (suffix\ k\ (nfilter\ P\ xs\ 0))$

proof −
have *1*: $(nth\ (nfilter\ P\ xs\ 0)\ k)\ \ \leq intlen\ xs$
by (*metis assms*(*1*) *assms*(*2*) *diff-zero filter-nfilter-prefix-intlen-0 interval-intlen-gr-zero nfilter-intlen nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *2*: $\exists x \in set\ xs\ .\ P\ x$
using *1 assms*(*1*) *nth-set* **by** *blast*
have *4*: $\exists\ x \in set(suffix\ ((nth\ (nfilter\ P\ xs\ 0)\ k))\ \ \ \ xs).\ P\ x$
using *1 assms*(*1*) *interval-nth-and-set* **by** *force*
have *10*: $set\ (nfilter\ P\ (suffix\ ((nth\ (nfilter\ P\ xs\ 0)\ k)\ )\ xs)\ (nth\ (nfilter\ P\ xs\ 0)\ k)) =$
$\{\ (nth\ (nfilter\ P\ xs\ 0)\ k)+i \mid\ i.\ i \leq intlen\ xs - (nth\ (nfilter\ P\ xs\ 0)\ k) \wedge$
$P\ (nth\ xs\ (i+(nth\ (nfilter\ P\ xs\ 0)\ k))\ )\}$

using *nfilter-suffix-set-b*[*of* ((*nth* (*nfilter P xs 0*) *k*)) *xs P* (*nth* (*nfilter P xs 0*) *k*)]
using *1 4* **by** *blast*
have *5*: $\{\ (nth\ (nfilter\ P\ xs\ 0)\ k)+i \mid i.\ \ i \leq intlen\ xs - (nth\ (nfilter\ P\ xs\ 0)\ k) \wedge$
$P\ (nth\ xs\ (i+(nth\ (nfilter\ P\ xs\ 0)\ k))\ \ )\}$
$=$
$\{\ (nth\ (nfilter\ P\ xs\ 0)\ k)+i \mid i.\ i \leq intlen\ xs - (nth\ (nfilter\ P\ xs\ 0)\ k) \wedge$
$i+(nth\ (nfilter\ P\ xs\ 0)\ k) \in set(nfilter\ P\ xs\ 0)\}$
using *4 1 2 nfilter-holds-b* **by** *auto*

have *51*: { (*nth* (*nfilter P xs 0*) *k*)+*i* |*i*. *i*≤ *intlen xs* − (*nth* (*nfilter P xs 0*) *k*) ∧
i+(*nth* (*nfilter P xs 0*) *k*)∈*set*(*nfilter P xs 0*)} =
{ (*nth* (*nfilter P xs 0*) *k*)+*i*| *i*.
(*nth* (*nfilter P xs 0*) *k*) ≤ *i* + (*nth* (*nfilter P xs 0*) *k*) ∧
i + (*nth* (*nfilter P xs 0*) *k*)≤ *intlen xs* ∧
i+(*nth* (*nfilter P xs 0*) *k*)∈*set*(*nfilter P xs 0*)}

using *1* **by** *auto*
have *52*: { (*nth* (*nfilter P xs 0*) *k*)+*i*| *i*.
(*nth* (*nfilter P xs 0*) *k*) ≤ *i* + (*nth* (*nfilter P xs 0*) *k*) ∧
i + (*nth* (*nfilter P xs 0*) *k*)≤ *intlen xs* ∧
i+(*nth* (*nfilter P xs 0*) *k*)∈*set*(*nfilter P xs 0*)} =
{ *j*. (*nth* (*nfilter P xs 0*) *k*) ≤ *j* ∧ *j*≤ *intlen xs* ∧ *j*∈*set*(*nfilter P xs 0*)}

by (*metis* (*no-types*, *lifting*) *add-diff-cancel-right′ le0 le-add-diff-inverse*
le-add-same-cancel2)
have *53*: { *j*. (*nth* (*nfilter P xs 0*) *k*) ≤ *j* ∧ *j*≤ *intlen xs* ∧ *j*∈*set*(*nfilter P xs 0*)} =
{ *j*. (*nth* (*nfilter P xs 0*) *k*) ≤ *j* ∧
j≤ *intlen xs* ∧ *j*∈{(*nth* (*nfilter P xs 0*) *jj*)| *jj*. *jj*≤ *intlen*(*nfilter P xs 0*)} }

by (*auto simp add*: *interval-nth-and-set*)
have *54*: { *j*. (*nth* (*nfilter P xs 0*) *k*) ≤ *j* ∧
j≤ *intlen xs* ∧ *j*∈{(*nth* (*nfilter P xs 0*) *jj*)| *jj*. *jj*≤ *intlen*(*nfilter P xs 0*)} } =
{ (*nth* (*nfilter P xs 0*) *j*) | *j*. *k* ≤*j* ∧ *j* ≤ *intlen* (*nfilter P xs 0*) }

using *assms 2* **by** (*auto*,
metis dual-order.antisym idx-nfilter-less-eq le-cases nfilter-intlen,
metis idx-nfilter-gr-eq,
metis add-cancel-right-left nfilter-upper-bound)
have *8*: *k*≤ *intlen* (*nfilter P xs 0*)
by (*simp add*: *2 assms*(*2*) *nfilter-intlen*)
have *9*: { (*nth* (*nfilter P xs 0*) *j*) | *j*. *k* ≤*j* ∧ *j* ≤ *intlen* (*nfilter P xs 0*) } =
set (*suffix k* (*nfilter P xs 0*))
using *8 2* **using** *suffix-set-a* **by** *blast*
show *?thesis*
using *10 5 51 52 53 54 9* **by** *simp*
qed

lemma *filter-nfilter-suffix-set-eq-1*:
assumes ∃ *x* ∈ *set xs* . *P x*
k ≤ *intlen* (*filter P xs*)
shows *set* (*nfilter P* (*suffix* (*nth* (*nfilter P xs 0*) *k*) *xs*) (*nth* (*nfilter P xs 0*) *k*)) =
set (*suffix k* (*nfilter P xs 0*))

proof −
have *1*: (*nth* (*nfilter P xs 0*) *k*) ≤ *intlen xs*
by (*metis assms*(*1*) *assms*(*2*) *diff-zero interval-intlen-gr-zero*
nfilter-intlen nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
have *2*: ∃ *x* ∈ *set xs* . *P x*
using *assms*(*1*) **by** *blast*

have *4*: ∃ *x* ∈ *set*(*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*). *P x*
using *nfilter-holds*[*of xs P 0*]
by (*metis 1 2 assms*(*2*) *interval-intfirst-suffix interval-nth-zero-intfirst le0 nfilter-intlen nfilter-nth-n-zero nth-set*)
have *10*: *set* (*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*)) =
{ (*nth* (*nfilter P xs 0*) *k*)+*i*| *i*. *i*≤ *intlen xs* − (*nth* (*nfilter P xs 0*) *k*) ∧
P (*nth xs* (*i*+(*nth* (*nfilter P xs 0*) *k*)))}

using *nfilter-suffix-set-b*[*of* (*nth* (*nfilter P xs 0*) *k*) *xs P* (*nth* (*nfilter P xs 0*) *k*)]
using *1 4* **by** *blast*
have *5*: { (*nth* (*nfilter P xs 0*) *k*)+*i*|*i*. *i*≤ *intlen xs* − (*nth* (*nfilter P xs 0*) *k*) ∧
P (*nth xs* (*i*+(*nth* (*nfilter P xs 0*) *k*)))}
=
{ (*nth* (*nfilter P xs 0*) *k*)+*i*|*i*. *i*≤ *intlen xs* − (*nth* (*nfilter P xs 0*) *k*) ∧
i+(*nth* (*nfilter P xs 0*) *k*)∈*set*(*nfilter P xs 0*)}
using *4 1 2 nfilter-holds-b* **by** *auto*
have *51*: { (*nth* (*nfilter P xs 0*) *k*)+*i* |*i*. *i*≤ *intlen xs* − (*nth* (*nfilter P xs 0*) *k*) ∧
i+(*nth* (*nfilter P xs 0*) *k*)∈*set*(*nfilter P xs 0*)} =
{ (*nth* (*nfilter P xs 0*) *k*)+*i*| *i*.
(*nth* (*nfilter P xs 0*) *k*) ≤ *i* + (*nth* (*nfilter P xs 0*) *k*) ∧
i + (*nth* (*nfilter P xs 0*) *k*)≤ *intlen xs* ∧
i+(*nth* (*nfilter P xs 0*) *k*)∈*set*(*nfilter P xs 0*)}

using *1* **by** *auto*
have *52*: { (*nth* (*nfilter P xs 0*) *k*)+*i*| *i*.
(*nth* (*nfilter P xs 0*) *k*) ≤ *i* + (*nth* (*nfilter P xs 0*) *k*) ∧
i + (*nth* (*nfilter P xs 0*) *k*)≤ *intlen xs* ∧
i+(*nth* (*nfilter P xs 0*) *k*)∈*set*(*nfilter P xs 0*)} =
{ *j*. (*nth* (*nfilter P xs 0*) *k*) ≤ *j* ∧ *j*≤ *intlen xs* ∧ *j*∈*set*(*nfilter P xs 0*)}

by (*metis* (*no-types*, *lifting*) *add-diff-cancel-right′ le0 le-add-diff-inverse le-add-same-cancel2*)
have *53*: { *j*. (*nth* (*nfilter P xs 0*) *k*) ≤ *j* ∧ *j*≤ *intlen xs* ∧ *j*∈*set*(*nfilter P xs 0*)} =
{ *j*. (*nth* (*nfilter P xs 0*) *k*) ≤ *j* ∧
j≤ *intlen xs* ∧ *j*∈{(*nth* (*nfilter P xs 0*) *jj*)| *jj*. *jj*≤ *intlen*(*nfilter P xs 0*)} }

by (*auto simp add*: *interval-nth-and-set*)
have *54*: { *j*. (*nth* (*nfilter P xs 0*) *k*) ≤ *j* ∧
j≤ *intlen xs* ∧ *j*∈{(*nth* (*nfilter P xs 0*) *jj*)| *jj*. *jj*≤ *intlen*(*nfilter P xs 0*)} } =
{ (*nth* (*nfilter P xs 0*) *j*) | *j*. *k* ≤*j* ∧ *j* ≤ *intlen* (*nfilter P xs 0*) }

using *assms 2* **by** (*auto*,
metis dual-order.antisym idx-nfilter-less-eq le-cases nfilter-intlen,
simp add: *2 idx-nfilter-less-eq*,
metis add-cancel-right-left nfilter-upper-bound)
have *8*: *k*≤ *intlen* (*nfilter P xs 0*)
by (*simp add*: *2 assms*(*2*) *nfilter-intlen*)
have *9*: { (*nth* (*nfilter P xs 0*) *j*) | *j*. *k* ≤*j* ∧ *j* ≤ *intlen* (*nfilter P xs 0*) } =
set (*suffix k* (*nfilter P xs 0*))
using *8 2* **using** *suffix-set-a* **by** *blast*

show *?thesis*
using *10 5 51 52 53 54 9* **by** *simp*
qed

lemma *nfilter-nfilter-prefix*:
assumes P (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
$k \leq$ *intlen* (*filter P xs*)
shows (*prefix k* (*nfilter P xs 0*)) =
(*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)
using *assms*
by (*meson filter-nfilter-prefix-idx-a filter-nfilter-prefix-idx-b*
filter-nfilter-prefix-set-eq idx-set-eq)

lemma *nfilter-nfilter-prefix-1*:
assumes $\exists x \in$ *set xs* . $P\ x$
$k \leq$ *intlen* (*filter P xs*)
shows (*prefix k* (*nfilter P xs 0*)) =
(*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)
using *assms*
by (*meson filter-nfilter-prefix-idx-a-1 filter-nfilter-prefix-idx-b-1*
filter-nfilter-prefix-set-eq-1 idx-set-eq)

lemma *nfilter-nfilter-suffix*:
assumes P (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
$k \leq$ *intlen* (*filter P xs*)
shows (*suffix k* (*nfilter P xs 0*)) =
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*))
using *assms*
by (*metis filter-nfilter-suffix-idx-a filter-nfilter-suffix-idx-b*
filter-nfilter-suffix-set-eq idx-set-eq)

lemma *nfilter-nfilter-suffix-1*:
assumes $\exists x \in$ *set xs* . $P\ x$
$k \leq$ *intlen* (*filter P xs*)
shows (*suffix k* (*nfilter P xs 0*)) =
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*))
proof −
have *1*: P (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
by (*metis assms*(*1*) *assms*(*2*) *nfilter-holds nfilter-intlen nfilter-nth-n-zero nth-set*)
show *?thesis* **using** *assms 1 nfilter-nfilter-suffix* **by** *auto*
qed

lemma *nfilter-map-filter*:
assumes ($\exists$ $x \in$ *set xs*. $P\ x$)
shows *map* (λn. (*nth xs n*)) (*nfilter P xs 0*) = *filter P xs*
proof −
have *1*: *intlen* (*map* (λn. (*nth xs n*)) (*nfilter P xs 0*)) = *intlen* (*filter P xs*)

by (*simp add*: *assms nfilter-intlen*)
have *2*: $\bigwedge i.\ i \leq$ *intlen* (*map* ($\lambda n.$ (*nth xs n*)) (*nfilter P xs 0*)) $\longrightarrow$
(*nth* (*map* ($\lambda n.$ (*nth xs n*)) (*nfilter P xs 0*)) *i*) =
(*nth* (*filter P xs*) *i*)
by (*metis assms diff-zero interval-intlen-map interval-nth-map nfilter-filter*)
from *1 2* **show** *?thesis*
using *interval-eq-nth-eq* **by** *blast*
qed

lemma *filter-nfilter-prefix*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k $\leq$ *intlen* (*filter P xs*)
shows (*prefix k* (*filter P xs*)) =
(*filter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*))
proof −
have *1*: $\exists x \in$ *set xs* . *P x*
by (*metis assms*(*1*) *assms*(*2*) *diff-zero filter-nfilter-prefix-intlen-0*)
have *2*: (*filter P xs*) = *map* ($\lambda s.$ *nth xs s*) (*nfilter P xs 0*)
by (*simp add*: *1 nfilter-map-filter*)
have *3*: (*prefix k* (*filter P xs*)) =
(*prefix k* (*map* ($\lambda s.$ *nth xs s*) (*nfilter P xs 0*)))
by (*simp add*: *2*)
have *4*: (*prefix k* (*map* ($\lambda s.$ *nth xs s*) (*nfilter P xs 0*))) =
map ($\lambda s.$ *nth xs s*) (*prefix k* (*nfilter P xs 0*))
by (*simp add*: *1 assms*(*2*) *map-prefix nfilter-intlen*)
have *5*: $\exists$ *x* $\in$ *set*(*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*). *P x*
by (*metis 1 add.left-neutral assms*(*1*) *assms*(*2*) *interval-intlast-prefix interval-nth-and-set interval-nth-intlen-intlast nfilter-intlen nfilter-upper-bound order-refl*)
have *6*: (*filter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)) =
map ($\lambda s.$ (*nth* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *s*))
(*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)
by (*simp add*: *5 nfilter-map-filter*)
have *7*: *map* ($\lambda s.$ *nth xs s*) (*prefix k* (*nfilter P xs 0*)) =
map ($\lambda s.$ *nth xs s*) (*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nfilter-prefix*)
have *8*: *map* ($\lambda s.$ *nth xs s*) (*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) =
map ($\lambda s.$ (*nth* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *s*))
(*nfilter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)

using *1 5* **by** *simp*
show *?thesis*
by (*simp add*: *3 4 6 7 8*)
qed

lemma *filter-nfilter-prefix-1*:
assumes $\exists x \in$ *set xs* . *P x*
k $\leq$ *intlen* (*filter P xs*)
shows (*prefix k* (*filter P xs*)) =
(*filter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*))
proof −

have *1*: *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
  by (*metis assms*(*1*) *assms*(*2*) *nfilter-holds nfilter-intlen nfilter-nth-n-zero nth-set*)
show *?thesis* **using** *assms 1 filter-nfilter-prefix* **by** *auto*
qed

lemma *filter-nfilter-prefix-intlen*:
  assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
    $k \leq$ *intlen* (*filter P xs*)
  shows *intlen*(*prefix k* (*filter P xs*)) =
    intlen(*filter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*))
using *assms*
by (*simp add*: *filter-nfilter-prefix*)

lemma *filter-nfilter-prefix-intlen-1*:
  assumes $\exists x \in$ *set xs* . *P x*
    $k \leq$ *intlen* (*filter P xs*)
  shows *intlen*(*prefix k* (*filter P xs*)) =
    intlen(*filter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*))
using *assms*
by (*simp add*: *filter-nfilter-prefix-1*)

lemma *filter-nfilter-prefix-nth*:
  assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
    $k \leq$ *intlen* (*filter P xs*)
    $j \leq$ *intlen*(*prefix k* (*filter P xs*))
  shows *nth* (*prefix k* (*filter P xs*)) *j* =
    nth (*filter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)) *j*
using *assms*
by (*simp add*: *filter-nfilter-prefix*)

lemma *filter-nfilter-prefix-nth-1*:
  assumes $\exists x \in$ *set xs* . *P x*
    $k \leq$ *intlen* (*filter P xs*)
    $j \leq$ *intlen*(*prefix k* (*filter P xs*))
  shows *nth* (*prefix k* (*filter P xs*)) *j* =
    nth (*filter P* (*prefix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)) *j*
using *assms*
by (*simp add*: *filter-nfilter-prefix-1*)

lemma *nfilter-map-shift*:
assumes $\exists$ $x \in$ *set xs*. *P x*
shows *map* (λs. *nth xs* (*s*+*n*)) (*nfilter P xs 0*) =
  map (λs. *nth xs s*) (*nfilter P xs n*)
proof −
have *1*: *intlen* (*map* (λs. *nth xs* (*s*+*n*)) (*nfilter P xs 0*)) =
    intlen (*map* (λs. *nth xs s*) (*nfilter P xs n*))
  by (*metis assms*(*1*) *interval-intlen-map nfilter-intlen-n-zero*)
have *2*: $\bigwedge i$. *nth* (*map* (λs. *nth xs* (*s*+*n*)) (*nfilter P xs 0*)) *i* =
    (*nth xs* ((*nth* (*nfilter P xs 0*) *i*) +*n*))
  by (*simp add*: *interval-nth-map*)

have *3*: $\bigwedge i$. *nth* (*map* (λ*s*. *nth xs s*) (*nfilter P xs n*)) *i* =
(*nth xs* ((*nth* (*nfilter P xs n*) *i*)))
by (*simp add*: *interval-nth-map*)
have *4*: $\bigwedge i$. (*nth* (*nfilter P xs 0*) *i*) +*n* = (*nth* (*nfilter P xs n*) *i*)
by (*metis assms*(*1*) *interval-nth-map nfilter-n-zero*)
show *?thesis*
by (*metis 1 2 3 4 interval-eq-nth-eq*)
qed

lemma *nfilter-map-shift-suffix*:
assumes ∃ *x* ∈ *set*(*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*). *P x*
shows *map* (λ*s*. *nth xs* (*s*+(*nth* (*nfilter P xs 0*) *k*)))
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) =
map (λ*s*. *nth xs s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*))
proof −
have *1*: *intlen* (*map* (λ*s*. *nth xs* (*s*+(*nth* (*nfilter P xs 0*) *k*)))
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)) =
intlen (*map* (λ*s*. *nth xs s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*)))
by (*metis assms interval-intlen-map nfilter-intlen-n-zero*)
have *2*: $\bigwedge i$. *nth* (*map* (λ*s*. *nth xs* (*s*+(*nth* (*nfilter P xs 0*) *k*)))
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)) *i* =
nth xs
((*nth* (*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) *i*) + (*nth* (*nfilter P xs 0*) *k*))

using *interval-nth-map* **by** *blast*
have *3*: $\bigwedge i$.
nth (*map* (λ*s*. *nth xs s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)
(*nth* (*nfilter P xs 0*) *k*))) *i* =
nth xs (*nth* (*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)
(*nth* (*nfilter P xs 0*) *k*)) *i*)

using *interval-nth-map* **by** *blast*
have *4*: $\bigwedge i$. (*nth* (*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) *i*)
+ (*nth* (*nfilter P xs 0*) *k*) =
(*nth* (*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*)) *i*)
by (*metis assms interval-nth-map nfilter-n-zero*)
show *?thesis*
by (*metis 1 2 3 4 interval-eq-nth-eq*)
qed

lemma *filter-nfilter-suffix*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k ≤ *intlen* (*filter P xs*)
shows (*suffix k* (*filter P xs*)) =
(*filter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*))
proof −
have *1*: ∃ *x* ∈ *set xs* . *P x*

by (*metis assms*(*1*) *assms*(*2*) *diff-zero filter-nfilter-prefix-intlen-0*)
have *2*: (*filter P xs*) = *map* (λ*s*. *nth xs s*) (*nfilter P xs 0*)
by (*simp add*: *1 nfilter-map-filter*)
have *3*: (*suffix k* (*filter P xs*)) =
(*suffix k* (*map* (λ*s*. *nth xs s*) (*nfilter P xs 0*)))
by (*simp add*: *2*)
have *4*: (*suffix k* (*map* (λ*s*. *nth xs s*) (*nfilter P xs 0*))) =
map (λ*s*. *nth xs s*) (*suffix k* (*nfilter P xs 0*))
by (*simp add*: *1 assms*(*2*) *map-suffix nfilter-intlen*)
have *5*: ∃ *x* ∈ *set*(*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*). *P x*
by (*metis 1 add-cancel-right-left assms*(*1*) *assms*(*2*) *interval-intfirst-suffix*
interval-intlen-gr-zero interval-nth-and-set interval-nth-zero-intfirst nfilter-intlen
nfilter-upper-bound)
have *6*: (*filter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)) =
map (λ*s*. *nth* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)

by (*simp add*: *5 nfilter-map-filter*)
have *7*: *map* (λ*s*. *nth* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) =
map (λ*s*. *nth xs* (*s*+(*nth* (*nfilter P xs 0*) *k*)))
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)

using *1 5* **by** (*simp add*: *add.commute*)
have *8*: *map* (λ*s*. *nth xs* (*s*+(*nth* (*nfilter P xs 0*) *k*)))
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) =
map (λ*s*. *nth xs s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*))

using *5 nfilter-map-shift-suffix* **by** *metis*
have *9*: *map* (λ*s*. *nth xs s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*)) =
map (λ*s*. *nth xs s*)
(*suffix k* (*nfilter P xs 0*))

by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nfilter-suffix*)
show *?thesis*
by (*simp add*: *3 4 6 7 8 9*)
qed

lemma *filter-nfilter-suffix-1*:
assumes ∃ *x* ∈ *set xs*. *P x*
k ≤ *intlen* (*filter P xs*)
shows (*suffix k* (*filter P xs*)) =
(*filter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*))
proof −
have *1*: ∃*x* ∈ *set xs* . *P x*
using *assms* **by** *auto*
have *2*: (*filter P xs*) = *map* (λ*s*. *nth xs s*) (*nfilter P xs 0*)

by (*simp add*: *1 nfilter-map-filter*)
have *3*: (*suffix k* (*filter P xs*)) =
(*suffix k* (*map* (λ*s*. *nth xs s*) (*nfilter P xs 0*)))
by (*simp add*: *2*)
have *4*: (*suffix k* (*map* (λ*s*. *nth xs s*) (*nfilter P xs 0*))) =
map (λ*s*. *nth xs s*) (*suffix k* (*nfilter P xs 0*))
by (*simp add*: *1 assms*(*2*) *map-suffix nfilter-intlen*)
have *5*: ∃ *x* ∈ *set*(*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*). *P x*
using *assms nth-set 2*
by (*metis add.right-neutral diff-zero interval-intlen-map interval-nth-suffix le0 nfilter-holds*)
have *6*: (*filter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)) =
map (λ*s*. *nth* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)

by (*simp add*: *5 nfilter-map-filter*)
have *7*: *map* (λ*s*. *nth* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) =
map (λ*s*. *nth xs* (*s*+(*nth* (*nfilter P xs 0*) *k*)))
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*)

using *1 5*
by (*metis add.right-neutral interval-nth-suffix interval-suffix-suffix le0*)
have *8*: *map* (λ*s*. *nth xs* (*s*+(*nth* (*nfilter P xs 0*) *k*)))
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) *0*) =
map (λ*s*. *nth xs s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*))

using *5 nfilter-map-shift-suffix* **by** *metis*
have *9*: *map* (λ*s*. *nth xs s*)
(*nfilter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*) (*nth* (*nfilter P xs 0*) *k*)) =
map (λ*s*. *nth xs s*)
(*suffix k* (*nfilter P xs 0*))

by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nfilter-suffix-1*)
show *?thesis*
by (*simp add*: *3 4 6 7 8 9*)
qed

lemma *filter-nfilter-suffix-intlen*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k ≤ *intlen* (*filter P xs*)
shows *intlen*(*suffix k* (*filter P xs*)) =
intlen(*filter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*))
using *assms*
by (*simp add*: *filter-nfilter-suffix*)

lemma *filter-nfilter-suffix-intlen-1*:
assumes ∃ *x* ∈ *set xs*. *P x*
k ≤ *intlen* (*filter P xs*)
shows *intlen*(*suffix k* (*filter P xs*)) =

intlen(*filter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*))
using *assms*
by (*simp add*: *filter-nfilter-suffix-1*)

lemma *filter-nfilter-suffix-nth*:
assumes *P* (*nth xs* ((*nth* (*nfilter P xs 0*) *k*)))
k $\leq$ *intlen* (*filter P xs*)
j$\leq$ *intlen*(*suffix k* (*filter P xs*))
shows *nth* (*suffix k* (*filter P xs*)) *j* =
nth (*filter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)) *j*
using *assms*
by (*simp add*: *filter-nfilter-suffix*)

lemma *filter-nfilter-suffix-nth-1*:
assumes $\exists$ *x* $\in$ *set xs*. *P x*
k $\leq$ *intlen* (*filter P xs*)
j$\leq$ *intlen*(*suffix k* (*filter P xs*))
shows *nth* (*suffix k* (*filter P xs*)) *j* =
nth (*filter P* (*suffix* ((*nth* (*nfilter P xs 0*) *k*)) *xs*)) *j*
using *assms*
by (*simp add*: *filter-nfilter-suffix-1*)

lemma *nfilter-intlast*:
assumes *P* (*intlast* (*prefix k xs*))
k $\leq$ *intlen xs*
shows (*nth xs* ((*intlast* (*nfilter P* (*prefix k xs*) *n*))$-$*n*)) = (*intlast* (*prefix k xs*))
using *assms*
proof (*induction xs arbitrary*: *n k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*auto simp add*: *min.absorb1 split*:*nat.split*)
show $\bigwedge$*x2 x*.
($\bigwedge$*k n*. *P* (*nth xs k*) $\Longrightarrow$
k $\leq$ *intlen xs* $\Longrightarrow$
nth xs (*nth* (*nfilter P* (*prefix k xs*) *n*) (*intlen* (*nfilter P* (*prefix k xs*) *n*)) $-$ *n*) =
nth xs k) $\Longrightarrow$
P (*nth xs x2*) $\Longrightarrow$
x $\in$ *set* (*prefix x2 xs*) $\Longrightarrow$
P x $\Longrightarrow$
k = *Suc x2* $\Longrightarrow$
x2 $\leq$ *intlen xs* $\Longrightarrow$
nth (*nfilter P* (*prefix x2 xs*) (*Suc n*)) (*intlen* (*nfilter P* (*prefix x2 xs*) (*Suc n*))) $\leq$ *n* $\Longrightarrow$
x1a = *nth xs x2*
by (*metis le-refl nfilter-nth-n-zero-a not-less-eq-eq*)
show $\bigwedge$*x2 x x2a*.
($\bigwedge$*k n*. *P* (*nth xs k*) $\Longrightarrow$
k $\leq$ *intlen xs* $\Longrightarrow$

$nth\ xs\ (nth\ (nfilter\ P\ (prefix\ k\ xs)\ n)\ (intlen\ (nfilter\ P\ (prefix\ k\ xs)\ n)) - n) =$
$nth\ xs\ k) \Longrightarrow$
$P\ (nth\ xs\ x2) \Longrightarrow$
$x \in set\ (prefix\ x2\ xs) \Longrightarrow$
$P\ x \Longrightarrow$
$k = Suc\ x2 \Longrightarrow$
$x2 \le intlen\ xs \Longrightarrow$
$nth\ (nfilter\ P\ (prefix\ x2\ xs)\ (Suc\ n))\ (intlen\ (nfilter\ P\ (prefix\ x2\ xs)\ (Suc\ n))) - n =$
$Suc\ x2a \Longrightarrow$
$nth\ xs\ x2a = nth\ xs\ x2$
by (*metis Suc-eq-plus1 add-diff-cancel-left′ diff-diff-left plus-1-eq-Suc*)
show $\bigwedge x2.$
$(\bigwedge k\ n.\ P\ (nth\ xs\ k) \Longrightarrow$
$k \le intlen\ xs \Longrightarrow$
$nth\ xs\ (nth\ (nfilter\ P\ (prefix\ k\ xs)\ n)\ (intlen\ (nfilter\ P\ (prefix\ k\ xs)\ n)) - n) =$
$nth\ xs\ k) \Longrightarrow$
$P\ (nth\ xs\ x2) \Longrightarrow$
$\forall x \in set\ (prefix\ x2\ xs).\ \neg\ P\ x \Longrightarrow$
$k = Suc\ x2 \Longrightarrow$
$x2 \le intlen\ xs \Longrightarrow$
$x1a = nth\ xs\ x2$
using *interval-intlast-prefix nth-set* **by** *fastforce*
qed
qed

lemma *nfilter-intfirst*:
assumes $P\ (intfirst\ (suffix\ k\ xs))$
$k \le intlen\ xs$
shows $intfirst\ (nfilter\ P\ (suffix\ k\ xs)\ n) = n$
using *assms*
proof (*induction xs arbitrary*: *k n*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
by (*auto split*:*nat.split*)
qed

lemma *filter-intlast*:
assumes $P\ (intlast\ (prefix\ n\ xs))$
$n \le intlen\ xs$
shows $intlast\ (filter\ P\ (prefix\ n\ xs)) = (intlast\ (prefix\ n\ xs))$
using *assms*
proof (*induction n arbitrary*: *xs*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*

proof (*cases xs*)
case (*St x1*)
then show *?thesis*
by *simp*
next
case (*Cons x21 x22*)
then show *?thesis*
proof *auto*
show $\bigwedge x.\ xs = x21 \odot x22 \Longrightarrow$
$x \in set\ (prefix\ n\ x22) \Longrightarrow$
$P\ x \Longrightarrow$
$nth\ (filter\ P\ (prefix\ n\ x22))\ (intlen\ (filter\ P\ (prefix\ n\ x22))) =$
$nth\ x22\ (min\ n\ (intlen\ x22))$
using *Suc.IH Suc.prems* **by** *auto*
show $xs = x21 \odot x22 \Longrightarrow$
$\forall x{\in}set\ (prefix\ n\ x22).\ \neg\ P\ x \Longrightarrow$
$x21 = nth\ x22\ (min\ n\ (intlen\ x22))$
using *Suc.prems nth-set* **by** *force*
qed
qed
qed

lemma *filter-intfirst*:
assumes $P\ (intfirst\ (suffix\ n\ xs))$
shows $intfirst\ (filter\ P\ (suffix\ n\ xs)) = (intfirst\ (suffix\ n\ xs))$
using *assms*
proof (*induction xs arbitrary*: *n*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case* **by** (*auto split*:*nat.split*)
qed

lemma *filter-nth-aa*:
assumes $(\exists\ x \in set\ xs.\ P\ x)$
$n \le intlen\ (filter\ P\ xs)$
shows $P\ (nth\ (filter\ P\ xs)\ n)$
using *assms set-filter nth-set* **by** *fastforce*

lemma *filter-length-zero-conv-a*:
assumes $(\exists x{\in}set\ xs.\ P\ x)$
$intlen\ (filter\ P\ xs) = 0$
shows $(\exists\ k \le intlen\ xs.\ P\ (nth\ xs\ k)\ \wedge$
$(\forall\ j \le intlen\ xs.\ j \ne k \longrightarrow \neg\ P\ (nth\ xs\ j)))$
proof −
have *1*: $P\ (nth\ xs\ \ (nth\ (nfilter\ P\ xs\ 0)\ 0))$
by (*metis nfilter-map-filter assms*(*1*) *assms*(*2*) *filter-nth-aa interval-nth-map*

le-numeral-extra(*3*))
have *2*: (*nth* (*nfilter P xs 0*) *0*) $\leq$ *intlen xs*
by (*metis assms*(*1*) *diff-zero interval-intlen-gr-zero nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *3*: ($\forall$ *j* $\leq$ *intlen xs*. *j* $\neq$ (*nth* (*nfilter P xs 0*) *0*) $\longrightarrow$ $\neg$ *P* (*nth xs j*))
using *assms nfilter-not-after*[*of xs P*] *nfilter-not-before*[*of xs P*]
by (*metis linorder-neqE-nat nfilter-intlen*)
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *filter-length-zero-conv-c*:
($\exists$ *k* $\leq$ *intlen xs*. *P* (*nth xs k*) $\wedge$
($\forall$ *j* $\leq$ *intlen xs*. *j* $\neq$ *k* $\longrightarrow$ $\neg$ *P* (*nth xs j*))) =
($\exists$ *k* $\leq$ *intlen xs*. *P* (*nth xs k*) $\wedge$
($\forall$ *j* $\leq$ *intlen xs*. *j*<*k* $\vee$ *k*<*j* $\longrightarrow$ $\neg$ *P* (*nth xs j*)))

using *antisym-conv3* **by** *auto*

lemma *filter-length-zero-conv-d*:
($\exists$ *k* $\leq$ *intlen xs*. *P* (*nth xs k*) $\wedge$
($\forall$ *j* $\leq$ *intlen xs*. *j*<*k* $\vee$ *k*<*j* $\longrightarrow$ $\neg$ *P* (*nth xs j*))) =
($\exists$ *k* $\leq$ *intlen xs*. *P* (*nth xs k*) $\wedge$
($\forall$ *j*. *j*<*k* $\longrightarrow$ $\neg$ *P* (*nth xs j*)) $\wedge$
($\forall$ *j* $\leq$ *intlen xs*. *k*<*j* $\longrightarrow$ $\neg$ *P* (*nth xs j*))
)

by *auto*

lemma *filter-length-zero-conv-b*:
assumes ($\exists$ *k* $\leq$ *intlen xs*. *P* (*nth xs k*) $\wedge$
($\forall$ *j* $\leq$ *intlen xs*. *j* $\neq$ *k* $\longrightarrow$ $\neg$ *P* (*nth xs j*)))
shows ($\exists$*x*$\in$*set xs*. *P x*) $\wedge$ *intlen* (*filter P xs*) =*0*
proof −
have *1*: ($\exists$*x*$\in$*set xs*. *P x*)
using *assms nth-set* **by** *auto*
obtain *k* **where** *2*: *k* $\leq$ *intlen xs* $\wedge$ *P* (*nth xs k*) $\wedge$
($\forall$ *j* $\leq$ *intlen xs*. *j* $\neq$ *k* $\longrightarrow$ $\neg$ *P* (*nth xs j*))
using *assms* **by** *auto*
have *3*: *intlen* (*filter P xs*) =*0*
using *1 2*
proof (*induct xs arbitrary*: *k*)
case (*St x*)
then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
proof (*cases k*)
show ($\bigwedge$*k*. $\exists$*a*$\in$*set xs*. *P a* $\Longrightarrow$
k $\leq$ *intlen xs* $\wedge$ *P* (*nth xs k*) $\wedge$ ($\forall$*j*$\leq$*intlen xs*. *j* $\neq$ *k* $\longrightarrow$ $\neg$ *P* (*nth xs j*)) $\Longrightarrow$

intlen (*filter P xs*) = *0*) ⟹
∃ *a*∈*set* (*x1a* ⊙ *xs*). *P a* ⟹
k ≤ *intlen* (*x1a* ⊙ *xs*) ∧ *P* (*nth* (*x1a* ⊙ *xs*) *k*) ∧
(∀ *j*≤*intlen* (*x1a* ⊙ *xs*). *j* ≠ *k* ⟶ ¬ *P* (*nth* (*x1a* ⊙ *xs*) *j*)) ⟹
k = *0* ⟹ *intlen* (*filter P* (*x1a* ⊙ *xs*)) = *0*
by (*metis Suc-le-mono filter-intlen-d interval-nth-Suc interval-nth-and-set intlen.simps*(*2*)
le0 not-less-eq-eq order-refl plus-1-eq-Suc)
show ⋀*nat*.
(⋀*k*. ∃ *a*∈*set xs*. *P a* ⟹
k ≤ *intlen xs* ∧ *P* (*nth xs k*) ∧
(∀ *j*≤*intlen xs*. *j* ≠ *k* ⟶ ¬ *P* (*nth xs j*)) ⟹
intlen (*filter P xs*) = *0*) ⟹
∃ *a*∈*set* (*x1a* ⊙ *xs*). *P a* ⟹
k ≤ *intlen* (*x1a* ⊙ *xs*) ∧ *P* (*nth* (*x1a* ⊙ *xs*) *k*) ∧
(∀ *j*≤*intlen* (*x1a* ⊙ *xs*). *j* ≠ *k* ⟶ ¬ *P* (*nth* (*x1a* ⊙ *xs*) *j*)) ⟹
k = *Suc nat* ⟹
intlen (*filter P* (*x1a* ⊙ *xs*)) = *0*
by *fastforce*
qed
qed
show *?thesis*
using *1 3* **by** *blast*
qed

lemma *filter-length-zero-conv*:
((∃ *x*∈*set xs*. *P x*) ∧ *intlen* (*filter P xs*) =*0*) =
(∃ *k* ≤ *intlen xs*. *P* (*nth xs k*) ∧
(∀ *j* ≤ *intlen xs*. *j* ≠ *k* ⟶ ¬ *P* (*nth xs j*)))
using *filter-length-zero-conv-a*[*of xs P*] *filter-length-zero-conv-b*[*of xs P*]
by *blast*

lemma *filter-length-zero-conv-1*:
((∃ *x*∈*set xs*. *P x*) ∧ *intlen* (*filter P xs*) =*0*) =
(∃ *k* ≤ *intlen xs*. *P* (*nth xs k*) ∧
(∀ *j* ≤ *intlen xs*. *j*<*k* ∨ *k*<*j* ⟶ ¬ *P* (*nth xs j*)))
proof −
have *1*: ((∃ *x*∈*set xs*. *P x*) ∧ *intlen* (*filter P xs*) =*0*) =
(∃ *k* ≤ *intlen xs*. *P* (*nth xs k*) ∧
(∀ *j* ≤ *intlen xs*. *j* ≠ *k* ⟶ ¬ *P* (*nth xs j*)))
by (*simp add*: *filter-length-zero-conv*)
have *2*: (∃ *k* ≤ *intlen xs*. *P* (*nth xs k*) ∧
(∀ *j* ≤ *intlen xs*. *j* ≠ *k* ⟶ ¬ *P* (*nth xs j*))) =
(∃ *k* ≤ *intlen xs*. *P* (*nth xs k*) ∧
(∀ *j* ≤ *intlen xs*. *j*<*k* ∨ *k*<*j* ⟶ ¬ *P* (*nth xs j*)))
by *fastforce*
show *?thesis* **by** (*simp add*: *1 2*)
qed

lemma *filter-length-zero-conv-2*:
((∃ *x*∈*set xs*. *P x*) ∧ *intlen* (*filter P xs*) =*0*) =

```
  (∃ k ≤ intlen xs. P (nth xs k) ∧
    (∀ j. j<k  ⟶ ¬ P (nth xs j)) ∧
    (∀ j ≤ intlen xs. k<j  ⟶ ¬ P (nth xs j))
  )
proof −
 have 1: ((∃x∈set xs. P x) ∧  intlen (filter P xs) =0) =
  (∃ k ≤ intlen xs. P (nth xs k) ∧
           (∀ j ≤ intlen xs. j<k ∨ k<j ⟶ ¬ P (nth xs j)))
  by (simp add: filter-length-zero-conv-1)
 have 2: (∃ k ≤ intlen xs. P (nth xs k) ∧
           (∀ j ≤ intlen xs. j<k ∨ k<j ⟶ ¬ P (nth xs j))) =
  (∃ k ≤ intlen xs. P (nth xs k) ∧
    (∀ j. j<k  ⟶ ¬ P (nth xs j)) ∧
    (∀ j ≤ intlen xs. k<j  ⟶ ¬ P (nth xs j))
  )
  using dual-order.strict-trans1 by auto
 from 1 2 show ?thesis   by auto
qed

lemma filter-suffixes-map-help-0-0:
assumes ∃ x ∈ set (x1a⊙xs). P x
        ¬ P x1a
shows   (filter P (suffix (nth(nfilter P (x1a⊙xs) 0) 0) (x1a⊙xs))) = (filter P xs)
using assms
by (metis filter.simps(2) filter-nfilter-suffix-1 interval-set-ConsD interval-suffix-zero le0)

lemma filter-suffixes-map-help-0:
assumes j ≤ nth(nfilter P xs 0) 0
        ∃ x ∈ set xs. P x
shows   (filter P (suffix (nth(nfilter P xs 0) 0) xs)) = (filter P  (suffix j xs))
using assms
proof (induct xs arbitrary:j)
case (St x)
then show ?case by simp
next
case (Cons x1a xs)
then show ?case
  proof (cases ∃a∈set xs. P a)
  case True
  then show ?thesis
    proof (cases P x1a)
    case True
    then show ?thesis
    by (metis Cons.prems(1) le-zero-eq nfilter-nth-cons)
    next
    case False
    then show ?thesis
      proof (cases j)
      case 0
      then show ?thesis
```

by (*metis Cons.prems*(*2*) *False filter.simps*(*2*) *True interval-suffix-zero filter-suffixes-map-help-0-0*)
next
case (*Suc nat*)
then show *?thesis*
proof (*auto split*: *nat.split*)
show $\bigwedge x.\ j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
$P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$x1a \odot filter\ P\ xs = filter\ P\ (suffix\ nat\ xs)$
using *False* **by** *blast*
show $j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
$P\ x1a \Longrightarrow$
$\forall x \in set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle x1a \rangle = filter\ P\ (suffix\ nat\ xs)$
using *False* **by** *blast*
show $\bigwedge x.\ j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$filter\ P\ xs = filter\ P\ (suffix\ nat\ xs)$
by (*metis Cons.prems*(*1*) *add-leD1 nfilter.simps*(*2*) *not-one-le-zero plus-1-eq-Suc*)
show $j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$\forall x \in set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle x1a \rangle = filter\ P\ (suffix\ nat\ xs)$
using *True* **by** *blast*
show $\bigwedge x2\ x.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$x1a \odot filter\ P\ xs = filter\ P\ (suffix\ nat\ xs)$
using *False* **by** *blast*
show $\bigwedge x2.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$P\ x1a \Longrightarrow$
$\forall x \in set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle x1a \rangle = filter\ P\ (suffix\ nat\ xs)$
using *False* **by** *blast*
show $\bigwedge x2\ x.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$

```
              ¬ P x1a ⟹
              x ∈ set xs ⟹
              P x ⟹
              filter P (suffix x2 xs) = filter P (suffix nat xs)
          by (metis Cons.hyps Cons.prems(1) One-nat-def Suc-le-mono add-diff-cancel-left' le0
             nfilter.simps(2) nfilter-nth-n-zero plus-1-eq-Suc)
        show ⋀x2.
              j = Suc nat ⟹
              nth (nfilter P xs (Suc 0)) 0 = Suc x2 ⟹
              ¬ P x1a ⟹
              ∀x∈set xs. ¬ P x ⟹
              ⟨x1a⟩ = filter P (suffix nat xs)
          using True by blast
      qed
    qed
   qed
  next
  case False
  then show ?thesis
  using Cons.prems(1) by auto
  qed
qed

lemma filter-suffixes-map-help-0-a:
assumes j ≤ nth(nfilter P (suffixes xs) 0) 0
       ∃ x ∈ set (suffixes xs). P x
shows  (filter P (suffixes (suffix (nth(nfilter P (suffixes xs) 0) 0) xs))) =
       (filter P (suffixes (suffix j xs)))
proof −
 have 1: (suffix (nth(nfilter P (suffixes xs) 0) 0) (suffixes xs)) =
         (suffixes (suffix (nth(nfilter P (suffixes xs) 0) 0) xs))
  by (metis assms(2) diff-zero interval-intlen-gr-zero nfilter-upper-bound
     ordered-cancel-comm-monoid-diff-class.add-diff-inverse suffix-suffixes)
 have 2: nth(nfilter P (suffixes xs) 0) 0 ≤ intlen (suffixes xs)
   by (metis add.left-neutral assms(2) interval-intlen-gr-zero nfilter-upper-bound)
 have 3: (suffix j (suffixes xs)) = (suffixes (suffix j xs))
   using 2 suffix-suffixes assms le-trans by blast
 show ?thesis
 using 1 3 assms filter-suffixes-map-help-0 by fastforce
qed

lemma filter-suffixes-map-help-1:
assumes j ≤ nth(nfilter P xs 0) 1
        0 < intlen(filter P xs)
         nth(nfilter P xs 0) 0 < j
       ∃ x ∈ set xs. P x
shows  (filter P (suffix (nth(nfilter P xs 0) 1) xs)) = (filter P  (suffix j xs))
using assms
proof (induct xs arbitrary:j)
case (St x)
```

then show *?case* **by** *simp*
next
case (*Cons x1a xs*)
then show *?case*
  proof (*cases* $\exists a \in set\ xs.\ P\ a$)
  case *True*
  then show *?thesis*
    proof (*cases P x1a*)
    case *True*
    then show *?thesis*
      proof (*cases j*)
      case *0*
      then show *?thesis*
      using *Cons.prems*(*3*) **by** *blast*
      next
      case (*Suc nat*)
      then show *?thesis*
        proof (*auto split*: *nat.split*)
        show $\bigwedge x.$
            $j = Suc\ nat \Longrightarrow$
            $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
            $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = 0 \Longrightarrow$
            $P\ x1a \Longrightarrow$
            $x \in set\ xs \Longrightarrow$
            $P\ x \Longrightarrow$
            $x1a \odot filter\ P\ xs = filter\ P\ (suffix\ nat\ xs)$
          by (*metis Cons.prems*(*1*) *Cons.prems*(*3*) *One-nat-def add-diff-cancel-left′ leD*
            nfilter-nth-cons plus-1-eq-Suc)
        show $j = Suc\ nat \Longrightarrow$
            $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
            $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = 0 \Longrightarrow$
            $P\ x1a \Longrightarrow$
            $\forall x \in set\ xs.\ \neg\ P\ x \Longrightarrow$
            $\langle x1a \rangle = filter\ P\ (suffix\ nat\ xs)$
          using *Cons.prems*(*2*) **by** *auto*
        show $\bigwedge x.$
            $j = Suc\ nat \Longrightarrow$
            $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
            $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = 0 \Longrightarrow$
            $\neg\ P\ x1a \Longrightarrow$
            $x \in set\ xs \Longrightarrow$
            $P\ x \Longrightarrow$
            $filter\ P\ xs = filter\ P\ (suffix\ nat\ xs)$
          using *True* **by** *blast*
        show $j = Suc\ nat \Longrightarrow$
            $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
            $nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = 0 \Longrightarrow$
            $\neg\ P\ x1a \Longrightarrow \forall x \in set\ xs.\ \neg\ P\ x \Longrightarrow$
            $\langle x1a \rangle = filter\ P\ (suffix\ nat\ xs)$
          using *True* **by** *blast*

show $\bigwedge x2\ x.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = Suc\ x2 \Longrightarrow$
$P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$x1a \odot filter\ P\ xs = filter\ P\ (suffix\ nat\ xs)$
by (*metis Cons.prems(1) Cons.prems(3) One-nat-def add-diff-cancel-left' leD nfilter-nth-cons plus-1-eq-Suc*)
show $\bigwedge x2.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = Suc\ x2 \Longrightarrow$
$P\ x1a \Longrightarrow$
$\forall x \in set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle x1a \rangle = filter\ P\ (suffix\ nat\ xs)$
using *Cons.prems(2)* **by** *auto*
show $\bigwedge x2\ x.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = Suc\ x2 \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$filter\ P\ (suffix\ x2\ xs) = filter\ P\ (suffix\ nat\ xs)$
using *True* **by** *blast*
show $\bigwedge x2.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = 0 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = Suc\ x2 \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$\forall x \in set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle x1a \rangle = filter\ P\ (suffix\ nat\ xs)$
using *True* **by** *blast*
show $\bigwedge x2\ x.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = 0 \Longrightarrow$
$P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$filter\ P\ (suffix\ x2\ xs) = filter\ P\ (suffix\ nat\ xs)$
by (*metis Cons.prems(1) One-nat-def Suc-le-mono add-diff-cancel-left' interval-intlen-gr-zero nfilter-nth-cons nfilter-nth-n-zero plus-1-eq-Suc filter-suffixes-map-help-0*)
show $\bigwedge x2.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = 0 \Longrightarrow$

$P\ x1a \Longrightarrow$
$\forall\, x{\in}set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle x1a\rangle = filter\ P\ (suffix\ nat\ xs)$
using *Cons.prems*(*2*) **by** *auto*
show $\bigwedge x2\ x.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = 0 \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$filter\ P\ xs = filter\ P\ (suffix\ nat\ xs)$
using *True* **by** *blast*
show $\bigwedge x2.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = 0 \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$\forall\, x{\in}set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle x1a\rangle = filter\ P\ (suffix\ nat\ xs)$
using *True* **by** *blast*
show $\bigwedge x2\ x2a\ x.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = Suc\ x2a \Longrightarrow$
$P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$filter\ P\ (suffix\ x2\ xs) = filter\ P\ (suffix\ nat\ xs)$
using *nfilter-nth-n-zero*[*of - P - Suc 0*] *nfilter-nth-cons*[*of P x1a xs*]
by (*metis Cons.prems*(*1*) *One-nat-def Suc-le-mono add-diff-cancel-left' add-leD1 interval-intlen-gr-zero not-one-le-zero plus-1-eq-Suc filter-suffixes-map-help-0*)
show $\bigwedge x2\ x2a.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = Suc\ x2a \Longrightarrow$
$P\ x1a \Longrightarrow$
$\forall\, x{\in}set\ xs.\ \neg\ P\ x \Longrightarrow$
$\langle x1a\rangle = filter\ P\ (suffix\ nat\ xs)$
using *Cons.prems*(*2*) **by** *auto*
show $\bigwedge x2\ x2a\ x.$
$j = Suc\ nat \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ 0 = Suc\ x2 \Longrightarrow$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ 0) = Suc\ x2a \Longrightarrow$
$\neg\ P\ x1a \Longrightarrow$
$x \in set\ xs \Longrightarrow$
$P\ x \Longrightarrow$
$filter\ P\ (suffix\ x2a\ xs) = filter\ P\ (suffix\ nat\ xs)$
using *True* **by** *blast*
show $\bigwedge x2\ x2a.$

```
          j = Suc nat ⟹
          nth (nfilter P xs (Suc 0)) 0 = Suc x2 ⟹
          nth (nfilter P xs (Suc 0)) (Suc 0) = Suc x2a ⟹
          ¬ P x1a ⟹
          ∀ x∈set xs. ¬ P x ⟹
          ⟨x1a⟩ = filter P (suffix nat xs)
      using True by blast
    qed
   qed
next
case False
then show ?thesis
  proof (auto split: nat.split)
   show ⋀x2 x.
        ¬ P x1a ⟹
        nth (nfilter P xs (Suc 0)) (Suc 0) = 0 ⟹
        j = Suc x2 ⟹
        x ∈ set xs ⟹
        P x ⟹
        filter P xs = filter P (suffix x2 xs)
     by (metis Cons.prems(2) Cons.prems(4) One-nat-def Suc-leI filter-intlen-c
         nfilter-intlen nfilter-lower-bound not-one-le-zero)
   show ⋀x2.
        ¬ P x1a ⟹
        nth (nfilter P xs (Suc 0)) (Suc 0) = 0 ⟹
        j = Suc x2 ⟹
        ∀ x∈set xs. ¬ P x ⟹
        ⟨x1a⟩ = filter P (suffix x2 xs)
     using True by blast
   show ⋀x2 x.
        ¬ P x1a ⟹
        nth (nfilter P xs (Suc 0)) (Suc 0) = Suc x2 ⟹
        j = 0 ⟹
        x ∈ set xs ⟹
        P x ⟹
        filter P (suffix x2 xs) = filter P xs
      using Cons.prems(3) by blast
    show ⋀x2 x2a x.
         ¬ P x1a ⟹
         nth (nfilter P xs (Suc 0)) (Suc 0) = Suc x2 ⟹
         j = Suc x2a ⟹
         x ∈ set xs ⟹
         P x ⟹
         filter P (suffix x2 xs) = filter P (suffix x2a xs)
       proof −
        fix x2
        fix x2a
        fix x
        assume a0: ¬ P x1a
        assume a1: nth (nfilter P xs (Suc 0)) (Suc 0) = Suc x2
```

```
        assume a2: j = Suc x2a
        assume a3: x ∈ set xs
        assume a4:  P x
         show  filter P (suffix x2 xs) = filter P (suffix x2a xs)
         proof -
          have 1: j>0
            using Cons.prems(3) gr-implies-not-zero by blast
          have 2: x2a = j−1
            by (simp add: a2)
          have 3: x2 = nth (nfilter P xs (Suc 0)) (Suc 0) − (Suc 0)
            by (simp add: a1)
          have 4: (nth (nfilter P xs (Suc 0)) (Suc 0) − (Suc 0)) = (nth (nfilter P xs 0) (Suc 0))
            by (metis Cons.prems(2) Cons.prems(4) Suc-leI True a0 filter-intlen-c
                nfilter-intlen nfilter-nth-n-zero)
          have 5: 0 < intlen (filter P xs)
           using Cons.prems(2) Cons.prems(4) a0 by auto
          have 6:  ∃ a∈set xs. P a
           using True by blast
          have 7: filter P (suffix x2 xs) = filter P (suffix (nth (nfilter P xs 0) (Suc 0)) xs)
            by (simp add: 3 4)
          have 8: filter P (suffix x2a xs) = filter P (suffix (j−1) xs)
             using 2 by blast
          have 9: j−1 ≤ nth (nfilter P xs 0) 1
            by (metis 2 3 4 Cons.prems(1) One-nat-def Suc-le-mono True a0 a1 a2
               nfilter-nth-cons)
          have 10: (nth (nfilter P xs (Suc 0)) 0)− (Suc 0) = (nth (nfilter P xs 0) 0)
              by (meson a3 a4 interval-intlen-gr-zero nfilter-nth-n-zero)
          have 101: (nth (nfilter P xs 0) 0)  < j − (Suc 0)
             using 10 2 a0 6 a2 Cons.prems nfilter-nth-cons[of P x1a xs - -]
             filter-nth-aa[of xs P j−1]
             nfilter-nth-n-zero[of xs P] nfilter-holds[of x1a⊙xs P]
              by simp-all
              (metis 2 One-nat-def diff-less-mono interval-intlen-gr-zero nfilter-lower-bound)
          have 11: filter P (suffix (nth (nfilter P xs 0) 1) xs) = filter P (suffix (j−1) xs)
            using 101 5 9 Cons.hyps True by simp
           show ?thesis using 3 4 5 6 Cons.hyps Cons.prems
           by (metis 11 2 One-nat-def)
         qed
        qed
   show ⋀x2 x2a.
          ¬ P x1a ⟹
          nth (nfilter P xs (Suc 0)) (Suc 0) = Suc x2 ⟹
           j = Suc x2a ⟹
           ∀ x∈set xs. ¬ P x ⟹
           ⟨x1a⟩ = filter P (suffix x2a xs)
     using True by blast
 qed
qed
next
case False
```

then show *?thesis*
using *Cons.prems*(*2*) **by** *auto*
qed
qed

lemma *filter-suffixes-map-help-j-a*:
assumes ($\bigwedge j\ i.\ j \leq nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i) \Longrightarrow$
$i < intlen\ (filter\ P\ xs) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ i < j \Longrightarrow$
$filter\ P\ (suffix\ (nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i))\ xs) = filter\ P\ (suffix\ j\ xs))$
$x2a \leq x2$
$i < Suc\ (intlen\ (filter\ P\ xs))$
$(case\ i\ of\ 0 \Rightarrow 0 \mid Suc\ k \Rightarrow nth\ (nfilter\ P\ xs\ (Suc\ 0))\ k) < Suc\ x2a$
$P\ x1a$
$x \in set\ xs$
$P\ x$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ i = Suc\ x2$
$j = Suc\ x2a$
shows $filter\ P\ (suffix\ x2\ xs) = filter\ P\ (suffix\ x2a\ xs)$
proof −
have *1*: $i=0 \longrightarrow filter\ P\ (suffix\ x2\ xs) = filter\ P\ (suffix\ x2a\ xs)$
using *assms*
by (*metis One-nat-def add-diff-cancel-left′ interval-intlen-gr-zero nfilter-nth-n-zero plus-1-eq-Suc filter-suffixes-map-help-0*)
have *2*: $i>0 \longrightarrow (nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (i-1)) - (Suc\ 0) = nth\ (nfilter\ P\ xs\ 0)\ (i-1)$
by (*metis One-nat-def Suc-leI add-leD1 assms*(*3*) *assms*(*6*) *assms*(*7*) *le-add-diff-inverse2 less-Suc-eq-le nfilter-intlen nfilter-nth-n-zero*)
have *21*: $i>0 \longrightarrow nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (i-1) < Suc\ x2a$
using *assms*(*4*)
by (*metis Nitpick.case-nat-unfold less-Suc0 not-less-eq*)
have *22*: $i>0 \longrightarrow nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (i-1) > 0$
by (*metis One-nat-def Suc-leI add-leD1 assms*(*3*) *assms*(*6*) *assms*(*7*) *gr-zeroI le-add-diff-inverse2 less-Suc-eq-le nfilter-intlen nfilter-lower-bound not-one-le-zero*)
have *23*: $i>0 \longrightarrow nth\ (nfilter\ P\ xs\ 0)\ (i-1) < x2a$
using *2 21 22* **by** *linarith*
have *24*: $i>0 \longrightarrow x2a \leq nth\ (nfilter\ P\ xs\ 0)\ i$
by (*metis One-nat-def add-diff-cancel-left′ assms*(*2*) *assms*(*3*) *assms*(*6*) *assms*(*7*) *assms*(*8*) *less-Suc-eq-le nfilter-intlen nfilter-nth-n-zero plus-1-eq-Suc*)
have *25*: $i>0 \longrightarrow i - 1 < intlen\ (filter\ P\ xs)$
using *assms*(*3*) **by** *linarith*
have *3*: $i>0 \longrightarrow filter\ P\ (suffix\ (nth\ (nfilter\ P\ xs\ 0)\ i)\ xs) = filter\ P\ (suffix\ x2a\ xs)$
by (*metis 23 24 25 One-nat-def Suc-pred assms*(*1*))
have *4*: $i>0 \longrightarrow (nth\ (nfilter\ P\ xs\ 0)\ i) = x2$
by (*metis 25 One-nat-def Suc-leI Suc-pred add-diff-cancel-left′ assms*(*6*) *assms*(*7*) *assms*(*8*) *nfilter-intlen nfilter-nth-n-zero plus-1-eq-Suc*)
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *filter-suffixes-map-help-j-b*:
assumes ($\bigwedge j\ i.\ j \leq nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i) \Longrightarrow$

$i < intlen\ (filter\ P\ xs) \Longrightarrow$
$nth\ (nfilter\ P\ xs\ 0)\ i < j \Longrightarrow$
$filter\ P\ (suffix\ (nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i))\ xs) = filter\ P\ (suffix\ j\ xs))$
$x2a \le x2$
$i < intlen\ (filter\ P\ xs)$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ i < Suc\ x2a$
$\neg\ P\ x1a$
$x \in set\ xs$
$P\ x$
$nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ i) = Suc\ x2$
$j = Suc\ x2a$
shows *filter P (suffix x2 xs) = filter P (suffix x2a xs)*
proof −
have *0*: $i{=}0 \longrightarrow (nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (Suc\ i)) - (Suc\ 0) = nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i)$
by (*metis Suc-leI assms(3) assms(6) assms(7) nfilter-intlen nfilter-nth-n-zero*)
have *00*: $i{=}0 \longrightarrow (nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i)) = x2$
using *0 assms(8)* **by** *linarith*
have *03*: $i{=}0 \longrightarrow\ nth\ (nfilter\ P\ xs\ 0)\ i < x2a$
by (*metis One-nat-def add-diff-cancel-left′ assms(4) assms(6) assms(7) interval-intlen-gr-zero less-Suc-eq-0-disj nfilter-lower-bound nfilter-nth-n-zero not-one-le-zero plus-1-eq-Suc*)
have *04*: $i{=}0 \longrightarrow i < intlen\ (filter\ P\ xs)$
using *assms(3)* **by** *blast*
have *05*: $i{=}0 \longrightarrow x2a \le nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i)$
using *00 assms(2)* **by** *blast*
have *1*: $i{=}0 \longrightarrow filter\ P\ (suffix\ x2\ xs) = filter\ P\ (suffix\ x2a\ xs)$
using *00 03 05 assms(1) assms(3)* **by** *blast*
have *2*: $i{>}0 \longrightarrow (nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (i)) - (Suc\ 0) = nth\ (nfilter\ P\ xs\ 0)\ (i)$
by (*metis assms(3) assms(6) assms(7) less-imp-le-nat nfilter-intlen nfilter-nth-n-zero*)
have *21*: $i{>}0 \longrightarrow\ nth\ (nfilter\ P\ xs\ (Suc\ 0))\ i < Suc\ x2a$
using *assms(4)* **by** *auto*
have *22*: $i{>}0 \longrightarrow nth\ (nfilter\ P\ xs\ (Suc\ 0))\ (i) > 0$
by (*metis assms(3) assms(6) assms(7) gr-zeroI idx-nfilter-gr less-imp-le-nat nfilter-intlen not-less0*)
have *23*: $i{>}0 \longrightarrow nth\ (nfilter\ P\ xs\ 0)\ (i) < x2a$
using *2 22 assms(4)* **by** *linarith*
have *24*: $i{>}0 \longrightarrow x2a \le nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i)$
using *assms* **by** (*metis One-nat-def Suc-leI add-diff-cancel-left′ nfilter-intlen nfilter-nth-n-zero plus-1-eq-Suc*)
have *25*: $i{>}0 \longrightarrow i\ < intlen\ (filter\ P\ xs)$
by (*simp add: assms(3)*)
have *3*: $i{>}0 \longrightarrow\ filter\ P\ (suffix\ (nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i))\ xs\ ) = filter\ P\ (suffix\ x2a\ xs)$
using *23 24 assms(1) assms(3)* **by** *blast*
have *4*: $i{>}0 \longrightarrow (nth\ (nfilter\ P\ xs\ 0)\ (Suc\ i)) = x2$
using *assms* **by** (*metis One-nat-def Suc-leI add-diff-cancel-left′ nfilter-intlen nfilter-nth-n-zero plus-1-eq-Suc*)
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *filter-suffixes-map-help-j*:
assumes $j \le nth(nfilter\ P\ xs\ 0)\ (Suc\ i)$

```
       i < intlen(filter P xs)
        nth(nfilter P xs 0) i < j
      ∃ x ∈ set xs. P x
shows  (filter P (suffix (nth(nfilter P xs 0) (Suc i)) xs)) = (filter P (suffix j xs))
using assms
proof (induct xs arbitrary:j i)
case (St x)
then show ?case by simp
next
case (Cons x1a xs)
then show ?case
  proof (cases (∃ a∈set xs. P a))
   show (⋀j i. j ≤ nth (nfilter P xs 0) (Suc i) ⟹
          i < intlen (filter P xs) ⟹
          nth (nfilter P xs 0) i < j ⟹
          ∃ a∈set xs. P a ⟹
          filter P (suffix (nth (nfilter P xs 0) (Suc i)) xs) = filter P (suffix j xs)) ⟹
        j ≤ nth (nfilter P (x1a ⊙ xs) 0) (Suc i) ⟹
        i < intlen (filter P (x1a ⊙ xs)) ⟹
        nth (nfilter P (x1a ⊙ xs) 0) i < j ⟹
        ∃ a∈set (x1a ⊙ xs). P a ⟹
        ∃ a∈set xs. P a ⟹
        filter P (suffix (nth (nfilter P (x1a ⊙ xs) 0) (Suc i)) (x1a ⊙ xs)) =
        filter P (suffix j (x1a ⊙ xs))
    proof (cases P x1a)
     show (⋀j i. j ≤ nth (nfilter P xs 0) (Suc i) ⟹
            i < intlen (filter P xs) ⟹
            nth (nfilter P xs 0) i < j ⟹
            ∃ a∈set xs. P a ⟹
            filter P (suffix (nth (nfilter P xs 0) (Suc i)) xs) = filter P (suffix j xs)) ⟹
          j ≤ nth (nfilter P (x1a ⊙ xs) 0) (Suc i) ⟹
          i < intlen (filter P (x1a ⊙ xs)) ⟹
          nth (nfilter P (x1a ⊙ xs) 0) i < j ⟹
          ∃ a∈set (x1a ⊙ xs). P a ⟹
          ∃ a∈set xs. P a ⟹
          P x1a ⟹
          filter P (suffix (nth (nfilter P (x1a ⊙ xs) 0) (Suc i)) (x1a ⊙ xs)) =
          filter P (suffix j (x1a ⊙ xs))
     using filter-suffixes-map-help-j-a by (auto split: nat.split) fastforce
    show (⋀j i. j ≤ nth (nfilter P xs 0) (Suc i) ⟹
             i < intlen (filter P xs) ⟹
             nth (nfilter P xs 0) i < j ⟹
             ∃ a∈set xs. P a ⟹
             filter P (suffix (nth (nfilter P xs 0) (Suc i)) xs) = filter P (suffix j xs)) ⟹
          j ≤ nth (nfilter P (x1a ⊙ xs) 0) (Suc i) ⟹
          i < intlen (filter P (x1a ⊙ xs)) ⟹
          nth (nfilter P (x1a ⊙ xs) 0) i < j ⟹
          ∃ a∈set (x1a ⊙ xs). P a ⟹
          ∃ a∈set xs. P a ⟹
          ¬ P x1a ⟹
```

filter P (*suffix* (*nth* (*nfilter P* (*x1a* ⊙ *xs*) *0*) (*Suc i*)) (*x1a* ⊙ *xs*)) =
filter P (*suffix j* (*x1a* ⊙ *xs*))
proof (*auto split*: *nat.split*)
show ⋀*x x2 x2a*.
(⋀*j i*. *j* ≤ *nth* (*nfilter P xs 0*) (*Suc i*) ⟹
i < *intlen* (*filter P xs*) ⟹
nth (*nfilter P xs 0*) *i* < *j* ⟹
filter P (*suffix* (*nth* (*nfilter P xs 0*) (*Suc i*)) *xs*) = *filter P* (*suffix j xs*)) ⟹
x2a ≤ *x2* ⟹
i < *intlen* (*filter P xs*) ⟹
nth (*nfilter P xs* (*Suc 0*)) *i* < *Suc x2a* ⟹
¬ *P x1a* ⟹
x ∈ *set xs* ⟹
P x ⟹
nth (*nfilter P xs* (*Suc 0*)) (*Suc i*) = *Suc x2* ⟹
j = *Suc x2a* ⟹
filter P (*suffix x2 xs*) = *filter P* (*suffix x2a xs*)
using *filter-suffixes-map-help-j-b*[*of P xs*]
by *blast*
qed
qed
show (⋀*j i*. *j* ≤ *nth* (*nfilter P xs 0*) (*Suc i*) ⟹
i < *intlen* (*filter P xs*) ⟹
nth (*nfilter P xs 0*) *i* < *j* ⟹
∃ *a*∈*set xs*. *P a* ⟹
filter P (*suffix* (*nth* (*nfilter P xs 0*) (*Suc i*)) *xs*) = *filter P* (*suffix j xs*)) ⟹
j ≤ *nth* (*nfilter P* (*x1a* ⊙ *xs*) *0*) (*Suc i*) ⟹
i < *intlen* (*filter P* (*x1a* ⊙ *xs*)) ⟹
nth (*nfilter P* (*x1a* ⊙ *xs*) *0*) *i* < *j* ⟹
∃ *a*∈*set* (*x1a* ⊙ *xs*). *P a* ⟹
¬ (∃ *a*∈*set xs*. *P a*) ⟹
filter P (*suffix* (*nth* (*nfilter P* (*x1a* ⊙ *xs*) *0*) (*Suc i*)) (*x1a* ⊙ *xs*)) =
filter P (*suffix j* (*x1a* ⊙ *xs*))
by *auto*
qed
qed

lemma *filter-suffixes-map-help-j-aa*:
assumes *j* ≤ *nth*(*nfilter P* (*suffixes xs*) *0*) (*Suc i*)
i < *intlen*(*filter P* (*suffixes xs*))
nth(*nfilter P* (*suffixes xs*) *0*) *i* < *j*
∃ *x* ∈ *set* (*suffixes xs*). *P x*
shows (*filter P* (*suffixes* (*suffix* (*nth*(*nfilter P* (*suffixes xs*) *0*) (*Suc i*)) *xs*))) =
(*filter P* (*suffixes* (*suffix j xs*)))
proof −
have *1*: *nth*(*nfilter P* (*suffixes xs*) *0*) (*Suc i*) ≤ *intlen* (*suffixes xs*)
by (*metis Suc-leI add-cancel-right-left assms*(*2*) *assms*(*4*) *nfilter-intlen nfilter-upper-bound*)
have *2*: (*suffixes* (*suffix* (*nth*(*nfilter P* (*suffixes xs*) *0*) (*Suc i*)) *xs*)) =
(*suffix* (*nth*(*nfilter P* (*suffixes xs*) *0*) (*Suc i*)) (*suffixes xs*))
using *1 suffix-suffixes* **by** *fastforce*

have *3*: (*suffixes* (*suffix j xs*)) = (*suffix j* (*suffixes xs*))
  using *1 assms*(*1*) *le-trans suffix-suffixes* **by** *fastforce*
show *?thesis*
by (*simp add*: *2 3 assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *filter-suffixes-map-help-j*)
qed

lemma *filter-suffixes-map*:
assumes (*Suc i*) $\leq$ *intlen* (*filter f* (*suffixes* σ))
  $\exists$ $x \in$ *set*(*suffixes* σ). *f x*
shows (*suffix* (*Suc i*) (*map* (λ*s. nth s 0*) (*filter f* (*suffixes* σ)))) =
  (*map* (λ*s. nth s 0*)
   (*filter f* (*suffixes* (*suffix* (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) *i*)) σ))))
proof −
have *1*: (*suffix* (*Suc i*) (*map* (λ*s. nth s 0*) (*filter f* (*suffixes* σ)))) =
  (*map* (λ*s. nth s 0*) (*suffix* (*Suc i*) (*filter f* (*suffixes* σ))))
by (*simp add*: *assms*(*1*) *map-suffix*)
have *2*: (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) *i*)) $\leq$ *intlen*(*suffixes* σ)
  by (*metis length-filter-le Suc-leD assms*(*1*) *assms*(*2*) *diff-is-0-eq diff-zero*
  filter-nfilter-suffix-1 interval-suffix-length not-less-eq-eq)
have *3*: (*filter f* (*suffixes* (*suffix* (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) *i*)) σ))) =
  (*filter f* (*suffix* (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) *i*)) (*suffixes* σ)))

  using *2 suffix-suffixes* **by** *fastforce*
have *4*: (*suffix* (*Suc i*) (*filter f* (*suffixes* σ))) =
  (*filter f* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) (*Suc i*)) (*suffixes* σ)))

  by (*simp add*: *assms*(*1*) *assms*(*2*) *filter-nfilter-suffix-1*)
have *5*: (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) *i*)) $\leq$ *nth*(*nfilter f* (*suffixes* σ) *0*) (*Suc i*)
  by (*simp add*: *Suc-leI Suc-le-lessD assms*(*1*) *assms*(*2*) *idx-nfilter-mono nfilter-intlen*)
have *6*: *i* < *intlen*(*filter f* (*suffixes* σ))
  using *Suc-le-eq assms*(*1*) **by** *blast*
have *7*: *nth* (*nfilter f* (*suffixes* σ) *0*) *i* < (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) *i*))
  by *simp*
have *8*: (*filter f* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) (*Suc i*)) (*suffixes* σ))) =
  (*filter f* (*suffix* (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) *i*)) (*suffixes* σ)))

using *5 6 7*
filter-suffixes-map-help-j[*of* (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) *i*)) *f suffixes* σ *i*]
  assms(*2*) **by** *blast*
show *?thesis*
by (*simp add*: *1 3 4 8*)
qed

lemma *sfxfilter-suffix-intlen*:
assumes *k*$\leq$*intlen* σ
  $\exists x \in$ *set* (*suffixes*(*suffix k* σ)). *f x*
shows *intlen* (*filter f* (*suffixes* (*suffix k* σ))) $\leq$ *intlen* (*filter f* (*suffixes* σ))
using *assms*
proof (*induct k arbitrary*: σ)
case *0*

then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
proof (*cases σ*)
case (*St x1*)
then show *?thesis*
by *simp*
next
case (*Cons x21 x22*)
then show *?thesis*
proof *auto*
show $\bigwedge y.$
$\sigma = x21 \odot x22 \Longrightarrow$
$f\ (x21 \odot x22) \Longrightarrow$
$osfx\ y\ x22 \Longrightarrow$
$f\ y \Longrightarrow$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \leq Suc\ (intlen\ (filter\ f\ (suffixes\ x22)))$
by (*metis length-filter-le Suc.hyps Suc.prems*(*2*) *interval-suffix-length-code interval-suffix-suc intlen-suffixes le-0-eq le-SucI less-imp-le-nat zero-less-Suc*)
show $\sigma = x21 \odot x22 \Longrightarrow$
$f\ (x21 \odot x22) \Longrightarrow$
$\forall x \in set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) = 0$
using *Suc.prems osfx-suffix* **by** *fastforce*
show $\bigwedge y.\ \sigma = x21 \odot x22 \Longrightarrow$
$\neg\ f\ (x21 \odot x22) \Longrightarrow$
$osfx\ y\ x22 \Longrightarrow$
$f\ y \Longrightarrow$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \leq intlen\ (filter\ f\ (suffixes\ x22))$
by (*metis length-filter-le Suc.hyps Suc.prems*(*2*) *interval-intlen-gr-zero interval-suffix-length-code interval-suffix-suc intlen-suffixes le-0-eq*)
show $\sigma = x21 \odot x22 \Longrightarrow$
$\neg\ f\ (x21 \odot x22) \Longrightarrow$
$\forall x \in set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) = 0$
using *Suc.prems osfx-suffix* **by** *fastforce*
qed
qed
qed

lemma *sfxfilter-suffix-nth*:
assumes $k \leq intlen\ \sigma$
$\exists x \in set\ (suffixes(suffix\ k\ \sigma)).\ f\ x$
$j \leq intlen\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma)))$
shows $(nth\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma)))\ j) =$
$(nth\ (suffix\ (intlen(filter\ f\ (suffixes\ \sigma)) - intlen((filter\ f\ (suffixes\ (suffix\ k\ \sigma)))))$
$(filter\ f\ (suffixes\ \sigma)))\ j)$
using *assms*
proof (*induct k arbitrary*: *σ*)

case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
  proof (*cases σ*)
  case (*St x1*)
  then show *?thesis*
  by *simp*
  next
  case (*Cons x21 x22*)
  then show *?thesis*
    proof (*auto split*: *nat.split*)
     show $\bigwedge y.$
        $\sigma = x21 \odot x22 \Longrightarrow$
        $j = 0 \Longrightarrow$
        $Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) \le$
        $intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \Longrightarrow$
        $f\ (x21 \odot x22) \Longrightarrow$
        $osfx\ y\ x22 \Longrightarrow$
        $f\ y \Longrightarrow$
        $nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ 0 = x21 \odot x22$
        using *Suc.prems not-less-eq-eq sfxfilter-suffix-intlen* **by** *fastforce*
     show $\sigma = x21 \odot x22 \Longrightarrow$
       $j = 0 \Longrightarrow$
       $Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) \le$
       $intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \Longrightarrow$
       $f\ (x21 \odot x22) \Longrightarrow$
       $\forall x{\in}set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
       $nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ 0 = x21 \odot x22$
       using *Suc.prems osfx-order.order.trans* **by** *fastforce*
   show $\bigwedge y.$
      $\sigma = x21 \odot x22 \Longrightarrow$
      $j = 0 \Longrightarrow$
      $Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) \le$
      $intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \Longrightarrow$
      $\neg\ f\ (x21 \odot x22) \Longrightarrow$
      $osfx\ y\ x22 \Longrightarrow$
      $f\ y \Longrightarrow$
      $nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ 0 = nth\ (filter\ f\ (suffixes\ x22))\ 0$
     using *Suc.hyps Suc.prems* **by** *auto*
    show $\sigma = x21 \odot x22 \Longrightarrow$
      $j = 0 \Longrightarrow$
      $Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) \le$
      $intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \Longrightarrow$
      $\neg\ f\ (x21 \odot x22) \Longrightarrow$
      $\forall x{\in}set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
      $nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ 0 = x21 \odot x22$
      using *Suc.prems osfx-order.order.trans* **by** *fastforce*
    show $\bigwedge x2\ y.$

$\sigma = x21 \odot x22 \Longrightarrow$
$j = 0 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) =$
$Suc\ x2 \Longrightarrow$
$f\ (x21 \odot x22) \Longrightarrow$
$osfx\ y\ x22 \Longrightarrow$
$f\ y \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ 0 = nth\ (filter\ f\ (suffixes\ x22))\ x2$
using *Suc.hyps Suc.prems*
by (*metis Suc-diff-diff Suc-le-mono add.right-neutral diff-zero interval-nth-suffix interval-suffix-suc intlen.simps*(2) *le0 plus-1-eq-Suc*)
show $\bigwedge x2.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = 0 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) =$
$Suc\ x2 \Longrightarrow$
$f\ (x21 \odot x22) \Longrightarrow$
$\forall x \in set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ 0 = x21 \odot x22$
using *Suc.prems osfx-suffix* **by** *fastforce*
show $\bigwedge x2\ y.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = 0 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) =$
$Suc\ x2 \Longrightarrow$
$\neg\ f\ (x21 \odot x22) \Longrightarrow$
$osfx\ y\ x22 \Longrightarrow$
$f\ y \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ 0 =$
$nth\ (filter\ f\ (suffixes\ x22))$
$(intlen\ (filter\ f\ (suffixes\ x22)) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))))$
using *Suc.hyps Suc.prems* **by** *auto*
show $\bigwedge x2.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = 0 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) =$
$Suc\ x2 \Longrightarrow$
$\neg\ f\ (x21 \odot x22) \Longrightarrow$
$\forall x \in set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ 0 = x21 \odot x22$
using *Suc.prems osfx-suffix* **by** *fastforce*
show $\bigwedge x2\ y.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = Suc\ x2 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) \leq$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \Longrightarrow$
$f\ (x21 \odot x22) \Longrightarrow$
$osfx\ y\ x22 \Longrightarrow$
$f\ y \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ (Suc\ x2) = nth\ (filter\ f\ (suffixes\ x22))\ x2$

using *Suc.prems not-less-eq-eq sfxfilter-suffix-intlen* **by** *fastforce*
show $\bigwedge x2.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = Suc\ x2 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) \leq$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \Longrightarrow$
$f\ (x21 \odot x22) \Longrightarrow$
$\forall x{\in}set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ (Suc\ x2) = x21 \odot x22$
using *Suc.prems osfx-order.order.trans* **by** *fastforce*
show $\bigwedge x2\ y.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = Suc\ x2 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) \leq$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \Longrightarrow$
$\neg\ f\ (x21 \odot x22) \Longrightarrow$
$osfx\ y\ x22 \Longrightarrow$
$f\ y \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ (Suc\ x2) =$
$nth\ (filter\ f\ (suffixes\ x22))\ (Suc\ x2)$
using *Suc.prems not-less-eq-eq sfxfilter-suffix-intlen* **by** *fastforce*
show $\bigwedge x2.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = Suc\ x2 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) \leq$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) \Longrightarrow$
$\neg\ f\ (x21 \odot x22) \Longrightarrow \forall x{\in}set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ (Suc\ x2) = x21 \odot x22$
using *Suc.prems osfx-order.order.trans* **by** *fastforce*
show $\bigwedge x2\ x2a\ y.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = Suc\ x2 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) =$
$Suc\ x2a \Longrightarrow$
$f\ (x21 \odot x22) \Longrightarrow$
$osfx\ y\ x22 \Longrightarrow$
$f\ y \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ (Suc\ x2) =$
$nth\ (suffix\ x2a\ (filter\ f\ (suffixes\ x22)))\ (Suc\ x2)$
using *Suc.hyps Suc.prems*
by (*metis Suc-diff-diff diff-zero interval-suffix-suc intlen.simps*(*2*) *not-less-eq-eq plus-1-eq-Suc*)
show $\bigwedge x2\ x2a.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = Suc\ x2 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) =$
$Suc\ x2a \Longrightarrow$
$f\ (x21 \odot x22) \Longrightarrow$
$\forall x{\in}set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ (Suc\ x2) = x21 \odot x22$

using *Suc.prems osfx-suffix* **by** *fastforce*
show $\bigwedge x2\ x2a\ y.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = Suc\ x2 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) =$
$Suc\ x2a \Longrightarrow$
$\neg\ f\ (x21 \odot x22) \Longrightarrow$
$osfx\ y\ x22 \Longrightarrow$
$f\ y \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ (Suc\ x2) =$
$nth\ (suffix\ (intlen\ (filter\ f\ (suffixes\ x22)) -$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))))$
$(filter\ f\ (suffixes\ x22)))$
$(Suc\ x2)$
using *Suc.hyps Suc.prems* **by** *auto*
show $\bigwedge x2\ x2a.$
$\sigma = x21 \odot x22 \Longrightarrow$
$j = Suc\ x2 \Longrightarrow$
$Suc\ (intlen\ (filter\ f\ (suffixes\ x22))) -$
$intlen\ (filter\ f\ (suffixes\ (suffix\ k\ x22))) = Suc\ x2a \Longrightarrow$
$\neg\ f\ (x21 \odot x22) \Longrightarrow$
$\forall x \in set\ (suffixes\ x22).\ \neg\ f\ x \Longrightarrow$
$nth\ (filter\ f\ (suffixes\ (suffix\ k\ x22)))\ (Suc\ x2) = x21 \odot x22$
using *Suc.prems osfx-suffix* **by** *fastforce*
qed
qed
qed

lemma *sfxfilter-suffix-suffix*:
assumes $k \le intlen\ \sigma$
$\exists x \in set\ (suffixes(suffix\ k\ \sigma)).\ f\ x$
shows $(filter\ f\ (suffixes\ (suffix\ k\ \sigma))) =$
$(suffix\ (intlen(filter\ f\ (suffixes\ \sigma)) - intlen((filter\ f\ (suffixes\ (suffix\ k\ \sigma)))))$
$(filter\ f\ (suffixes\ \sigma)))$
by (*simp add*: *assms*(1) *assms*(2) *interval-eq-nth-eq sfxfilter-suffix-intlen sfxfilter-suffix-nth*)

lemma *sfxfilter-suffix-suffix-a*:
assumes $jj \le intlen\ (filter\ f\ (suffixes\ xs))$
$\exists x \in set\ (suffixes\ xs).\ f\ x$
shows $(suffix\ jj\ (filter\ f\ (suffixes\ xs))) =$
$filter\ f\ (suffixes\ (suffix\ ((intlen\ xs) - intlen\ (nth\ (filter\ f\ (suffixes\ xs))\ jj))\ \ xs))$

proof −
have *1*: $(suffix\ jj\ (filter\ f\ (suffixes\ xs))) =$
$filter\ f\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ xs)\ 0)\ jj)\ (suffixes\ xs))$
using *filter-nfilter-suffix-1*
by (*simp add*: *filter-nfilter-suffix-1 assms*(1) *assms*(2))
have *2*: $(nth\ (nfilter\ f\ (suffixes\ xs)\ 0)\ jj) \le intlen\ (suffixes\ xs)$
by (*metis add-cancel-right-left assms*(1) *assms*(2) *nfilter-intlen nfilter-upper-bound*)
have *3*: $(suffix\ (nth\ (nfilter\ f\ (suffixes\ xs)\ 0)\ jj)\ (suffixes\ xs)) =$

(*suffixes* (*suffix* (*nth* (*nfilter f* (*suffixes xs*) *0*) *jj*) *xs*))
using *suffix-suffixes*[*of* (*nth* (*nfilter f* (*suffixes xs*) *0*) *jj*) *xs*] *2* **by** *blast*
have *4*: (*nth* (*filter f* (*suffixes xs*)) *jj*) = *nth* (*suffixes xs*) (*nth* (*nfilter f* (*suffixes xs*) *0*) *jj*)
by (*metis nfilter-map-filter assms*(*2*) *interval-nth-map*)
have *5*: *intlen*(*suffixes xs*) ≤ *intlen* (*xs*)
by *simp*
have *6*: (*nth* (*nfilter f* (*suffixes xs*) *0*) *jj*) ≤ *intlen* (*xs*)
using *2 5 le-trans* **by** *blast*
have *7*: *nth* (*suffixes xs*) (*nth* (*nfilter f* (*suffixes xs*) *0*) *jj*) =
(*suffix* (*nth* (*nfilter f* (*suffixes xs*) *0*) *jj*) *xs*)
by (*simp add*: *6 nth-suffixes*)
have *8*: (*nth* (*nfilter f* (*suffixes xs*) *0*) *jj*) =
((*intlen xs*) − *intlen* (*nth* (*filter f* (*suffixes xs*)) *jj*))
by (*simp add*: *4 6 7*)
show *?thesis*
using *1 3 8* **by** *auto*
qed

lemma *sfx-suffix-upperbound*:
(∀ *jj*<(*intlen*(*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*)))).
((*intlen σ*) − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*)) < *k*
)
proof
fix *jj*
show *jj* < *intlen* (*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*))) ⟶
intlen σ − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*) < *k*
proof (*induct k arbitrary*: *σ jj*)
show ⋀*σ jj*.
jj < *intlen* (*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix 0 σ*))) ⟶
intlen σ − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*) < *0*
by *simp*
show ⋀*k σ jj*.
(⋀*σ jj*.
jj < *intlen* (*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*))) ⟶
intlen σ − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*) < *k*) ⟹
jj < *intlen* (*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix* (*Suc k*) *σ*))) ⟶
intlen σ − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*) < *Suc k*
proof (*case-tac σ*)
show ⋀*k σ jj x1*.
(⋀*σ jj*.
jj < *intlen* (*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*))) ⟶
intlen σ − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*) < *k*) ⟹
σ = ⟨*x1*⟩ ⟹
jj < *intlen* (*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix* (*Suc k*) *σ*))) ⟶
intlen σ − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*) < *Suc k*
by *simp*
show ⋀*k σ jj x21 x22*.
(⋀*σ jj*.
jj < *intlen* (*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*))) ⟶
intlen σ − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*) < *k*) ⟹

$\sigma = x21 \odot x22 \Longrightarrow$
$jj < intlen\ (filter\ f\ (suffixes\ \sigma)) - intlen\ (filter\ f\ (suffixes\ (suffix\ (Suc\ k)\ \sigma))) \longrightarrow$
$intlen\ \sigma - intlen\ (nth\ (filter\ f\ (suffixes\ \sigma))\ jj) < Suc\ k$
proof (*case-tac jj*)
show $\bigwedge k\ \sigma\ jj\ x21\ x22.$
$(\bigwedge \sigma\ jj.$
$jj < intlen\ (filter\ f\ (suffixes\ \sigma)) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma))) \longrightarrow$
$intlen\ \sigma - intlen\ (nth\ (filter\ f\ (suffixes\ \sigma))\ jj) < k) \Longrightarrow$
$\sigma = x21 \odot x22 \Longrightarrow$
$jj = 0 \Longrightarrow$
$jj < intlen\ (filter\ f\ (suffixes\ \sigma)) - intlen\ (filter\ f\ (suffixes\ (suffix\ (Suc\ k)\ \sigma))) \longrightarrow$
$intlen\ \sigma - intlen\ (nth\ (filter\ f\ (suffixes\ \sigma))\ jj) < Suc\ k$
by *auto*
(*metis Suc-diff-le Suc-mono in-set-suffixes interval-intlen-gr-zero*
sfxfilter-nth-bound zero-less-diff)
show $\bigwedge k\ \sigma\ jj\ x21\ x22\ nat.$
$(\bigwedge \sigma\ jj.$
$jj < intlen\ (filter\ f\ (suffixes\ \sigma)) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma))) \longrightarrow$
$intlen\ \sigma - intlen\ (nth\ (filter\ f\ (suffixes\ \sigma))\ jj) < k) \Longrightarrow$
$\sigma = x21 \odot x22 \Longrightarrow$
$jj = Suc\ nat \Longrightarrow$
$jj < intlen\ (filter\ f\ (suffixes\ \sigma)) - intlen\ (filter\ f\ (suffixes\ (suffix\ (Suc\ k)\ \sigma))) \longrightarrow$
$intlen\ \sigma - intlen\ (nth\ (filter\ f\ (suffixes\ \sigma))\ jj) < Suc\ k$
using *Suc-diff-le less-Suc-eq-0-disj* **by** *auto*
qed
qed
qed
qed

end

21 Until and Since operator

theory *UntilSince*
imports *Semantics Fuse Theorems TimeReversal*
begin

This theory introduces the weak and strong versions of the until and since operators. The theorems from [11] are proven in a mostly deductive style.

21.1 Definitions

definition *until-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where $until\text{-}d\ F\ G \equiv \lambda s.\ (\ (\exists\ k \le intlen\ s.\ ((suffix\ k\ s\ ) \models G) \wedge$
$(\forall j.\ j < k\ \longrightarrow\ ((suffix\ j\ s\ ) \models F)\ ))\ )$

definition *suntil-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where $suntil\text{-}d\ F\ G \equiv \lambda s.\ (\ intlen\ s > 0 \wedge (\exists\ k.\ 0<k \wedge k \le intlen\ s \wedge ((suffix\ k\ s\ ) \models G) \wedge$

$(\forall j.\ 0{<}j \wedge j < k \longrightarrow ((\textit{suffix } j\ s\ ) \models F)\ ))\ )$

definition *since-d* :: $('a :: world)$ *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *since-d F G* $\equiv \lambda s.\ (\ (\exists\ k \leq \textit{intlen } s.\ (\ (\textit{prefix } k\ s) \models G) \wedge$
$(\forall\ j.\ k{<}\ j\ \wedge j \leq \textit{intlen } s \longrightarrow ((\textit{prefix } j\ s) \models F)\ ))\ )$

definition *ssince-d* :: $('a :: world)$ *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *ssince-d F G* $\equiv \lambda s.\ (\ \textit{intlen } s > 0 \wedge (\exists\ k\ .\ \ k < \textit{intlen } s \wedge (\ (\textit{prefix } k\ s) \models G) \wedge$
$(\forall\ j.\ k{<}\ j \wedge j < \textit{intlen } s \longrightarrow ((\textit{prefix } j\ s) \models F)\ ))\ )$

syntax
-until-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\mathcal{U}$ -) [*84*,*84*] *83*)
-since-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\mathcal{S}$ -) [*84*,*84*] *83*)
-suntil-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\mathcal{U}^s$ -) [*84*,*84*] *83*)
-ssince-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\mathcal{S}^s$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-until-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *until* -) [*84*,*84*] *83*)
-since-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *since* -) [*84*,*84*] *83*)
-suntil-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *suntil* -) [*84*,*84*] *83*)
-ssince-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- *ssince* -) [*84*,*84*] *83*)

translations
-until-d $\rightleftharpoons$ *CONST until-d*
-since-d $\rightleftharpoons$ *CONST since-d*
-suntil-d $\rightleftharpoons$ *CONST suntil-d*
-ssince-d $\rightleftharpoons$ *CONST ssince-d*

definition *wait-d* :: $('a :: world)$ *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *wait-d F G* $\equiv$ *LIFT*$(\Box\ F \vee F\ \mathcal{U}\ G)$

definition *pwait-d* :: $('a :: world)$ *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *pwait-d F G* $\equiv$ *LIFT*$(\textit{bi } F \vee F\ \mathcal{S}\ G)$

definition *release-d* :: $('a :: world)$ *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *release-d F G* $\equiv$ *LIFT*$(\neg((\neg\ F)\ \mathcal{U}\ (\neg\ G)))$

definition *prelease-d* :: $('a :: world)$ *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *prelease-d F G* $\equiv$ *LIFT*$(\neg((\neg\ F)\ \mathcal{S}\ (\neg\ G)))$

syntax
-wait-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\mathcal{W}$ -) [*84*,*84*] *83*)
-pwait-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\mathcal{PW}$ -) [*84*,*84*] *83*)
-release-d :: [*lift*,*lift*] $\Rightarrow$ *lift* ((- $\mathcal{R}$ -) [*84*,*84*] *83*)

-prelease-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{PR}$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-wait-d :: [*lift*,*lift*] ⇒ *lift* ((- *wait* -) [*84*,*84*] *83*)
-pwait-d :: [*lift*,*lift*] ⇒ *lift* ((- *pwait* -) [*84*,*84*] *83*)
-release-d :: [*lift*,*lift*] ⇒ *lift* ((- *release* -) [*84*,*84*] *83*)
-prelease-d :: [*lift*,*lift*] ⇒ *lift* ((- *prelease* -) [*84*,*84*] *83*)

translations
-wait-d ⇌ *CONST wait-d*
-pwait-d ⇌ *CONST pwait-d*
-release-d ⇌ *CONST release-d*
-prelease-d ⇌ *CONST prelease-d*

definition *srelease-d* :: ('*a* :: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *srelease-d F G* ≡ *LIFT*(¬((¬ *F*) $\mathcal{W}$ (¬ *G*)))

definition *psrelease-d* :: ('*a* :: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *psrelease-d F G* ≡ *LIFT*(¬((¬ *F*) $\mathcal{PW}$ (¬ *G*)))

syntax
-srelease-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{M}$ -) [*84*,*84*] *83*)
-psrelease-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{PM}$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-srelease-d :: [*lift*,*lift*] ⇒ *lift* ((- *srelease* -) [*84*,*84*] *83*)
-psrelease-d :: [*lift*,*lift*] ⇒ *lift* ((- *psrelease* -) [*84*,*84*] *83*)

translations
-srelease-d ⇌ *CONST srelease-d*
-psrelease-d ⇌ *CONST psrelease-d*

21.2 Semantic Lemmas

lemma *SUntilNextUntilsema*:
assumes $\sigma \models f\ \mathcal{U}^s\ g$
shows $\sigma \models \bigcirc\ (f\ \mathcal{U}\ g)$
proof −
have *1*: $\sigma \models f\ \mathcal{U}^s\ g$
using *assms* **by** *auto*
have *2*: $0 < intlen\ \sigma\ \wedge$
$(\exists\, k{>}0.\ k \le intlen\ \sigma \wedge g\ (suffix\ k\ \sigma) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma)))$
using *1* **by** (*simp add*: *suntil-d-def*)
have *3*: $(\exists\, k{>}0.\ k \le intlen\ \sigma \wedge g\ (suffix\ k\ \sigma) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma)))$
using *2* **by** *auto*

obtain *k* **where** *4*: $(Suc\ k)>0 \wedge (Suc\ k) \leq intlen\ \sigma \wedge g\ (suffix\ (Suc\ k)\ \sigma) \wedge$
$(\forall j.\ 0 < j \wedge j < (Suc\ k) \longrightarrow f\ (suffix\ j\ \sigma))$
using *3* **by** (*metis Suc-pred*)
have *5*: $k \leq intlen\ (suffix\ (Suc\ 0)\ \sigma)$
using *4* **by** *auto*
have *6*: $g\ (suffix\ (Suc\ k)\ \sigma)$
using *4* **by** *auto*
have *7*: $(\forall j<k.\ f\ (suffix\ (Suc\ j)\ \sigma))$
using *4* **by** *blast*
have *8*: $(\exists k \leq intlen\ (suffix\ (Suc\ 0)\ \sigma).\ g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j<k.\ f\ (suffix\ (Suc\ j)\ \sigma)))$
using *4 5* **by** *blast*
have *9*: $0 < intlen\ \sigma \wedge$
$(\exists k \leq intlen\ (suffix\ (Suc\ 0)\ \sigma).\ g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j<k.\ f\ (suffix\ (Suc\ j)\ \sigma)))$
using *2 8* **by** *blast*
from *9* **show** *?thesis* **by** (*auto simp add*: *next-defs until-d-def*)
qed

lemma *SUntilNextUntilsemb*:
assumes $\sigma \models \bigcirc\ (\ f\ \mathcal{U}\ g)$
shows $\sigma \models\ f\ \mathcal{U}^s\ g$
proof −
have *1*: $0 < intlen\ \sigma \wedge$
$(\exists k \leq intlen\ (suffix\ (Suc\ 0)\ \sigma).\ g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j<k.\ f\ (suffix\ (Suc\ j)\ \sigma)))$
using *assms* **by** (*auto simp add*: *next-defs until-d-def*)
have *2*: $(\exists k \leq intlen\ (suffix\ (Suc\ 0)\ \sigma).\ g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j<k.\ f\ (suffix\ (Suc\ j)\ \sigma)))$
using *1* **by** *auto*
obtain *k* **where** *3*: $k \leq intlen\ (suffix\ (Suc\ 0)\ \sigma) \wedge g\ (suffix\ (Suc\ k)\ \sigma) \wedge$
$(\forall j<k.\ f\ (suffix\ (Suc\ j)\ \sigma))$
using *2* **by** *auto*
have *4*: $(Suc\ k)>0$
by *simp*
have *5*: $g\ (suffix\ (Suc\ k)\ \sigma)$
using *3* **by** *auto*
have *6*: $(Suc\ k) \leq intlen\ \sigma$
using *1 3* **by** *auto*
have *7*: $(\forall j.\ 0 < j \wedge j < (Suc\ k) \longrightarrow f\ (suffix\ j\ \sigma))$
using *3 less-Suc-eq-0-disj* **by** *auto*
have *8*: $(\exists k>0.\ k \leq intlen\ \sigma \wedge g\ (suffix\ k\ \sigma) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma)))$
using *3 6 7* **by** *blast*
have *9*: $0 < intlen\ \sigma \wedge$
$(\exists k>0.\ k \leq intlen\ \sigma \wedge g\ (suffix\ k\ \sigma) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma)))$
using *1 8* **by** *blast*
from *9* **show** *?thesis* **by** (*simp add*: *suntil-d-def*)
qed

lemma *SUntilNextUntilsem*:
$\sigma \models\ f\ \mathcal{U}^s\ g = \bigcirc\ (\ f\ \mathcal{U}\ g)$
using *SUntilNextUntilsema SUntilNextUntilsemb unl-lift2* **by** *blast*

lemma *SSincePrevSincesema*:
assumes $\sigma \models f\ \mathcal{S}^s\ g$
shows $\sigma \models prev\ (f\ \mathcal{S}\ g)$
proof −
have *1*: $0 < intlen\ \sigma\ \wedge$
$(\exists\, k{<}intlen\ \sigma.\ g\ (prefix\ k\ \sigma) \wedge (\forall j.\ k < j \wedge j < intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma)))$
using *assms* **by** (*simp add*: *ssince-d-def*)
have *2*: $(\exists\, k{<}intlen\ \sigma.\ g\ (prefix\ k\ \sigma) \wedge (\forall j.\ k < j \wedge j < intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma)))$
using *1* **by** *auto*
obtain *k* **where** *3*: $k{<}intlen\ \sigma \wedge g\ (prefix\ k\ \sigma) \wedge (\forall j.\ k < j \wedge j < intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma))$
using *2* **by** *auto*
have *4*: $k{\le}intlen\ \sigma - Suc\ 0$
using *3* **by** *linarith*
have *5*: $g\ (prefix\ k\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma))$
by (*simp add*: *1 3 interval-pref-pref-help*)
have *6*: $(\forall j.\ k < j \wedge j \le intlen\ \sigma - Suc\ 0 \longrightarrow f\ (prefix\ j\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma)))$
using *3 interval-pref-pref-help* **by** *fastforce*
have *7*: $(\exists\, k{\le}intlen\ \sigma - Suc\ 0.$
$g\ (prefix\ k\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma)) \wedge$
$(\forall j.\ k < j \wedge j \le intlen\ \sigma - Suc\ 0 \longrightarrow f\ (prefix\ j\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma))))$
using *4 5 6* **by** *blast*
have *8*: $0 < intlen\ \sigma\ \wedge$
$(\exists\, k{\le}intlen\ \sigma - Suc\ 0.$
$g\ (prefix\ k\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma)) \wedge$
$(\forall j.\ k < j \wedge j \le intlen\ \sigma - Suc\ 0 \longrightarrow f\ (prefix\ j\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma))))$
using *1 7* **by** *blast*
from *8* **show** *?thesis* **by** (*simp add*: *min.absorb1 prev-defs since-d-def*)
qed

lemma *SSincePrevSincesemb*:
assumes $\sigma \models prev\ (f\ \mathcal{S}\ g)$
shows $\sigma \models f\ \mathcal{S}^s\ g$
proof −
have *1*: $0 < intlen\ \sigma\ \wedge$
$(\exists\, k{\le}intlen\ \sigma - Suc\ 0.$
$g\ (prefix\ k\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma)) \wedge$
$(\forall j.\ k < j \wedge j \le intlen\ \sigma - Suc\ 0 \longrightarrow f\ (prefix\ j\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma))))$
using *assms* **by** (*simp add*: *min.absorb1 prev-defs since-d-def*)
have *2*: $(\exists\, k{\le}intlen\ \sigma - Suc\ 0.$
$g\ (prefix\ k\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma)) \wedge$
$(\forall j.\ k < j \wedge j \le intlen\ \sigma - Suc\ 0 \longrightarrow f\ (prefix\ j\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma))))$
using *1* **by** *auto*
obtain *k* **where** *3*: $k{\le}intlen\ \sigma - Suc\ 0 \wedge g\ (prefix\ k\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma)) \wedge$
$(\forall j.\ k < j \wedge j \le intlen\ \sigma - Suc\ 0 \longrightarrow f\ (prefix\ j\ (prefix\ (intlen\ \sigma - Suc\ 0)\ \sigma)))$
using *2* **by** *auto*
have *4*: $k{<}intlen\ \sigma$
using *1 3* **by** *linarith*
have *5*: $g\ (prefix\ k\ \sigma)$
using *3 4 interval-pref-pref-help* **by** *force*

have *6*: $(\forall j.\ k < j \wedge j < intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma))$
  using *3 interval-pref-pref-help* **by** *fastforce*
 have *7*: $(\exists k{<}intlen\ \sigma.\ g\ (prefix\ k\ \sigma) \wedge (\forall j.\ k < j \wedge j < intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma)))$
  using *4 5 6* **by** *blast*
 have *8*: $0 < intlen\ \sigma\ \wedge$
   $(\exists k{<}intlen\ \sigma.\ g\ (prefix\ k\ \sigma) \wedge (\forall j.\ k < j \wedge j < intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma)))$
  using *1 7* **by** *blast*
 from *8* **show** *?thesis* **by** (*simp add*: *ssince-d-def*)
qed

lemma *SSincePrevSincesem*:
 $\sigma \models\ f\ \mathcal{S}^s\ g = prev\ (\ f\ \mathcal{S}\ g)$
using *SSincePrevSincesema SSincePrevSincesemb unl-lift2* **by** *blast*

lemma *UntilSUntilsem*:
 $\sigma \models f\ \mathcal{U}\ g = (g \vee (f \wedge f\ \mathcal{U}^s\ g))$
proof (*auto simp add*: *suntil-d-def until-d-def*)
 show $\bigwedge k.\ k \leq intlen\ \sigma \Longrightarrow g\ (suffix\ k\ \sigma) \Longrightarrow \forall j{<}k.\ f\ (suffix\ j\ \sigma) \Longrightarrow \neg\ g\ \sigma \Longrightarrow f\ \sigma$
  by *fastforce*
 show $\bigwedge k.\ k \leq intlen\ \sigma \Longrightarrow g\ (suffix\ k\ \sigma) \Longrightarrow \forall j{<}k.\ f\ (suffix\ j\ \sigma) \Longrightarrow \neg\ g\ \sigma \Longrightarrow 0 < intlen\ \sigma$
  using *gr0I* **by** *force*
 show $\bigwedge k.\ k \leq intlen\ \sigma \Longrightarrow$
   $g\ (suffix\ k\ \sigma) \Longrightarrow$
   $\forall j{<}k.\ f\ (suffix\ j\ \sigma) \Longrightarrow$
   $\neg\ g\ \sigma \Longrightarrow$
   $\exists k{>}0.\ k \leq intlen\ \sigma \wedge g\ (suffix\ k\ \sigma) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma))$
  by (*metis interval-suffix-zero neq0-conv*)
 show $\bigwedge k.\ f\ \sigma \Longrightarrow$
   $0 < k \Longrightarrow$
   $k \leq intlen\ \sigma \Longrightarrow$
   $g\ (suffix\ k\ \sigma) \Longrightarrow$
   $\forall j.\ 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma) \Longrightarrow$
   $\exists k{\leq}intlen\ \sigma.\ g\ (suffix\ k\ \sigma) \wedge (\forall j{<}k.\ f\ (suffix\ j\ \sigma))$
  by (*metis Suc-pred interval-suffix-zero less-Suc-eq-0-disj*)
qed

lemma *SinceSSincesem*:
 $\sigma \models f\ \mathcal{S}\ g = (\ g \vee (f \wedge f\ \mathcal{S}^s\ g))$
proof (*auto simp add*: *ssince-d-def since-d-def*)
 show $\bigwedge k.\ k \leq intlen\ \sigma \Longrightarrow$
   $g\ (prefix\ k\ \sigma) \Longrightarrow$
   $\forall j.\ k < j \wedge j \leq intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma) \Longrightarrow$
   $\neg\ g\ \sigma \Longrightarrow$
   $f\ \sigma$
  using *le-eq-less-or-eq* **by** *auto*
 show $\bigwedge k.\ k \leq intlen\ \sigma \Longrightarrow$
   $g\ (prefix\ k\ \sigma) \Longrightarrow$
   $\forall j.\ k < j \wedge j \leq intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma) \Longrightarrow$
   $\neg\ g\ \sigma \Longrightarrow$
   $0 < intlen\ \sigma$

by (*metis gr0I interval-prefix-intlen le-zero-eq*)
show $\bigwedge k.\ k \le intlen\ \sigma \Longrightarrow$
$g\ (prefix\ k\ \sigma) \Longrightarrow$
$\forall j.\ k < j \wedge j \le intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma) \Longrightarrow$
$\neg\ g\ \sigma \Longrightarrow$
$\exists\, k{<}intlen\ \sigma.\ g\ (prefix\ k\ \sigma) \wedge (\forall j.\ k < j \wedge j < intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma))$
using *le-eq-less-or-eq* **by** *auto*
show $\bigwedge k.\ f\ \sigma \Longrightarrow$
$k < intlen\ \sigma \Longrightarrow$
$g\ (prefix\ k\ \sigma) \Longrightarrow$
$\forall j.\ k < j \wedge j < intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma) \Longrightarrow$
$\exists\, k{\le}intlen\ \sigma.\ g\ (prefix\ k\ \sigma) \wedge (\forall j.\ k < j \wedge j \le intlen\ \sigma \longrightarrow f\ (prefix\ j\ \sigma))$
by (*metis antisym-conv2 interval-prefix-intlen less-imp-le-nat*)
qed

lemma *UntilAndDistsem*:
$\sigma \models (f \wedge g)\ \mathcal{U}\ h = ((f\ \mathcal{U}\ h) \wedge (g\ \mathcal{U}\ h))$
by (*auto simp add*: *until-d-def*)
(*metis dual-order.strict-trans linorder-cases*)

lemma *UntilOrDistsem*:
$\sigma \models f\ \mathcal{U}\ (g \vee h) = (\ f\ \mathcal{U}\ g \vee f\ \mathcal{U}\ h)$
by (*auto simp add*: *until-d-def*)

lemma *NextUntilsema*:
assumes $(\sigma \models \bigcirc(f\ \mathcal{U}\ g))$
shows $(\sigma \models (\bigcirc\ f)\ \mathcal{U}\ (\bigcirc\ g))$
proof −
have *0*: $0 < intlen\ \sigma\ \wedge$
$(\exists\, k{\le}intlen\ (suffix\ (Suc\ 0)\ \sigma).\ g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j{<}k.\ f\ (suffix\ (Suc\ j)\ \sigma)))$
using *assms* **by** (*auto simp add*: *next-defs until-d-def*)
have *1*: $0 < intlen\ \sigma$
using *0* **by** *auto*
have *2*: $(\exists\, k{\le}intlen\ (suffix\ (Suc\ 0)\ \sigma).\ g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j{<}k.\ f\ (suffix\ (Suc\ j)\ \sigma)))$
using *0* **by** *auto*
obtain *k* **where** *3*: $k{\le}intlen\ (suffix\ (Suc\ 0)\ \sigma) \wedge g\ (suffix\ (Suc\ k)\ \sigma)\ \wedge$
$(\forall j{<}k.\ f\ (suffix\ (Suc\ j)\ \sigma))$
using *2* **by** *auto*
have *4*: $g\ (suffix\ (Suc\ k)\ \sigma)$
using *3* **by** *auto*
have *5*: $k{\le}\ intlen\ \sigma$
using *3 0* **by** *auto*
have *6*: $0 < intlen\ (suffix\ k\ \sigma)$
using *1 3* **by** *auto*
have *7*: $(\forall j{<}k.\ 0 < intlen\ (suffix\ j\ \sigma) \wedge f\ (suffix\ (Suc\ j)\ \sigma))$
using *3 5* **by** *force*
have *8*: $\exists\, k{\le}intlen\ \sigma.$
$0 < intlen\ (suffix\ k\ \sigma)\ \wedge$
$g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j{<}k.\ 0 < intlen\ (suffix\ j\ \sigma) \wedge f\ (suffix\ (Suc\ j)\ \sigma))$

using *3 5 6 7* **by** *blast*
from *8* **show** *?thesis* **by** (*simp add*: *next-defs until-d-def*)
qed

lemma *NextUntilsemb*:
assumes ($\sigma \models (\bigcirc f)\ \mathcal{U}\ (\bigcirc g)$)
shows ($\sigma \models \bigcirc(f\ \mathcal{U}\ g)$)
proof −
have *1*: $\exists\, k \leq intlen\ \sigma.$
$0 < intlen\ (suffix\ k\ \sigma) \wedge$
$g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall\, j<k.\ 0 < intlen\ (suffix\ j\ \sigma) \wedge f\ (suffix\ (Suc\ j)\ \sigma))$
using *assms* **by** (*auto simp add*: *next-defs until-d-def*)
obtain *k* **where** *2*: $k \leq intlen\ \sigma \wedge 0 < intlen\ (suffix\ k\ \sigma) \wedge$
$g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall\, j<k.\ 0 < intlen\ (suffix\ j\ \sigma) \wedge f\ (suffix\ (Suc\ j)\ \sigma))$
using *1* **by** *auto*
have *3*: $0 < intlen\ \sigma$
using *2* **by** *auto*
have *4*: $k \leq intlen\ (suffix\ (Suc\ 0)\ \sigma)$
using *2 interval-suffix-length-good* **by** *auto*
have *5*: $g\ (suffix\ (Suc\ k)\ \sigma)$
using *2* **by** *auto*
have *6*: $(\forall\, j<k.\ f\ (suffix\ (Suc\ j)\ \sigma))$
using *2* **by** *blast*
have *7*: $0 < intlen\ \sigma \wedge$
$(\exists\, k \leq intlen\ (suffix\ (Suc\ 0)\ \sigma).\ g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall\, j<k.\ f\ (suffix\ (Suc\ j)\ \sigma)))$
using *2 3 4* **by** *blast*
from *7* **show** *?thesis* **by** (*auto simp*: *next-defs until-d-def*)
qed

lemma *NextUntilsem*:
$\sigma \models \bigcirc(f\ \mathcal{U}\ g) = (\bigcirc f)\ \mathcal{U}\ (\bigcirc g)$
using *NextUntilsema NextUntilsemb* **using** *unl-lift2* **by** *blast*

lemma *UntilUntilsem*:
$\sigma \models f\ \mathcal{U}\ g = f\ \mathcal{U}\ (f\ \mathcal{U}\ g)$
proof (*auto simp add*: *until-d-def*)
show $\bigwedge k.\ k \leq intlen\ \sigma \Longrightarrow$
$g\ (suffix\ k\ \sigma) \Longrightarrow$
$\forall\, j<k.\ f\ (suffix\ j\ \sigma) \Longrightarrow$
$\exists\, k \leq intlen\ \sigma.$
$(\exists\, ka \leq intlen\ \sigma - k.$
$g\ (suffix\ (ka + k)\ \sigma) \wedge (\forall\, j<ka.\ f\ (suffix\ (j + k)\ \sigma))) \wedge (\forall\, j<k.\ f\ (suffix\ j\ \sigma))$
by *force*
show $\bigwedge k\ ka.$
$k \leq intlen\ \sigma \Longrightarrow$
$\forall\, j<k.\ f\ (suffix\ j\ \sigma) \Longrightarrow$
$ka \leq intlen\ \sigma - k \Longrightarrow$
$g\ (suffix\ (ka + k)\ \sigma) \Longrightarrow$
$\forall\, j<ka.\ f\ (suffix\ (j + k)\ \sigma) \Longrightarrow \exists\, k \leq intlen\ \sigma.\ g\ (suffix\ k\ \sigma) \wedge (\forall\, j<k.\ f\ (suffix\ j\ \sigma))$
by (*metis leI le-add-diff-inverse2 less-diff-conv2*

ordered-cancel-comm-monoid-diff-class.le-diff-conv2)
qed

lemma *LFPUntilsem1*:
assumes $\forall\, n{\le}intlen\ \sigma.$
$(g\ (suffix\ n\ \sigma) \longrightarrow h\ (suffix\ n\ \sigma)) \wedge$
$(f\ (suffix\ n\ \sigma) \wedge n < intlen\ \sigma \wedge h\ (suffix\ (Suc\ n)\ \sigma) \longrightarrow$
$h\ (suffix\ n\ \sigma))$
$k \le intlen\ \sigma$
$g\ (suffix\ k\ \sigma)$
$\forall\, j{<}k.\ f\ (suffix\ j\ \sigma)$
shows $h\ \sigma$
using *assms*
proof (*induct k arbitrary*: σ)
case *0*
then show *?case* **by** *auto*
next
case (*Suc k*)
then show *?case*
proof (*cases* σ)
case (*St x1*)
then show *?thesis* **using** *Suc.prems Suc.hyps* **by** (*metis suffix.simps*(*1*))
next
case (*Cons x21 x22*)
then show *?thesis*
using *Suc.prems Suc.hyps*
by (*metis Suc-less-eq interval-suffix-suc interval-suffix-zero*
intlen.simps(*2*) *le-eq-less-or-eq not-less-eq-eq plus-1-eq-Suc zero-less-Suc*)
qed
qed

lemma *LFPUntilsem*:
$\sigma \models \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h)$
using *LFPUntilsem1* **by** (*simp add*: *always-defs next-defs until-d-def*, *blast*)

lemma *RevUntilsema*:
assumes $\sigma \models (f\ \mathcal{U}\ g)^r$
shows $\sigma \models (\ (f^r)\ \mathcal{S}\ (g^r)\ )$
proof −
have *1*: $\sigma \models (f\ \mathcal{U}\ g)^r$
using *assms* **by** *auto*
have *2*: $\exists\, k{\le}intlen\ \sigma.$
$g\ (suffix\ k\ (intrev\ \sigma)) \wedge (\forall\, j{<}k.\ f\ (suffix\ j\ (intrev\ \sigma)))$
using *1* **by** (*simp add*: *until-d-def reverse-d-def*)
obtain *k* **where** *3*: $k{\le}intlen\ \sigma \wedge g\ (suffix\ k\ (intrev\ \sigma)) \wedge$
$(\forall\, j{<}k.\ f\ (suffix\ j\ (intrev\ \sigma)))$
using *2* **by** *auto*
have *4*: $g\ (intrev\ (prefix\ (intlen\ \sigma\ -k)\ \sigma))$

by (*simp add*: *3 interval-intrev-prefix*)
have *5*: $(\forall j{<}k.\ f\ (intrev\ (prefix\ (intlen\ \sigma\ - j)\ \sigma)))$
by (*simp add*: *3 interval-intrev-prefix*)
have *6*: $(intlen\ \sigma\ - k) \leq intlen\ \sigma$
using *diff-le-self* **by** *blast*
have *7*: $(\forall j.\ (intlen\ \sigma\ - k) < j \wedge j \leq intlen\ \sigma \longrightarrow f\ (intrev\ (prefix\ j\ \sigma)))$
by (*simp add*: *3 interval-intrev-prefix less-diff-conv2*)
have *8*: $\exists\, k{\leq}intlen\ \sigma.\ g\ (intrev\ (prefix\ k\ \sigma)) \wedge$
$(\forall j.\ k < j \wedge j \leq intlen\ \sigma \longrightarrow f\ (intrev\ (prefix\ j\ \sigma)))$
using *4 6 7* **by** *blast*
have *10*: $\sigma \models (\ (f^r)\ \mathcal{S}\ (g^r)\ )$
using *8* **by** (*simp add*: *since-d-def reverse-d-def*)
show *?thesis* **by** (*simp add*: *10*)
qed

lemma *RevUntilsemb*:
assumes $\sigma \models (\ (f^r)\ \mathcal{S}\ (g^r)\ )$
shows $\sigma \models (f\ \mathcal{U}\ g)^r$
proof −
have *1*: $\sigma \models (\ (f^r)\ \mathcal{S}\ (g^r)\ )$
using *assms* **by** *auto*
have *2*: $\exists\, k{\leq}intlen\ \sigma.\ g\ (intrev\ (prefix\ k\ \sigma)) \wedge$
$(\forall j.\ k < j \wedge j \leq intlen\ \sigma \longrightarrow f\ (intrev\ (prefix\ j\ \sigma)))$
using *1* **by** (*simp add*: *since-d-def reverse-d-def*)
obtain *k* **where** *3*: $k{\leq}intlen\ \sigma \wedge g\ (intrev\ (prefix\ k\ \sigma)) \wedge$
$(\forall j.\ k < j \wedge j \leq intlen\ \sigma \longrightarrow f\ (intrev\ (prefix\ j\ \sigma)))$
using *2* **by** *auto*
have *4*: $g\ (suffix\ (intlen\ \sigma\ - k)\ (intrev\ \sigma))$
using *3 interval-intrev-prefix* **by** *fastforce*
have *5*: $(\forall j.\ j{<}\ intlen\ \sigma\ - k \longrightarrow f\ (suffix\ j\ (intrev\ \sigma)))$
by (*metis 3 add.commute diff-le-self dual-order.trans interval-intrev-intlen interval-intrev-suffix interval-rev-rev-ident less-diff-conv less-imp-le-nat*)
have *6*: $\exists\, k{\leq}intlen\ \sigma.\ g\ (suffix\ k\ (intrev\ \sigma)) \wedge$
$(\forall j{<}k.\ f\ (suffix\ j\ (intrev\ \sigma)))$
using *4 5 diff-le-self* **by** *blast*
have *7*: $\sigma \models (f\ \mathcal{U}\ g)^r$
using *6* **by** (*simp add*: *until-d-def reverse-d-def*)
show *?thesis*
using *7* **by** *blast*
qed

lemma *RevUntilsem*:
$\sigma \models (f\ \mathcal{U}\ g)^r = (\ (f^r)\ \mathcal{S}\ (g^r)\ )$
using *RevUntilsema RevUntilsemb* **using** *unl-lift2* **by** *blast*

lemma *UntilRightAndsem*:
assumes $(\sigma \models f\ \mathcal{U}\ (g \wedge h))$
shows $(\sigma \models (f\ \mathcal{U}\ g)\ \mathcal{U}\ h)$
proof −
have *1*: $\exists\, k{\leq}intlen\ \sigma.\ g\ (suffix\ k\ \sigma) \wedge h\ (suffix\ k\ \sigma) \wedge (\forall j{<}k.\ f\ (suffix\ j\ \sigma))$

```
  using assms by (simp add: until-d-def)
 obtain k where 2: k≤intlen σ ∧ g (suffix k σ) ∧ h (suffix k σ) ∧ (∀ j<k. f (suffix j σ))
  using 1 by auto
 have 3: h (suffix k σ)
  using 2 by auto
 have 4: k≤intlen σ
  using 2 by auto
 have 5: (∀ j<k.
        ∃ ka≤intlen (suffix j σ). g (suffix (ka + j) σ) ∧ (∀ ja<ka. f (suffix (ja + j) σ)))
   proof
    fix j
    show j < k ⟶
         (∃ ka≤intlen (suffix j σ). g (suffix (ka + j) σ) ∧ (∀ ja<ka. f (suffix (ja + j) σ)))
     proof
      assume a0: j < k
      show (∃ ka≤intlen (suffix j σ). g (suffix (ka + j) σ) ∧ (∀ ja<ka. f (suffix (ja + j) σ)))
       proof −
        have 51: k−j ≤ intlen (suffix j σ)
            using 4 a0 by auto
        have 52:  g (suffix ((k−j) + j) σ)
          by (simp add: 2 a0 less-imp-le-nat)
        have 53: (∀ ja<(k−j). f (suffix (ja + j) σ))
          using 2 less-diff-conv by blast
       show ?thesis
       using 51 52 53 by blast
      qed
     qed
    qed
 have 6: ∃ k≤intlen σ.
     h (suffix k σ) ∧
     (∀ j<k. ∃ k≤intlen (suffix j σ). g (suffix (k + j) σ) ∧ (∀ ja<k. f (suffix (ja + j) σ)))
  using 2 5 by blast
 from 6 show ?thesis by (simp add: until-d-def)
qed

lemma interval-suf-first:
assumes (∃ i≤intlen xs. f (suffix i xs))
shows   (∃ i≤intlen xs. f (suffix i xs) ∧
         (∀ j. j<i ⟶ ¬ (f (suffix j xs))))
using assms interval-suf-first-upto[of intlen xs +1 f xs ] by (simp add: discrete)

lemma NotSuffixFirst:
assumes (∃ n≤intlen xs. ¬ f (suffix n xs))
shows   (∃ n ≤ intlen xs. ¬ f (suffix n xs) ∧ (∀ k. k<n ⟶ f (suffix k xs)))
using assms interval-suf-first[of xs LIFT(¬ f)] by auto

lemma NotSuffixFirst-upto:
assumes (∃ i<k. ¬ f (suffix i xs))
       k ≤ intlen xs +1
shows   (∃ i<k. ¬f (suffix i xs) ∧
```

$(\forall j<i\ .\ \ (f\ \ (suffix\ j\ xs))))$
using *assms interval-suf-first-upto*[*of k LIFT*$(\neg\ f)$ *xs*] **by** *auto*

lemma *WaitNotDistUntilsem1*:
assumes $(\sigma \models \neg(f\ \mathcal{W}\ g))$
shows $(\sigma \models ((\neg\ g)\ \mathcal{U}\ ((\neg\ f) \wedge (\neg g))))$
proof −
have *1*: $(\forall\, k.\ g\ (suffix\ k\ \sigma) \longrightarrow k \leq intlen\ \sigma \longrightarrow (\exists\, j<k.\ \neg\ f\ \ (suffix\ j\ \sigma))) \wedge$
$(\exists\, n\leq intlen\ \sigma.\ \neg\ f\ \ (suffix\ n\ \sigma))$
using *assms* **by** (*simp add*: *wait-d-def until-d-def always-defs*)
have *2*: $(\forall\, k.\ k \leq intlen\ \sigma \longrightarrow \neg\ g\ \ (suffix\ k\ \sigma) \vee (\exists\, j<k.\ \neg\ f\ \ (suffix\ j\ \sigma)))$
using *1* **by** *auto*
have *3*: $(\exists\, n\leq intlen\ \sigma.\ \neg\ f\ \ (suffix\ n\ \sigma))$
using *1* **by** *auto*
obtain *n* **where** *4*: $n\leq\ intlen\ \sigma \wedge \neg\ f\ \ (suffix\ n\ \sigma) \wedge$
$(\forall\ \ k\ <n\ .\ f\ \ (suffix\ k\ \sigma))$
using *3* **using** *NotSuffixFirst* **by** *blast*
have *16*: $n \leq intlen\ \sigma$
by (*simp add*: *4*)
have *17*: $\neg\ g\ \ (suffix\ n\ \sigma)$
using *1 4* **by** *blast*
have *18*: $(\forall\, j<n.\ \neg\ g\ \ (suffix\ j\ \sigma))$
by (*meson 2 4 le-eq-less-or-eq less-le-trans*)
have *19*: $\exists\, k\leq intlen\ \sigma.\ \neg\ f\ \ (suffix\ k\ \sigma) \wedge \neg\ g\ \ (suffix\ k\ \sigma) \wedge (\forall\, j<k.\ \neg\ g\ \ (suffix\ j\ \sigma))$
using *16 17 18 4* **by** *blast*
have *20*: $(\sigma \models ((\neg\ g)\ \mathcal{U}\ (\neg\ f\ \wedge \neg g)))$
using *19* **by** (*simp add*: *until-d-def*)
show *?thesis* **using** *20* **by** *auto*
qed

lemma *WaitNotDistUntilsem2*:
assumes $(\sigma \models ((\neg\ g)\ \mathcal{U}\ ((\neg\ f) \wedge (\neg g))))$
shows $(\sigma \models \neg(f\ \mathcal{W}\ g))$
using *assms not-less-iff-gr-or-eq* **by** (*auto simp add*: *always-defs wait-d-def until-d-def*)

lemma *WaitNotDistUntilsem*:
$(\sigma \models (\neg(f\ \mathcal{W}\ g))\ \ =\ \ ((\neg\ g)\ \mathcal{U}\ ((\neg\ f) \wedge (\neg g))))$
using *WaitNotDistUntilsem1 WaitNotDistUntilsem2 unl-lift2* **by** *blast*

lemma *SUntilNextUntil*:
$\vdash f\ \mathcal{U}^s\ g = \bigcirc\ (\ f\ \mathcal{U}\ g)$
using *SUntilNextUntilsem Valid-def* **by** *blast*

lemma *SSincePrevSince*:
$\vdash f\ \mathcal{S}^s\ g = prev\ \ (\ f\ \mathcal{S}\ g)$
using *SSincePrevSincesem Valid-def* **by** *blast*

lemma *UntilSUntil*:
$\vdash f\ \mathcal{U}\ g = (g \lor (f \land f\ \mathcal{U}^s\ g))$
using *UntilSUntilsem Valid-def* **by** *blast*

lemma *SinceSSince*:
$\vdash f\ \mathcal{S}\ g = (g \lor (f \land f\ \mathcal{S}^s\ g))$
using *SinceSSincesem Valid-def* **by** *blast*

lemma *UntilAndDist*:
$\vdash (f \land g)\ \mathcal{U}\ h = ((f\ \mathcal{U}\ h) \land (g\ \mathcal{U}\ h))$
using *UntilAndDistsem Valid-def* **by** *blast*

lemma *UntilOrDist*:
$\vdash f\ \mathcal{U}\ (g \lor h) = (\ f\ \mathcal{U}\ g \lor f\ \mathcal{U}\ h)$
using *UntilOrDistsem Valid-def* **by** *blast*

lemma *NextUntil*:
$\vdash \bigcirc(f\ \mathcal{U}\ g) = (\bigcirc\ f)\ \mathcal{U}\ (\bigcirc\ g)$
using *NextUntilsem Valid-def* **by** *blast*

lemma *UntilUntil*:
$\vdash f\ \mathcal{U}\ g = f\ \mathcal{U}\ (f\ \mathcal{U}\ g)$
using *UntilUntilsem Valid-def* **by** *blast*

lemma *LFPUntil*:
$\vdash \Box((g \lor (f \land \bigcirc h)) \longrightarrow h) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h)$
using *LFPUntilsem Valid-def* **by** *blast*

lemma *RevUntil*:
$\vdash (f\ \mathcal{U}\ g)^r = (\ (f^r)\ \mathcal{S}\ (g^r)\ )$
using *RevUntilsem Valid-def* **by** *blast*

lemma *UntilEqvUntil*:
assumes $\vdash$ *f0* = *f1*
$\vdash$ *g0* = *g1*
shows $\vdash$ *f0* $\mathcal{U}$ *g0* = *f1* $\mathcal{U}$ *g1*
using *assms* **by** (*simp add*: *until-d-def Valid-def*)

lemma *UntilImpUntil*:
assumes $\vdash$ *f0* $\longrightarrow$ *f1*
$\vdash$ *g0* $\longrightarrow$ *g1*
shows $\vdash$ *f0* $\mathcal{U}$ *g0* $\longrightarrow$ *f1* $\mathcal{U}$ *g1*
using *assms* **by** (*auto simp add*: *until-d-def Valid-def*)

lemma *SinceEqvSince*:
assumes $\vdash$ *f0* = *f1*
$\vdash$ *g0* = *g1*
shows $\vdash$ *f0* $\mathcal{S}$ *g0* = *f1* $\mathcal{S}$ *g1*

using *assms* **by** (*simp add*: *since-d-def Valid-def*)

lemma *SinceImpSince*:
assumes $\vdash f0 \longrightarrow f1$
$\vdash g0 \longrightarrow g1$
shows $\vdash f0\ \mathcal{S}\ g0 \longrightarrow f1\ \mathcal{S}\ g1$
using *assms* **by** (*auto simp add*: *since-d-def Valid-def*)

lemma *UntilLeftDistAnd*:
$\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow f\ \mathcal{U}\ g \wedge f\ \mathcal{U}\ h$
by (*auto simp add*: *Valid-def until-d-def*)

lemma *UntilRightDistOr*:
$\vdash f\ \mathcal{U}\ h \vee g\ \mathcal{U}\ h \longrightarrow (f \vee g)\ \mathcal{U}\ h$
by (*auto simp add*: *Valid-def until-d-def*)

lemma *UntilNotImp*:
$\vdash f\ \mathcal{U}\ g \wedge (\neg\ g)\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h$
by (*auto simp add*: *Valid-def until-d-def*)
(*metis less-trans linorder-cases*)

lemma *UntilRightOr*:
$\vdash f\ \mathcal{U}\ (\ g\ \mathcal{U}\ h\ ) \longrightarrow (f \vee g)\ \mathcal{U}\ h$
by (*auto simp add*: *Valid-def until-d-def*)
(*metis Nat.le-diff-conv2 leI le-add-diff-inverse2 less-diff-conv2*)

lemma *DiamondEqvTrueUntil*:
$\vdash \diamond\ f = \#True\ \mathcal{U}\ f$
by (*simp add*: *Valid-def sometimes-defs until-d-def*)

lemma *UntilRightAnd*:
$\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow (f\ \mathcal{U}\ g)\ \mathcal{U}\ h$
using *UntilRightAndsem Valid-def* **by** *auto*

lemma *WaitNotDistUntil*:
$\vdash (\neg(f\ \mathcal{W}\ g)) = ((\neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg g))$
using *WaitNotDistUntilsem Valid-def* **by** *blast*

lemma *UntilAlwaysAndDist*:
$\vdash \Box\ f \wedge g\ \mathcal{U}\ h \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h)$
by (*auto simp add*: *Valid-def always-defs until-d-def*)

lemma *UntilRightMono*:
$\vdash \Box(f \longrightarrow g) \longrightarrow (h\ \mathcal{U}\ f \longrightarrow h\ \mathcal{U}\ g)$
by (*auto simp add*: *Valid-def always-defs until-d-def*)

lemma *UntilLeftMono*:
$\vdash \Box\ (f \longrightarrow g) \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
by (*auto simp add*: *Valid-def always-defs until-d-def*)

lemma *UntilInduction-help*:
$\vdash \Box\ (f \longrightarrow \neg\ g \land \bigcirc\ f) \longrightarrow \Box(g \lor \#True \land \bigcirc\ (\neg\ f) \longrightarrow \neg\ f)$
by (*auto simp add*: *Valid-def always-defs next-defs*)

lemma *UntilInduction-a-help*:
$\vdash (f \longrightarrow ((\bigcirc\ f) \land g) \lor h) \longrightarrow (\ ((\neg g \land \neg h) \lor (\neg h \land \bigcirc(\neg\ f))) \longrightarrow \neg f)$
by (*auto simp add*: *Valid-def next-defs*)

lemma *UntilInduction-b-help*:
$\vdash \Box(f \longrightarrow (\bigcirc f) \lor g) \longrightarrow\ \ \Box(\ ((\neg\ f \land \neg\ g) \lor (\neg g \land \bigcirc\ (\neg\ f))) \longrightarrow \neg\ f)$
by (*auto simp add*: *Valid-def always-defs next-defs*)

lemma *UntilImpNot*:
$\vdash f\ \mathcal{U}\ g \longrightarrow (f \land \neg g)\ \mathcal{U}\ g$
by (*auto simp add*: *until-d-def Valid-def*)
  (*metis* (*full-types*) *interval-suf-first leI less-le-trans*)

lemma *WaitAndRule*:
$\vdash f\ \mathcal{W}\ g = (f \land \neg\ g)\ \mathcal{W}\ g$
proof (*auto simp add*: *Valid-def wait-d-def until-d-def always-defs*)
 show $\bigwedge w\ n.$
  $\forall k.\ g\ (suffix\ k\ w) \longrightarrow k \le intlen\ w \longrightarrow (\exists j<k.\ f\ (suffix\ j\ w) \longrightarrow g\ (suffix\ j\ w)) \Longrightarrow$
   $n \le intlen\ w \Longrightarrow$
   $\forall n \le intlen\ w.\ f\ (suffix\ n\ w) \Longrightarrow$
   $g\ (suffix\ n\ w) \Longrightarrow$
   False
 by (*metis* (*full-types*) *dual-order.order-iff-strict interval-suf-first less-le-trans*)
 show $\bigwedge w\ n\ k.$
  $\forall k.\ g\ (suffix\ k\ w) \longrightarrow k \le intlen\ w \longrightarrow (\exists j<k.\ f\ (suffix\ j\ w) \longrightarrow g\ (suffix\ j\ w)) \Longrightarrow$
  $n \le intlen\ w \Longrightarrow$
  $k \le intlen\ w \Longrightarrow$
  $g\ (suffix\ k\ w) \Longrightarrow$
  $\forall j<k.\ f\ (suffix\ j\ w) \Longrightarrow$
  $f\ (suffix\ n\ w)$
 by (*metis* (*full-types*) *interval-suf-first leI less-le-trans*)
 show $\bigwedge w\ n\ k.$
  $\forall k.\ g\ (suffix\ k\ w) \longrightarrow k \le intlen\ w \longrightarrow (\exists j<k.\ f\ (suffix\ j\ w) \longrightarrow g\ (suffix\ j\ w)) \Longrightarrow$
  $n \le intlen\ w \Longrightarrow$
  $k \le intlen\ w \Longrightarrow$
  $g\ (suffix\ k\ w) \Longrightarrow$
  $\forall j<k.\ f\ (suffix\ j\ w) \Longrightarrow$
  $g\ (suffix\ n\ w) \Longrightarrow$
  False
  by (*metis* (*full-types*) *interval-suf-first less-le-trans not-le-imp-less*)
qed

lemma *WaitLeftDistAnd*:
$\vdash f\ \mathcal{W}\ (g \land h) \longrightarrow f\ \mathcal{W}\ g \land f\ \mathcal{W}\ h$
by (*auto simp add*: *Valid-def wait-d-def until-d-def always-defs*)

lemma *WaitRightDistAnd*:
$\vdash (f \wedge g)\ \mathcal{W}\ h = (f\ \mathcal{W}\ h \wedge g\ \mathcal{W}\ h)$
proof (*auto simp add*: *Valid-def wait-d-def until-d-def always-defs*)
show $\bigwedge w\ n\ k\ ka.$
$\forall k.\ h\ (suffix\ k\ w) \longrightarrow k \leq intlen\ w \longrightarrow (\exists j{<}k.\ f\ (suffix\ j\ w) \longrightarrow \neg\ g\ (suffix\ j\ w)) \Longrightarrow$
$n \leq intlen\ w \Longrightarrow$
$k \leq intlen\ w \Longrightarrow$
$h\ (suffix\ k\ w) \Longrightarrow$
$\forall j{<}k.\ f\ (suffix\ j\ w) \Longrightarrow$
$ka \leq intlen\ w \Longrightarrow$
$h\ (suffix\ ka\ w) \Longrightarrow$
$\forall j{<}ka.\ g\ (suffix\ j\ w) \Longrightarrow$
$f\ (suffix\ n\ w)$
by (*metis less-trans linorder-cases*)
show $\bigwedge w\ n\ k\ ka.$
$\forall k.\ h\ (suffix\ k\ w) \longrightarrow k \leq intlen\ w \longrightarrow (\exists j{<}k.\ f\ (suffix\ j\ w) \longrightarrow \neg\ g\ (suffix\ j\ w)) \Longrightarrow$
$n \leq intlen\ w \Longrightarrow$
$k \leq intlen\ w \Longrightarrow$
$h\ (suffix\ k\ w) \Longrightarrow$
$\forall j{<}k.\ f\ (suffix\ j\ w) \Longrightarrow$
$ka \leq intlen\ w \Longrightarrow$
$h\ (suffix\ ka\ w) \Longrightarrow$
$\forall j{<}ka.\ g\ (suffix\ j\ w) \Longrightarrow$
$g\ (suffix\ n\ w)$
by (*metis less-trans linorder-cases*)
qed

lemma *WaitRightDistImp*:
$\vdash (f \longrightarrow g)\ \mathcal{W}\ h \longrightarrow (f\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ h)$
by (*auto simp add*: *Valid-def wait-d-def until-d-def always-defs*)
(*metis less-trans linorder-cases*)

lemma *WaitImpRule*:
$\vdash (f \longrightarrow g)\ \mathcal{W}\ f$
by (*auto simp add*: *Valid-def wait-d-def until-d-def always-defs*)
(*metis interval-suf-first*)

lemma *WaitInductionc-help*:
$\vdash \Box(f \longrightarrow \bigcirc\ f) \longrightarrow \Box\ (f \longrightarrow wnext\ f)$
by (*simp add*: *ImpBoxRule intI next-defs wnext-defs*)

lemma *WaitInductiond-help*:
$\vdash \Box(f \longrightarrow g \wedge \bigcirc\ f) \longrightarrow \Box\ (f \longrightarrow\ wnext\ f)$
by (*simp add*: *Valid-def always-defs next-defs wnext-defs*)

lemma *WaitOrder-help*:
$\vdash (\Box\ (\neg\ f) \vee \Box\ (\neg\ g)) \vee \Diamond(f \vee g)$
by (*auto simp add*: *always-defs sometimes-defs Valid-def*)

21.3 Lemmas

lemma *NextFalseSUntil*:
$\vdash \bigcirc\ g = \#False\ \mathcal{U}^s\ g$
by (*metis SUntilNextUntil UntilSUntil int-simps*(*19*) *int-simps*(*25*) *inteq-reflection*)

lemma *PrevFalseSSince*:
$\vdash prev\ g = \#False\ \mathcal{S}^s\ g$
by (*metis SSincePrevSince SinceSSince int-simps*(*19*) *int-simps*(*25*) *inteq-reflection*)

lemma *UntilUnrol*:
$\vdash f\ \mathcal{U}\ g = (g \lor (f \land \bigcirc(f\ \mathcal{U}\ g)))$
by (*metis SUntilNextUntil UntilSUntil inteq-reflection*)

lemma *WNextUntil*:
$\vdash wnext(f\ \mathcal{U}\ g) = (empty \lor (\bigcirc\ f)\ \mathcal{U}\ (\bigcirc\ g))$
by (*meson NextUntil Prop06 WnextEqvEmptyOrNext*)

lemma *UntilRelease*:
$\vdash f\ \mathcal{R}\ g = (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g)))$
by (*simp add*: *release-d-def*)

lemma *SReleaseWait*:
$\vdash f\ \mathcal{M}\ g = (\neg\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))$
by (*simp add*: *srelease-d-def*)

lemma *ReleaseUntil*:
$\vdash f\ \mathcal{U}\ g = (\neg\ (\ (\neg\ f)\ \mathcal{R}\ (\neg\ g)))$
by (*simp add*: *release-d-def*)

lemma *WaitSRelease*:
$\vdash f\ \mathcal{W}\ g = (\neg\ (\neg\ f)\ \mathcal{M}\ (\neg\ g))$
by (*simp add*: *srelease-d-def*)

lemma *NotUntilRelease*:
$\vdash \neg(f\ \mathcal{U}\ g) = (\neg\ f)\ \mathcal{R}\ (\neg\ g)$
by (*simp add*: *ReleaseUntil*)

lemma *NotWaitSRelease*:
$\vdash \neg(f\ \mathcal{W}\ g) = (\neg\ f)\ \mathcal{M}\ (\neg\ g)$
by (*simp add*: *WaitSRelease*)

lemma *NotReleaseUntil*:
$\vdash \neg(f\ \mathcal{R}\ g) = (\neg\ f)\ \mathcal{U}\ (\neg\ g)$
by (*simp add*: *UntilRelease*)

lemma *NotSReleaseWait*:
$\vdash \neg(f\ \mathcal{M}\ g) = (\neg\ f)\ \mathcal{W}\ (\neg\ g)$
by (*simp add*: *SReleaseWait*)

lemma *BoxEqvFalseRelease*:

$\vdash \Box\ f = \#False\ \mathcal{R}\ f$
by (*metis DiamondEqvTrueUntil EqvReverseReverse always-d-def int-simps*(*3*) *inteq-reflection release-d-def*)

lemma *RevSince*:
$\vdash (f\ \mathcal{S}\ g)^r = (\ (f^r)\ \mathcal{U}\ (g^r)\ )$
proof −
have *1*: $\vdash (f^r\ \mathcal{U}\ g^r)^r = (\ (f^r)^r\ \mathcal{S}\ (g^r)^r\ )$
by (*simp add*: *RevUntil*)
show *?thesis*
by (*metis 1 EqvReverseReverse inteq-reflection*)
qed

lemma *LFPSince*:
$\vdash$ *bi* $((g \vee (f \wedge \mathit{prev}\ h)) \longrightarrow h) \longrightarrow (f\ \mathcal{S}\ g \longrightarrow h)$
by (*metis* (*no-types*, *lifting*) *LFPUntil RBiEqvBox RPrevEqvNext RevSince ReverseEqv all-rev-eq*(*3*) *inteq-reflection*)

lemma *UntilTrue*:
$\vdash f\ \mathcal{U}\ \#True$
using *UntilSUntil* **by** *fastforce*

lemma *NotUntilFalse*:
$\vdash \neg\ (f\ \mathcal{U}\ \#False)$
by (*simp add*: *intI until-d-def*)

lemma *UntilIdempotent*:
$\vdash f\ \mathcal{U}\ f = f$
using *UntilSUntil* **by** *fastforce*

lemma *UntilRightDistImp*:
$\vdash (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
proof −
have *1*: $\vdash (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h) =$
$((f \longrightarrow g)\ \mathcal{U}\ h \wedge f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
by *auto*
have *2*: $\vdash ((f \longrightarrow g)\ \mathcal{U}\ h \wedge f\ \mathcal{U}\ h) = ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ h$
by (*simp add*: *UntilAndDist int-iffD1 int-iffD2 int-iffI*)
have *3*: $\vdash ((f \longrightarrow g) \wedge f) = (f \wedge g)$
by *auto*
have *4*: $\vdash h = h$
by *auto*
have *5*: $\vdash ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ h = (f \wedge g)\ \mathcal{U}\ h$
using *3 4* **using** *UntilEqvUntil* **by** *blast*
have *6*: $\vdash (f \wedge g)\ \mathcal{U}\ h = (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h)$
by (*simp add*: *UntilAndDist*)
show *?thesis*
using *2 5 6* **by** *fastforce*
qed

lemma *FalseUntil*:
$\vdash \#False\ \mathcal{U}\ g = g$
by (*metis Prop10 Prop12 TrueW UntilUnrol int-simps*(*14*) *int-simps*(*21*) *int-simps*(*25*) *int-simps*(*3*) *inteq-reflection*)

lemma *UntilExclMid*:
$\vdash f\ \mathcal{U}\ g \lor f\ \mathcal{U}\ (\neg\ g)$
using *UntilOrDist UntilTrue* **by** *fastforce*

lemma *NotUntilImp*:
$\vdash (\neg\ f)\ \mathcal{U}\ (g\ \mathcal{U}\ h) \land f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h$
proof −
have *1*: $\vdash (\neg\ f)\ \mathcal{U}\ (g\ \mathcal{U}\ h) \longrightarrow (\neg\ f \lor g)\ \mathcal{U}\ h$
by (*simp add*: *UntilRightOr*)
have *2*: $\vdash (\neg\ f \lor g) = (f \longrightarrow g)$
by *auto*
have *3*: $\vdash h = h$
by *auto*
have *4*: $\vdash (\neg\ f \lor g)\ \mathcal{U}\ h = (f \longrightarrow g)\ \mathcal{U}\ h$
by (*simp add*: *2 UntilEqvUntil*)
have *5*: $\vdash (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
by (*simp add*: *UntilRightDistImp*)
have *6*: $\vdash (\neg\ f)\ \mathcal{U}\ (g\ \mathcal{U}\ h) \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
using *1 4 5* **by** *fastforce*
from *6* **show** *?thesis* **by** *auto*
qed

lemma *UntilNotImpa*:
$\vdash f\ \mathcal{U}\ ((\neg\ g)\ \mathcal{U}\ h) \land g\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h$
proof −
have *1*: $\vdash f\ \mathcal{U}\ ((\neg\ g)\ \mathcal{U}\ h) \longrightarrow (f \lor (\neg\ g))\ \mathcal{U}\ h$
by (*simp add*: *UntilRightOr*)
have *2*: $\vdash (f \lor (\neg\ g)) = (g \longrightarrow f)$
by *auto*
have *3*: $\vdash h = h$
by *auto*
have *4*: $\vdash (f \lor (\neg\ g))\ \mathcal{U}\ h = (g \longrightarrow f)\ \mathcal{U}\ h$
by (*simp add*: *2 UntilEqvUntil*)
have *5*: $\vdash (g \longrightarrow f)\ \mathcal{U}\ h \longrightarrow (\ g\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h)$
by (*simp add*: *UntilRightDistImp*)
have *6*: $\vdash f\ \mathcal{U}\ ((\neg\ g)\ \mathcal{U}\ h) \longrightarrow (\ g\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h)$
using *1 4 5* **by** *fastforce*
from *6* **show** *?thesis* **by** *auto*
qed

lemma *UntilNotUntilImp*:
$\vdash f\ \mathcal{U}\ g \land (\neg\ g)\ \mathcal{U}\ f \longrightarrow f$
proof −
have *1*: $\vdash f\ \mathcal{U}\ g \land (\neg\ g)\ \mathcal{U}\ f \longrightarrow f\ \mathcal{U}\ f$
using *UntilNotImp* **by** *auto*

have *2*: $\vdash$ $f \ \mathcal{U}\ f = f$
using *UntilIdempotent* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *AndNotUntilImp*:
$\vdash f \wedge (\neg\ f)\ \mathcal{U}\ g \longrightarrow g$
proof $-$
have *1*: $\vdash f = f\ \mathcal{U}\ f$
by (*simp add*: *UntilIdempotent int-iffD1 int-iffD2 int-iffI*)
have *2*: $\vdash g = \ \#False\ \mathcal{U}\ g$
by (*meson FalseUntil Prop11*)
have *3*: $\vdash f\ \mathcal{U}\ f \wedge (\neg\ f)\ \mathcal{U}\ g \longrightarrow \#False\ \mathcal{U}\ g$
by (*metis 1 FalseUntil UntilNotImp inteq-reflection*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilImpOr*:
$\vdash f\ \mathcal{U}\ g \longrightarrow f \vee g$
proof $-$
have $\vdash f \wedge f\ \mathcal{U}^s\ g \longrightarrow f \vee g$
by *force*
then show *?thesis*
by (*metis* (*no-types*) *FalseUntil Prop02 Prop03 UntilSUntil inteq-reflection*)
qed

lemma *UntilIntro*:
$\vdash g \longrightarrow f\ \mathcal{U}\ g$
proof $-$
have *1*: $\vdash g = \#False\ \mathcal{U}\ g$
by (*meson FalseUntil Prop11*)
have *2*: $\vdash \#False \longrightarrow f$
by *auto*
have *3*: $\vdash g \longrightarrow g$
by *auto*
have *4*: $\vdash \#False\ \mathcal{U}\ g \longrightarrow f\ \mathcal{U}\ g$
by (*simp add*: *UntilImpUntil*)
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *OrImpUntil*:
$\vdash f \wedge g \longrightarrow f\ \mathcal{U}\ g$
by (*simp add*: *Prop01 Prop05 UntilIntro*)

lemma *UntilAbsorp-a*:
$\vdash (f \vee f\ \mathcal{U}\ g) = (f \vee g)$
proof $-$
have *1*: $\vdash (f \vee f\ \mathcal{U}\ g) \longrightarrow f \vee g$
using *UntilImpOr* **by** *fastforce*
have *2*: $\vdash f \vee g \longrightarrow (f \vee f\ \mathcal{U}\ g)$

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

lemma *UntilAbsorp-b*:
$\vdash (f\ \mathcal{U}\ g \vee g) = f\ \mathcal{U}\ g$
using *UntilSUntil* **by** *fastforce*

lemma *UntilAbsorp-c*:
$\vdash (f\ \mathcal{U}\ g \wedge g) = g$
using *UntilIntro* **by** *fastforce*

lemma *UntilAbsorp-d*:
$\vdash (f\ \mathcal{U}\ g \vee (f \wedge g)) = f\ \mathcal{U}\ g$
using *UntilSUntil* **by** *fastforce*

lemma *UntilAbsorp-e*:
$\vdash (f\ \mathcal{U}\ g \wedge (f \vee g)) = f\ \mathcal{U}\ g$
by (*meson Prop10 Prop11 UntilImpOr*)

lemma *LeftUntilAbsorp*:
$\vdash f\ \mathcal{U}\ (f\ \mathcal{U}\ g) = f\ \mathcal{U}\ g$
by (*meson Prop11 UntilUntil*)

lemma *RightUntilAbsorp*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*metis Prop11 UntilAbsorp-b UntilAbsorp-c UntilImpOr UntilRightAnd UntilUntil inteq-reflection*)

lemma *UntilAbsorpAndDiamond*:
$\vdash (f\ \mathcal{U}\ g \wedge \Diamond\ g) = f\ \mathcal{U}\ g$
by (*metis DiamondEqvTrueUntil Prop11 Prop12 UntilAbsorp-c UntilRightAnd int-simps*(*17*)
inteq-reflection)

lemma *UntilAbsorpOrDiamond*:
$\vdash (f\ \mathcal{U}\ g \vee \Diamond\ g) = \Diamond\ g$
using *UntilAbsorpAndDiamond* **by** *fastforce*

lemma *UntilAbsorpDiamond*:
$\vdash f\ \mathcal{U}\ (\Diamond\ g) = \Diamond\ g$
using *DiamondDiamondEqvDiamond UntilAbsorpOrDiamond UntilAbsorp-b* **by** *fastforce*

lemma *UntilImpDiamond*:
$\vdash f\ \mathcal{U}\ g \longrightarrow \Diamond\ g$
using *UntilAbsorpAndDiamond* **by** *fastforce*

lemma *UntilInduction-a*:
$\vdash \Box(f \longrightarrow ((\bigcirc\ f) \wedge g) \vee h) \longrightarrow (f \longrightarrow \Box\ g \vee g\ \mathcal{U}\ h)$
proof −
have *1*: $\vdash (\Box\ g \vee g\ \mathcal{U}\ h) = g\ \mathcal{W}\ h$

by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash (f \longrightarrow \Box\ g \vee g\ \mathcal{U}\ h) = (\ (\neg\ h)\ \mathcal{U}\ (\neg\ g \wedge \neg h) \longrightarrow \neg\ f)$
using *1 WaitNotDistUntil* **by** *fastforce*
have *3*: $\vdash \Box(\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f)))\longrightarrow \neg f) \longrightarrow (\ (\neg\ h)\ \mathcal{U}\ (\neg\ g \wedge \neg h) \longrightarrow \neg\ f)$
using *LFPUntil* **by** *blast*
have *4*: $\vdash (f \longrightarrow ((\bigcirc\ f) \wedge g) \vee h) \longrightarrow (\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f)))\longrightarrow \neg f)$
using *UntilInduction-a-help* **by** *simp*
have *5*: $\vdash \Box(f \longrightarrow ((\bigcirc\ f) \wedge g) \vee h) \longrightarrow \Box(\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f)))\longrightarrow \neg f)$
using *4* **by** (*rule ImpBoxRule*)
show *?thesis*
using *2 3 5* **by** *fastforce*
qed

lemma *UntilInduction-b*:
$\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow (f \longrightarrow \Box\ f \vee f\ \mathcal{U}\ g)$
proof –
have *1*: $\vdash (\Box\ f \vee f\ \mathcal{U}\ g) = f\ \mathcal{W}\ g$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash (f \longrightarrow \Box\ f \vee f\ \mathcal{U}\ g) = (\ (\neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg\ g) \longrightarrow \neg\ f)$
using *1 WaitNotDistUntil* **by** *fastforce*
have *3*: $\vdash \Box\ (\ ((\neg\ f \wedge \neg\ g) \vee (\neg g \wedge \bigcirc\ (\neg\ f))) \longrightarrow \neg\ f) \longrightarrow (\ (\neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg\ g) \longrightarrow \neg\ f)$
using *LFPUntil* **by** *blast*
have *4*: $\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow \quad \Box(\ ((\neg\ f \wedge \neg\ g) \vee (\neg g \wedge \bigcirc\ (\neg\ f))) \longrightarrow \neg\ f)$
using *UntilInduction-b-help* **by** *simp*
show *?thesis*
using *2 3 4* **by** *fastforce*
qed

lemma *AlwaysImpNotUntilNot*:
$\vdash \Box\ f \longrightarrow \neg(g\ \mathcal{U}\ (\neg\ f))$
by (*simp add*: *UntilImpDiamond always-d-def*)

lemma *UntilAndImp*:
$\vdash \Box\ f \wedge \Diamond\ g \longrightarrow f\ \mathcal{U}\ g$
proof –
have *1*: $\vdash \Diamond\ g = \#True\ \mathcal{U}\ g$
by (*simp add*: *DiamondEqvTrueUntil*)
have *2*: $\vdash \Box\ f \wedge \#True\ \mathcal{U}\ g \longrightarrow (f \wedge \#True)\ \mathcal{U}\ (f \wedge g)$
using *UntilAlwaysAndDist* **by** *blast*
have *3*: $\vdash (f \wedge \#True)\ \mathcal{U}\ (f \wedge g) = f\ \mathcal{U}\ (f \wedge g)$
by *simp*
have *4*: $\vdash f\ \mathcal{U}\ (f \wedge g) \longrightarrow (f\ \mathcal{U}\ f)\ \mathcal{U}\ g$
by (*simp add*: *UntilRightAnd*)
have *5*: $\vdash (f\ \mathcal{U}\ f) = f$
by (*simp add*: *UntilIdempotent*)
have *6*: $\vdash (f\ \mathcal{U}\ f)\ \mathcal{U}\ g\ = f\ \mathcal{U}\ g$
by (*simp add*: *5 UntilEqvUntil*)
show *?thesis*
by (*metis 1 2 3 4 5 inteq-reflection lift-imp-trans*)
qed

lemma *UntilCatRule*:
$\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow (f \longrightarrow (g\ \mathcal{U}\ i))$
proof –
have *1*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow \Box\ (f \longrightarrow g\ \mathcal{U}\ h)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *2*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow \Box\ (h \longrightarrow g\ \mathcal{U}\ i)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *3*: $\vdash \Box\ (h \longrightarrow g\ \mathcal{U}\ i) \longrightarrow \Box\ (\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ (g\ \mathcal{U}\ i))$
by (*metis BoxEqvBoxBox BoxImpBoxRule UntilRightMono inteq-reflection*)
have *4*: $\vdash \Box\ (\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ (g\ \mathcal{U}\ i)) \longrightarrow \Box(\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
by (*metis BoxEqvBoxBox UntilUntil int-iffD1 inteq-reflection*)
have *5*: $\vdash\ \ \Box\ (f \longrightarrow g\ \mathcal{U}\ h) \longrightarrow (f \longrightarrow g\ \mathcal{U}\ h)$
by (*simp add*: *BoxElim*)
have *6*: $\vdash \Box(\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i) \longrightarrow (g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
by (*simp add*: *BoxElim*)
have *7*: $\vdash (f \longrightarrow g\ \mathcal{U}\ h) \wedge (g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i) \longrightarrow (f \longrightarrow g\ \mathcal{U}\ i)$
by *auto*
have *8*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow$
$(f \longrightarrow g\ \mathcal{U}\ h) \wedge (g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
using *1 2 3 4 5 6* **by** *fastforce*
from *7 8* **show** *?thesis* **by** *auto*
qed

lemma *UntilStrengthen*:
$\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
proof –
have *1*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g)$
by (*metis BoxImpBoxRule Prop01 UntilLeftMono int-simps*(*12*) *int-simps*(*29*) *int-simps*(*9*) *inteq-reflection lift-imp-trans*)
have *2*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
by (*metis BoxImpBoxRule Prop01 Prop05 UntilIdempotent UntilIntro UntilRightMono inteq-reflection lift-imp-trans*)
have *3*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g) \wedge (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
using *1 2* **by** *fastforce*
have *4*: $\vdash (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g) \wedge (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
by *auto*
from *3 4* **show** *?thesis* **by** *auto*
qed

lemma *UntilInduction*:
$\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc\ f) \longrightarrow (f \longrightarrow \neg(h\ \mathcal{U}\ g))$
proof –
have *1*: $\vdash \Box\ (\neg\ g) \longrightarrow \neg(h\ \mathcal{U}\ g)$
by (*simp add*: *UntilImpDiamond always-d-def*)
have *2*: $\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc\ f) \longrightarrow \Box(g \vee \#True \wedge \bigcirc\ (\neg\ f) \longrightarrow \neg\ f)$
using *UntilInduction-help* **by** *simp*
have *3*: $\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc\ f) \longrightarrow (\#True\ \mathcal{U}\ g \longrightarrow \neg\ f)$

using *2 LFPUntil*[*of g LIFT*(*#True*) *LIFT*(¬ *f*)]
by *fastforce*
have *4*: $\vdash$ (*#True* $\mathcal{U}$ *g* $\longrightarrow$ ¬ *f*) $\longrightarrow$ (*f* $\longrightarrow$ ¬(*#True* $\mathcal{U}$ *g*))
by *auto*
have *5*: $\vdash$ ¬(*#True* $\mathcal{U}$ *g*) = $\Box$ (¬ *g*)
using *BoxEqvFalseRelease NotUntilRelease inteq-reflection* by *fastforce*
from *5 4 3 1* show *?thesis* by *fastforce*
qed

lemma *UntilBoxImphelp1*:
$\vdash$ *f* $\mathcal{U}$ $\Box$*g* $\longrightarrow$ *f* $\mathcal{U}$ *g*
by (*meson BoxElim BoxGen MP UntilRightMono*)

lemma *UntilBoxImphelp2*:
$\vdash$ *more* ∧ *f* $\mathcal{U}$ $\Box$*g* $\longrightarrow$ $\bigcirc$ (*f* $\mathcal{U}$ $\Box$*g*)
proof −
have *f4*: $\vdash$ *wnext* (*f* $\mathcal{U}$ $\Box$*g*) = (*empty* ∨ $\bigcirc$ (*f* $\mathcal{U}$ $\Box$*g*))
by (*meson WnextEqvEmptyOrNext*)
have *f5*: $\vdash$ $\Box$ *g* = (*g* ∧ *wnext* ($\Box$*g*))
by (*metis* (*no-types*) *BoxEqvAndWnextBox*)
have *f6*: $\vdash$*f* $\mathcal{U}$ $\Box$*g* = ($\Box$*g* ∨ *f* ∧ $\bigcirc$ (*f* $\mathcal{U}$ $\Box$*g*))
by (*meson UntilUnrol*)
have $\vdash$ *g* ∧ *wnext* ($\Box$*g*) $\longrightarrow$ *empty* ∨ $\bigcirc$ (*f* $\mathcal{U}$ $\Box$*g*)
by (*metis NextUntil Prop01 Prop05 Prop08 UntilIntro WnextEqvEmptyOrNext int-iffD1 inteq-reflection*)
then have $\vdash$ *more* ∧ *f* $\mathcal{U}$ $\Box$*g* $\longrightarrow$ *empty* ∨ $\bigcirc$ (*f* $\mathcal{U}$ $\Box$*g*)
using *f6 f5* by *fastforce*
then show *?thesis*
using *f4* using *WnextAndMoreEqvNext* by *fastforce*
qed

lemma *UntilBoxImp*:
$\vdash$ *f* $\mathcal{U}$ ($\Box$ *g*) $\longrightarrow$ $\Box$(*f* $\mathcal{U}$ *g*)
using *BoxIntro*[*of LIFT*(*f* $\mathcal{U}$ ($\Box$ *g*)) *LIFT*(*f* $\mathcal{U}$ *g*)]
by (*simp add*: *UntilBoxImphelp1 UntilBoxImphelp2*)

lemma *UntilBoxEqvBox*:
$\vdash$ *f* $\mathcal{U}$ ($\Box$ *f*) = $\Box$ *f*
proof −
have *1*: $\vdash$ *f* $\mathcal{U}$ ($\Box$ *f*) $\longrightarrow$ $\Box$(*f* $\mathcal{U}$ *f*)
using *UntilBoxImp*[*of f f*] by *auto*
have *2*: $\vdash$ $\Box$(*f* $\mathcal{U}$ *f*) = $\Box$ *f*
by (*simp add*: *BoxEqvBox UntilIdempotent*)
have *3*: $\vdash$ $\Box$ *f* $\longrightarrow$ *f* $\mathcal{U}$ ($\Box$ *f*)
by (*simp add*: *UntilIntro*)
from *1 2 3* show *?thesis* by *fastforce*
qed

lemma *UntilRightStrengthen*:

$\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow f\ \mathcal{U}\ (g\ \mathcal{U}\ h)$
by (*meson BoxGen MP OrImpUntil UntilRightMono*)

lemma *UntilLeftStrengthen*:
$\vdash (f \wedge g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ g)\ \mathcal{U}\ h$
by (*simp add*: *OrImpUntil UntilImpUntil*)

lemma *UntilLeftAndOrder*:
$\vdash (f \wedge g)\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ (g\ \mathcal{U}\ h)$
by (*metis Prop05 Prop07 ReleaseUntil RightUntilAbsorp UntilAbsorp-c UntilAndDist UntilRightStrengthen inteq-reflection*)

lemma *UntilFrameNext*:
$\vdash \Box\ f \longrightarrow (\bigcirc\ g \longrightarrow \bigcirc\ (f\ \mathcal{U}\ g))$
by (*simp add*: *NextImpNext Prop01 Prop05 Prop09 UntilIntro*)

lemma *UntilFrameDiamond*:
$\vdash \Box\ f \longrightarrow (\Diamond\ g \longrightarrow \Diamond\ (f\ \mathcal{U}\ g))$
by (*meson NowImpDiamond Prop09 UntilAndImp lift-imp-trans*)

lemma *UntilFrameBox*:
$\vdash \Box\ f \longrightarrow (\Box\ g \longrightarrow \Box\ (f\ \mathcal{U}\ g))$
by (*simp add*: *BoxAndBoxImpBoxRule OrImpUntil Prop09*)

lemma *UntilAndRule*:
$\vdash f\ \mathcal{U}\ g = (f \wedge \neg g)\ \mathcal{U}\ g$
proof −
have *1*: $\vdash (f \wedge \neg g)\ \mathcal{U}\ g \longrightarrow f\ \mathcal{U}\ g$
using *UntilAndDist* **by** *fastforce*
show *?thesis* **by** (*simp add*: *1 UntilImpNot int-iffI*)
qed

lemma *UntilWait*:
$\vdash f\ \mathcal{U}\ g = (f\ \mathcal{W}\ g \wedge \Diamond\ g)$
proof −
have *1*: $\vdash f\ \mathcal{U}\ g \longrightarrow f\ \mathcal{W}\ g \wedge \Diamond\ g$
by (*simp add*: *Prop05 Prop12 UntilImpDiamond wait-d-def*)
have *2*: $\vdash (f\ \mathcal{W}\ g \wedge \Diamond\ g) = ((\Box\ f \vee f\ \mathcal{U}\ g) \wedge \Diamond\ g)$
by (*auto simp add*: *wait-d-def*)
have *3*: $\vdash ((\Box\ f \vee f\ \mathcal{U}\ g) \wedge \Diamond\ g) = ((\Box\ f \wedge \Diamond g) \vee (f\ \mathcal{U}\ g \wedge \Diamond\ g))$
by *auto*
have *4*: $\vdash (\Box\ f \wedge \Diamond g) \longrightarrow f\ \mathcal{U}\ g$
by (*simp add*: *UntilAndImp*)
have *5*: $\vdash (f\ \mathcal{U}\ g \wedge \Diamond\ g) \longrightarrow f\ \mathcal{U}\ g$
by *auto*
show *?thesis*
using *1 2 4* **by** *fastforce*
qed

lemma *WaitUntilb*:

$\vdash f\ \mathcal{W}\ \ g = (\Box\ (\ f \wedge \neg\ g) \vee f\ \mathcal{U}\ g)$
proof −
 have *1*: $\vdash f\ \mathcal{W}\ g = (f \wedge \neg\ g)\ \mathcal{W}\ g$
  by (*simp add*: *WaitAndRule*)
 have *2*: $\vdash (f \wedge \neg\ g)\ \mathcal{W}\ g = (\Box\ (f \wedge \neg\ g) \vee (f \wedge \neg\ g)\ \mathcal{U}\ g)$
  by (*auto simp add*: *wait-d-def*)
 have *3*: $\vdash (f \wedge \neg\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
  by (*meson Prop11 UntilAndRule*)
 show *?thesis*
 using *1 2 3* **by** *fastforce*
qed

lemma *UntilNotDistWait*:
$\vdash (\neg\ (f\ \mathcal{U}\ g)) = (\neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
proof −
 have *1*: $\vdash (\neg\ ((\neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g))) = (\neg(\neg\ f \wedge \neg\ g))\ \mathcal{U}\ (\ \neg\ (\neg\ g) \wedge \neg\ (\neg\ f \wedge \neg\ g))$
  using *WaitNotDistUntil* **by** *blast*
 have *2*: $\vdash (\neg(\neg\ f \wedge \neg\ g)) = (f \vee g)$
   by *auto*
 have *3*: $\vdash (\ \neg\ (\neg\ g) \wedge \neg\ (\neg\ f \wedge \neg\ g)) = g$
  by *auto*
 have *4*: $\vdash (\neg(\neg\ f \wedge \neg\ g))\ \mathcal{U}\ (\ \neg\ (\neg\ g) \wedge \neg\ (\neg\ f \wedge \neg\ g)) =$
  $(f \vee g)\ \mathcal{U}\ g$
  using *2 3 UntilEqvUntil* **by** *blast*
 have *5*: $\vdash (f \vee g)\ \mathcal{U}\ g = ((f \vee g) \wedge \neg g)\ \mathcal{U}\ g$
  by (*simp add*: *UntilAndRule*)
 have *6*: $\vdash ((f \vee g) \wedge \neg g) = (f \wedge \neg\ g)$
  by *auto*
 have *7*: $\vdash ((f \vee g) \wedge \neg g)\ \mathcal{U}\ g = (f \wedge \neg g)\ \mathcal{U}\ g$
  using *6 inteq-reflection* **by** *fastforce*
 have *8*: $\vdash (f \wedge \neg g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
   by (*meson Prop11 UntilAndRule*)
 have *9*: $\vdash f\ \mathcal{U}\ g = (\neg\ ((\neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)))$
   using *1 4 5 7 8* **by** *fastforce*
 show *?thesis* **using** *9* **by** *auto*
qed

lemma *UntilImpWait*:
$\vdash f\ \mathcal{U}\ g \longrightarrow f\ \mathcal{W}\ g$
by (*meson Prop03 WaitUntilb*)

lemma *WaitAndDist*:
$\vdash (\Box\ f \wedge g\ \mathcal{W}\ h) \longrightarrow (f \wedge g)\ \mathcal{W}\ (f \wedge h)$
proof −
 have *1*: $\vdash (\Box\ f \wedge g\ \mathcal{W}\ h) = (\Box\ f \wedge (\Box\ g \vee g\ \mathcal{U}\ h))$
  by (*auto simp add*: *wait-d-def*)
 have *2*: $\vdash (\Box\ f \wedge (\Box\ g \vee g\ \mathcal{U}\ h)) = ((\Box f \wedge \Box g) \vee (\Box\ f \wedge g\ \mathcal{U}\ h))$
  by *auto*
 have *3*: $\vdash (\Box f \wedge \Box g) = \Box(f \wedge g)$
  by (*simp add*: *BoxAndBoxEqvBoxRule*)

have *4*: $\vdash (\Box\ f \wedge g\ \mathcal{U}\ h) \longrightarrow (f \wedge g)\ \mathcal{U}\ (f \wedge h)$
by (*simp add*: *UntilAlwaysAndDist*)
have *5*: $\vdash ((\Box f \wedge \Box g) \vee (\Box\ f \wedge g\ \mathcal{U}\ h)) \longrightarrow \Box(f \wedge g) \vee (f \wedge g)\ \mathcal{U}\ (f \wedge h)$
using *3 4* **by** *fastforce*
have *6*: $\vdash (\Box(f \wedge g) \vee (f \wedge g)\ \mathcal{U}\ (f \wedge h)) = (f \wedge g)\ \mathcal{W}\ (f \wedge h)$
by (*auto simp add*: *wait-d-def*)
show *?thesis*
using *1 5 6* **by** *fastforce*
qed

lemma *WaitDiamondOr*:
$\vdash f\ \mathcal{W}\ (\Diamond\ g) = (\Box\ f \vee \Diamond\ g)$
proof −
have *1*: $\vdash f\ \mathcal{W}\ (\Diamond\ g) = (\Box\ f \vee f\ \mathcal{U}\ (\Diamond\ g))$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash f\ \mathcal{U}\ (\Diamond\ g) = \Diamond\ g$
by (*simp add*: *UntilAbsorpDiamond*)
show *?thesis* **using** *1 2 Prop06* **by** *blast*
qed

lemma *WaitBoxImp*:
$\vdash f\ \mathcal{W}\ (\Box\ g) \longrightarrow \Box\ (\ f\ \mathcal{W}\ g)$
proof −
have *1*: $\vdash f\ \mathcal{W}\ (\Box\ g) = (\Box\ f \vee f\ \mathcal{U}\ (\Box\ g))$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash \Box\ f = \Box\ (\Box\ f)$
by (*simp add*: *BoxEqvBoxBox*)
have *3*: $\vdash f\ \mathcal{U}\ (\Box\ g) \longrightarrow \Box(f\ \mathcal{U}\ g)$
by (*simp add*: *UntilBoxImp*)
have *4*: $\vdash (\Box\ f \vee f\ \mathcal{U}\ (\Box\ g)) \longrightarrow (\Box\ (\Box\ f) \vee \Box(f\ \mathcal{U}\ g))$
using *2 3* **by** *fastforce*
have *5*: $\vdash \Box\ (\Box\ f) \longrightarrow \Box(\ \Box\ f \vee f\ \mathcal{U}\ g)$
by (*metis BoxImpBoxRule Prop08 UntilIdempotent UntilIntro int-simps*(*11*) *int-simps*(*25*) *inteq-reflection*)
have *6*: $\vdash \Box(f\ \mathcal{U}\ g) \longrightarrow \Box(\ \Box\ f \vee f\ \mathcal{U}\ g)$
by (*metis BoxImpBoxRule UntilImpWait wait-d-def*)
have *7*: $\vdash (\Box\ (\Box\ f) \vee \Box(f\ \mathcal{U}\ g)) \longrightarrow \Box(\ \Box\ f \vee f\ \mathcal{U}\ g)$
using *5 6* **by** *fastforce*
have *6*: $\vdash \Box(\ \Box\ f \vee f\ \mathcal{U}\ g) = \Box\ (\ f\ \mathcal{W}\ g)$
by (*simp add*: *wait-d-def*)
show *?thesis*
by (*metis 4 7 lift-imp-trans wait-d-def*)
qed

lemma *WaitAbsorptionBox*:
$\vdash f\ \mathcal{W}\ (\Box\ f) = \Box\ f$
by (*metis Prop02 Prop11 UntilBoxEqvBox UntilImpWait inteq-reflection wait-d-def*)

lemma *BoxImpWait*:

$\vdash \Box\ f \longrightarrow f\ \mathcal{W}\ g$
by (*auto simp add*: *wait-d-def*)

lemma *WaitDistNext*:
$\vdash \bigcirc\ (\ f\ \mathcal{W}\ g) = (\bigcirc\ f)\ \mathcal{W}\ (\bigcirc\ g)$
— nitpick finds counterexample
oops

lemma *WnextAlwaysEqvAlwaysWnext*:
$\vdash$ *wnext* $(\Box\ f) = \Box($*wnext* $f)$
by (*metis* (*no-types*, *lifting*) *NextDiamondEqvDiamondNext always-d-def int-eq int-simps*(4) *wnext-d-def*)

lemma *WaitExpand*:
$\vdash f\ \mathcal{W}\ g = (g \lor (f \land \bigcirc(f\ \mathcal{W}\ g)))$
— nitpick finds counterexample
oops

lemma *WaitExpand*:
$\vdash f\ \mathcal{W}\ g = (g \lor (f \land \mathit{wnext}(f\ \mathcal{W}\ g)))$
proof −
have *1*: $\vdash\ f\ \mathcal{W}\ g = (\Box\ f \lor f\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash \Box\ f = (f \land \mathit{wnext}(\Box\ f))$
by (*simp add*: *BoxEqvAndWnextBox*)
have *3*: $\vdash f\ \mathcal{U}\ g = (g \lor (f \land \bigcirc\ (f\ \mathcal{U}\ g)))$
using *UntilUnrol* **by** *blast*
have *4*: $\vdash (f \land \mathit{wnext}(\Box\ f)) = (f \land (\mathit{empty} \lor \bigcirc\ (\Box\ f)))$
using *2 BoxEqvAndEmptyOrNextBox* **by** *fastforce*
have *5*: $\vdash\ \mathit{wnext}(f\ \mathcal{W}\ g) = (\mathit{empty} \lor \bigcirc\ (f\ \mathcal{W}\ g))$
using *WnextEqvEmptyOrNext* **by** *blast*
have *6*: $\vdash (f \land (\mathit{empty} \lor \bigcirc\ (\Box\ f))) = (\ (f \land \mathit{empty}) \lor (f \land \bigcirc\ (\Box\ f)))$
by *auto*
have *7*: $\vdash ((\ (f \land \mathit{empty}) \lor (f \land \bigcirc\ (\Box\ f))) \lor$
$(g \lor (f \land \bigcirc\ (f\ \mathcal{U}\ g)))\ ) =$
$(g \lor (f \land (\mathit{empty} \lor \bigcirc\ (\Box f)\ \ \lor \bigcirc\ (f\ \mathcal{U}\ g)\ )\ ))$
by *auto*
have *8*: $\vdash (\bigcirc\ (\Box f)\ \ \lor \bigcirc\ (f\ \mathcal{U}\ g)) = \bigcirc(\Box f \lor f\ \mathcal{U}\ g)$
by (*metis ChopOrEqv Prop11 next-d-def*)
show *?thesis*
by (*metis 1 2 3 4 5 6 7 8 inteq-reflection*)
qed

lemma *WaitExclMid*:
$\vdash f\ \mathcal{W}\ g \lor f\ \mathcal{W}\ (\neg\ g)$
using *WaitExpand*
proof −
have *1*: $\vdash f\ \mathcal{W}\ g = (g \lor f \land \mathit{wnext}\ (f\ \mathcal{W}\ g))$
by (*simp add*: *WaitExpand*)
have *2*: $\vdash f\ \mathcal{W}\ (\neg\ g) = ((\neg\ g) \lor f \land \mathit{wnext}\ (\ f\ \mathcal{W}\ (\neg\ g)))$

by (*simp add*: *WaitExpand*)
have *3*: $\vdash (f\ \mathcal{W}\ g \lor f\ \mathcal{W}\ (\neg\ g)) =$
$((g \lor f \land \mathit{wnext}\ (f\ \mathcal{W}\ g)) \lor ((\neg\ g) \lor f \land \mathit{wnext}\ (f\ \mathcal{W}\ (\neg\ g))))$
using *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** *fastforce*
qed

lemma *WaitleftZero*:
$\vdash \#\mathit{True}\ \mathcal{W}\ g$
by (*meson BoxGen BoxImpWait MP TrueW*)

lemma *WaitLeftDistOr*:
$\vdash f\ \mathcal{W}\ (g \lor h) = (f\ \mathcal{W}\ g \lor f\ \mathcal{W}\ h)$
proof −
have *1*: $\vdash f\ \mathcal{W}\ (g \lor h) = (\Box\ f \lor f\ \mathcal{U}\ (g \lor h))$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash (f\ \mathcal{W}\ g \lor f\ \mathcal{W}\ h) = ((\Box\ f \lor f\ \mathcal{U}\ g) \lor (\Box\ f \lor f\ \mathcal{U}\ h))$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash f\ \mathcal{U}\ (g \lor h) = (f\ \mathcal{U}\ g \lor f\ \mathcal{U}\ h)$
by (*simp add*: *UntilOrDist*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *WaitRightDistOr*:
$\vdash f\ \mathcal{W}\ h \lor g\ \mathcal{W}\ h \longrightarrow (f \lor g)\ \mathcal{W}\ h$
proof −
have *0*: $\vdash \Box\ g \longrightarrow \Box\ (f \lor g)$
by (*simp add*: *BoxImpBoxRule intI*)
have *1*: $\vdash \Box f \longrightarrow \Box\ (f \lor g)$
by (*simp add*: *BoxImpBoxRule intI*)
have *11*: $\vdash \Box\ f \lor \Box\ g \longrightarrow \Box\ (f \lor g)$
using *0 1 Prop02* **by** *blast*
have *2*: $\vdash f\ \mathcal{W}\ h = (\Box\ f \lor f\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash (f \lor g)\ \mathcal{W}\ h = (\Box\ (f \lor g) \lor (f \lor g)\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *4*: $\vdash g\ \mathcal{W}\ h = (\Box g \lor g\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *5*: $\vdash f\ \mathcal{U}\ h \lor g\ \mathcal{U}\ h \longrightarrow (f \lor g)\ \mathcal{U}\ h$
using *UntilRightDistOr* **by** *simp*
have *6*: $\vdash (f\ \mathcal{W}\ h \lor g\ \mathcal{W}\ h) = ((\Box\ f \lor \Box\ g) \lor (f\ \mathcal{U}\ h \lor g\ \mathcal{U}\ h))$
using *2 4* **by** *fastforce*
from *11 5 6 3* **show** *?thesis*
using *BoxImpWait* **by** *fastforce*
qed

lemma *WaitOrRule*:
$\vdash f \ \mathcal{W}\ g = (f \lor g)\ \mathcal{W}\ g$
proof −
 have *1*: $\vdash f \ \mathcal{W}\ g \longrightarrow (f \lor g)\ \mathcal{W}\ g$
  by (*metis* (*no-types*, *lifting*) *Prop03 Prop10 UntilAbsorp-a WaitNotDistUntil int-iffD1 int-simps*(*14*) *int-simps*(*32*) *int-simps*(*33*) *inteq-reflection*)
 have *2*: $\vdash (f \lor g)\ \mathcal{W}\ g \longrightarrow f\ \mathcal{W}\ g$
  by (*metis* (*no-types*, *lifting*) *Prop03 Prop10 WaitNotDistUntil int-iffD2 int-simps*(*14*) *int-simps*(*32*) *int-simps*(*33*) *inteq-reflection*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilOrRule*:
$\vdash f \ \mathcal{U}\ g = (f \lor g)\ \mathcal{U}\ g$
by (*metis UntilWait WaitOrRule inteq-reflection*)

lemma *WaitRule*:
$\vdash (\neg f)\ \mathcal{W}\ f$
by (*metis BoxGen BoxImpWait MP WaitOrRule int-eq-true int-simps*(*29*) *inteq-reflection*)

lemma *UntilRule*:
$\vdash (\neg f)\ \mathcal{U}\ f = \diamond f$
using *DiamondEqvTrueUntil UntilOrRule inteq-reflection* **by** *fastforce*

lemma *DiamondUntilImpRule*:
$\vdash \diamond f \longrightarrow (f \longrightarrow g)\ \mathcal{U}\ f$
using *UntilWait WaitImpRule* **by** *fastforce*

lemma *WaitNotDist*:
$\vdash (\neg (f\ \mathcal{W}\ g)) = (f \land \neg g)\ \mathcal{U}\ (\neg f \land \neg g)$
proof −
 have *1*: $\vdash (\neg (f\ \mathcal{W}\ g)) = (\neg g\ )\ \mathcal{U}\ (\neg f \land \neg g)$
  using *WaitNotDistUntil* **by** *blast*
 have *2*: $\vdash (\neg g\ )\ \mathcal{U}\ (\neg f \land \neg g) = (\neg g \land \neg(\neg f \land \neg g))\ \mathcal{U}\ (\neg f \land \neg g)$
  using *UntilAndRule* **by** *blast*
 have *3*: $\vdash (\neg g \land \neg(\neg f \land \neg g)) = (f \land \neg g)$
  by *auto*
 have *4*: $\vdash (\neg g \land \neg(\neg f \land \neg g))\ \mathcal{U}\ (\neg f \land \neg g) = (f \land \neg g)\ \mathcal{U}\ (\neg f \land \neg g)$
  using *3 inteq-reflection* **by** *force*
 show *?thesis* **using** *1 2 4* **by** *fastforce*
qed

lemma *UntilNotDist*:
$\vdash (\neg (f\ \mathcal{U}\ g)) = (f \land \neg g)\ \mathcal{W}\ (\neg f \land \neg g)$
proof −
 have *1*: $\vdash (\neg (f\ \mathcal{U}\ g)) = (\neg g\ )\ \mathcal{W}\ (\neg f \land \neg g)$
  using *UntilNotDistWait* **by** *blast*
 have *2*: $\vdash\ (\neg g\ )\ \mathcal{W}\ (\neg f \land \neg g) = (\neg g \land \neg(\neg f \land \neg g))\ \mathcal{W}\ (\neg f \land \neg g)$
  by (*simp add*: *WaitAndRule*)

have *3*: $\vdash (\neg\, g \wedge \neg(\neg\, f \wedge \neg\, g)) = (f \wedge \neg\, g)$
  by *auto*
have *4*: $\vdash (\neg\, g \wedge \neg(\neg\, f \wedge \neg\, g))\ \mathcal{W}\ (\neg\, f \wedge \neg\, g) = (f \wedge \neg\, g)\ \mathcal{W}\ (\neg\, f \wedge \neg\, g)$
  using *3 inteq-reflection* **by** *force*
show *?thesis* **using** *1 2 4* **by** *fastforce*
qed

lemma *UntilDuala*:
$\vdash (\neg\, (\,(\neg\, f)\ \mathcal{U}\ (\neg\, g))) = g\ \mathcal{W}\ (f \wedge g)$
proof –
have *1*: $\vdash (\neg\, (\,(\neg\, f)\ \mathcal{U}\ (\neg\, g))) = (\neg\, f \wedge \neg\, (\neg\, g))\ \mathcal{W}\ (\,\neg(\neg\, f) \wedge \neg\, (\neg\, g))$
  using *UntilNotDist* **by** *blast*
have *2*: $\vdash (\neg\, f \wedge g)\ \mathcal{W}\ (\,f \wedge g) = g\ \mathcal{W}\ (\,f \wedge g)$
  using *1 UntilNotDistWait int-eq* **by** *fastforce*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *UntilDualb*:
$\vdash (\neg\, (\,(\neg\, f)\ \mathcal{U}\ (\neg\, g))) = (\neg\, f \wedge g)\ \mathcal{W}\ (f \wedge g)$
proof –
have *1*: $\vdash (\neg\, (\,(\neg\, f)\ \mathcal{U}\ (\neg\, g))) = (\neg\, f \wedge \neg\, (\neg\, g))\ \mathcal{W}\ (\,\neg(\neg\, f) \wedge \neg\, (\neg\, g))$
  using *UntilNotDist* **by** *blast*
show *?thesis*
using *1* **by** *auto*
qed

lemma *WaitDuala*:
$\vdash (\neg\, (\,(\neg\, f)\ \mathcal{W}\ (\neg\, g))) = g\ \mathcal{U}\ (f \wedge g)$
proof –
have *1*: $\vdash (\neg\, (\,(\neg\, f)\ \mathcal{W}\ (\neg\, g))) = (\neg\, f \wedge \neg\, (\neg\, g))\ \mathcal{U}\ (\,\neg(\neg\, f) \wedge \neg\, (\neg\, g))$
  using *WaitNotDist* **by** *blast*
have *2*: $\vdash (\neg\, f \wedge g)\ \mathcal{U}\ (\,f \wedge g) = g\ \mathcal{U}\ (\,f \wedge g)$
  using *1 WaitNotDistUntil int-eq* **by** *fastforce*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *WaitDualb*:
$\vdash (\neg\, (\,(\neg\, f)\ \mathcal{W}\ (\neg\, g))) = (\neg\, f \wedge g)\ \mathcal{U}\ (f \wedge g)$
proof –
have *1*: $\vdash (\neg\, (\,(\neg\, f)\ \mathcal{W}\ (\neg\, g))) = (\neg\, f \wedge \neg\, (\neg\, g))\ \mathcal{U}\ (\,\neg(\neg\, f) \wedge \neg\, (\neg\, g))$
  using *WaitNotDist* **by** *blast*
show *?thesis* **using** *1* **by** *auto*
qed

lemma *WaitIdempotent*:
$\vdash f\ \mathcal{W}\ f = f$
by (*meson BoxElim Prop02 Prop12 UntilIdempotent UntilImpWait UntilIntro WaitUntilb int-iffD1 int-iffI lift-imp-trans*)

lemma *WaitRightZero*:
$\vdash f\ \mathcal{W}\ \#True$
by (*meson MP TrueW UntilImpWait UntilIntro*)

lemma *WaitLeftIdentity*:
$\vdash \#False\ \mathcal{W}\ g = g$
by (*metis* (*no-types*, *lifting*) *UntilAbsorp-c UntilNotDistWait WaitDuala WaitIdempotent WaitSRelease int-eq int-simps*(*17*) *int-simps*(*3*) *srelease-d-def*)

lemma *WaitImpOr*:
$\vdash f\ \mathcal{W}\ g \longrightarrow f \lor g$
by (*metis Prop03 WaitIdempotent WaitLeftDistOr WaitOrRule inteq-reflection*)

lemma *BoxOrImpWait*:
$\vdash \Box(f \lor g) \longrightarrow f\ \mathcal{W}\ g$
using *BoxImpWait WaitOrRule* **by** *fastforce*

lemma *BoxImpImpWait*:
$\vdash \Box(\neg\ g \longrightarrow f) \longrightarrow f\ \mathcal{W}\ g$
proof −
have *1*: $\vdash (\neg\ g \longrightarrow f) = (f \lor g)$
by *auto*
have *2*: $\vdash \Box\ (\neg\ g \longrightarrow f) = \Box(f \lor g)$
using *1 BoxEqvBox* **by** *blast*
show *?thesis* **using** *2 BoxOrImpWait* **by** *fastforce*
qed

lemma *WaitInsertion*:
$\vdash g \longrightarrow f\ \mathcal{W}\ g$
by (*simp add*: *Prop05 UntilIntro wait-d-def*)

lemma *WaitFrameNext*:
$\vdash \Box\ f \longrightarrow (\bigcirc\ g \longrightarrow \bigcirc\ (f\ \mathcal{W}\ g))$
by (*simp add*: *NextImpNext Prop01 Prop05 Prop09 WaitInsertion*)

lemma *WaitFrameDiamond*:
$\vdash \Box\ f \longrightarrow (\Diamond\ g \longrightarrow \Diamond\ (f\ \mathcal{W}\ g))$
by (*simp add*: *DiamondImpDiamond Prop01 Prop05 Prop09 WaitInsertion*)

lemma *WaitFrameBox*:
$\vdash \Box\ f \longrightarrow (\Box\ g \longrightarrow \Box\ (f\ \mathcal{W}\ g))$
by (*meson BoxAndBoxImpBoxRule OrImpUntil Prop09 UntilImpWait lift-imp-trans*)

lemma *WaitInductiona*:
$\vdash \Box\ (f \longrightarrow (\bigcirc\ f \land g) \lor h) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ h)$
by (*simp add*: *UntilInduction-a wait-d-def*)

lemma *WaitInductionb*:

$\vdash \Box\ (f \longrightarrow \bigcirc\ f \vee g) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
by (*simp add*: *UntilInduction-b wait-d-def*)

lemma *WaitInductionc*:
$\vdash \Box(f \longrightarrow \bigcirc\ f) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
by (*meson WaitInductionc-help BoxImpWait BoxInduct Prop09 lift-imp-trans*)

lemma *WaitInductiond*:
$\vdash \Box\ (f \longrightarrow g \wedge \bigcirc\ f) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
by (*meson WaitInductiond-help BoxImpWait BoxInduct Prop09 lift-imp-trans*)

lemma *WaitAbsorptiona*:
$\vdash (f \vee f\ \mathcal{W}\ g) = (f \vee g)$
proof −
have *1*: $\vdash (f \vee f\ \mathcal{W}\ g) \longrightarrow (f \vee g)$
using *WaitImpOr* **by** *fastforce*
have *2*: $\vdash f \vee g \longrightarrow f \vee f\ \mathcal{W}\ g$
using *WaitInsertion* **by** *fastforce*
show *?thesis* **using** *1 2 int-iffI* **by** *blast*
qed

lemma *WaitAbsorptionb*:
$\vdash (f\ \mathcal{W}\ g \vee g) = f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *BoxEqvBoxBox UntilAbsorp-a UntilAbsorp-b WaitAbsorptiona WaitLeftDistOr WaitOrRule inteq-reflection wait-d-def*)

lemma *WaitAbsorptionc*:
$\vdash (f\ \mathcal{W}\ g \wedge g) = g$
using *WaitInsertion* **by** *fastforce*

lemma *WaitAbsorptiond*:
$\vdash (f\ \mathcal{W}\ g \wedge (f \vee g)) = f\ \mathcal{W}\ g$
by (*meson Prop10 Prop11 WaitImpOr*)

lemma *WaitAbsorptione*:
$\vdash (f\ \mathcal{W}\ g \vee (f \wedge g)) = f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *BoxEqvBoxBox UntilAbsorp-a UntilAbsorp-d WaitAbsorptiona WaitLeftDistOr WaitOrRule inteq-reflection wait-d-def*)

lemma *WaitLeftAbsorption*:
$\vdash f\ \mathcal{W}\ (f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *BoxEqvBoxBox UntilUntil WaitAbsorptionBox WaitAbsorptiona WaitLeftDistOr inteq-reflection wait-d-def*)

lemma *WaitRightAbsorption*:
$\vdash (f\ \mathcal{W}\ g)\ \mathcal{W}\ g = f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *LeftUntilAbsorp Prop10 WaitInsertion WaitNotDistUntil int-iffD1 int-iffI int-simps*(*32*) *inteq-reflection*)

lemma *WaitBox*:

$\vdash \square\ f = f\ \mathcal{W}\ \#False$
by (*metis* (*no-types*, *lifting*) *BoxGen DiamondNotEqvNotBox UntilAbsorpAndDiamond UntilAbsorp-c int-eq-true int-simps*(*2*) *int-simps*(*25*) *inteq-reflection wait-d-def*)

lemma *WaitDiamond*:
$\vdash \diamond\ f = (\neg(\ (\neg\ f)\ \mathcal{W}\ \#False))$
using *DiamondNotEqvNotBox WaitBox* **by** *fastforce*

lemma *WaitImp*:
$\vdash f\ \mathcal{W}\ g \longrightarrow \square\ f \lor \diamond\ g$
by (*metis Prop08 UntilImpDiamond WaitAbsorptionb WaitImpOr WaitRightAbsorption int-eq wait-d-def*)

lemma *WaitRightUntilAbsorption*:
$\vdash f\ \mathcal{W}\ (f\ \mathcal{U}\ g) = f\ \mathcal{W}\ g$
by (*metis UntilUntil WaitOrRule inteq-reflection wait-d-def*)

lemma *WaitLeftUntilAbsorption*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{W}\ g = f\ \mathcal{U}\ g$
by (*metis Prop11 RightUntilAbsorp UntilAbsorp-b UntilImpWait WaitImpOr inteq-reflection*)

lemma *UntilRightWaitAbsorption*:
$\vdash f\ \mathcal{U}\ (f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
using *UntilImpWait UntilIntro WaitLeftAbsorption* **by** *fastforce*

lemma *UntilLeftWaitAbsorption*:
$\vdash (f\ \mathcal{W}\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*metis UntilWait WaitRightAbsorption inteq-reflection*)

lemma *WaitDiamondAbsorption*:
$\vdash (\diamond\ g)\ \mathcal{W}\ g = \diamond\ g$
by (*metis DiamondEqvTrueUntil WaitLeftUntilAbsorption inteq-reflection*)

lemma *WaitAndBoxAbsorption*:
$\vdash (\square\ f \land f\ \mathcal{W}\ g) = \square\ f$
by (*meson BoxImpWait NotDiamondNotEqvBox Prop04 Prop10*)

lemma *WaitOrBoxAbsorption*:
$\vdash (\square\ f \lor f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
by (*metis UntilRightWaitAbsorption WaitLeftAbsorption inteq-reflection wait-d-def*)

lemma *WaitAndBoxImpBox*:
$\vdash f\ \mathcal{W}\ g \land \square\ (\neg\ g) \longrightarrow \square\ f$
by (*metis* (*no-types*, *hide-lams*) *Prop02 Prop05 Prop07 Prop08 UntilIdempotent UntilImpDiamond UntilIntro always-d-def int-simps*(*25*) *int-simps*(*4*) *inteq-reflection wait-d-def*)

lemma *BoxImpUntilOrBox*:
$\vdash \square\ f \longrightarrow f\ \mathcal{U}\ g \lor \square\ (\neg\ g)$
proof –
have *1*: $\vdash (\square\ f \longrightarrow f\ \mathcal{U}\ g \lor \square\ (\neg\ g)) =$
$((\square\ f \land \diamond\ g) \longrightarrow f\ \mathcal{U}\ g)$

by (*auto simp add*: *always-d-def*)
have *2*: $\vdash (\Box\ f \wedge \Diamond\ g) \ \longrightarrow f\ \mathcal{U}\ g$
using *UntilAndImp* **by** *blast*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *NotBoxAndWaitImpDiamond*:
$\vdash \neg(\ \Box\ f) \wedge f\ \mathcal{W}\ g \longrightarrow \Diamond\ g$
using *WaitImp* **by** *fastforce*

lemma *DiamondImpNotBoxOrUntil*:
$\vdash \Diamond\ g \longrightarrow \neg\ (\Box\ f) \vee f\ \mathcal{U}\ g$
proof −
have *1*: $\vdash \Diamond\ g \wedge \Box\ f \ \longrightarrow f\ \mathcal{U}\ g$
using *UntilAndImp* **by** *fastforce*
show *?thesis* **using** *1* **by** *auto*
qed

lemma *WaitLeftMono*:
$\vdash \Box\ (f \longrightarrow g) \longrightarrow (f\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ h)$
by (*meson BoxImpWait WaitRightDistImp lift-imp-trans*)

lemma *WaitRightMono*:
$\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{W}\ f \longrightarrow h\ \mathcal{W}\ g)$
proof −
have *1*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{U}\ f \longrightarrow h\ \mathcal{U}\ g)$
by (*simp add*: *UntilRightMono*)
have *2*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (\Box\ h \ \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
by *auto*
have *3*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{U}\ f \ \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
using *1* **by** *auto*
have *4*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (\Box\ h \vee h\ \mathcal{U}\ f \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *wait-d-def*)
qed

lemma *WaitStrengthen*:
$\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
proof −
have *1*: $\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g)$
by (*meson BoxAndBoxEqvBoxRule Prop01 Prop05 Prop11 WaitLeftMono lift-and-com lift-imp-trans*)
have *2*: $\vdash \ \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
by (*metis BoxImpBoxRule Prop01 Prop05 TrueW WaitRightMono int-simps*(*13*) *inteq-reflection lift-imp-trans*)
have *3*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g) \wedge (\ h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
using *1 2* **by** *fastforce*
have *4*: $\vdash (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g) \wedge (\ h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
by *auto*
from *3 4* **show** *?thesis* **by** *auto*

qed

lemma *WaitCatRule*:
$\vdash \Box(\ (f \longrightarrow g \ \mathcal{W} \ h) \wedge (h \longrightarrow g \ \mathcal{W} \ i)) \longrightarrow (f \longrightarrow g \ \mathcal{W} \ i)$
proof −
have *1*: $\vdash \Box \ (\ (f \longrightarrow g \ \mathcal{W} \ \ h) \wedge (h \longrightarrow g \ \mathcal{W} \ \ i)) \longrightarrow \Box \ (f \longrightarrow g \ \mathcal{W} \ h)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *2*: $\vdash \Box \ (\ (f \longrightarrow g \ \mathcal{W} \ h) \wedge (h \longrightarrow g \ \mathcal{W} \ i)) \longrightarrow \Box \ (h \longrightarrow g \ \mathcal{W} \ i)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *3*: $\vdash \Box \ (h \longrightarrow g \ \mathcal{W} \ i) \longrightarrow \Box \ (\ g \ \mathcal{W} \ h \longrightarrow g \ \mathcal{W} \ (g \ \mathcal{W} \ i))$
by (*metis BoxEqvBoxBox BoxImpBoxRule WaitRightMono inteq-reflection*)
have *4*: $\vdash \Box \ (\ g \ \mathcal{W} \ h \longrightarrow g \ \mathcal{U} \ (g \ \mathcal{W} \ i)) \longrightarrow \Box(\ g \ \mathcal{W} \ h \longrightarrow g \ \mathcal{W} \ i)$
by (*metis BoxBoxImpBox BoxEqvBoxBox UntilRightWaitAbsorption inteq-reflection*)
have *5*: $\vdash \ \Box \ (f \longrightarrow g \ \mathcal{W} \ h) \longrightarrow (f \longrightarrow g \ \mathcal{W} \ h)$
by (*simp add*: *BoxElim*)
have *6*: $\vdash \Box(\ g \ \mathcal{W} \ h \longrightarrow g \ \mathcal{W} \ i) \longrightarrow (g \ \mathcal{W} \ h \longrightarrow g \ \mathcal{W} \ i)$
by (*simp add*: *BoxElim*)
have *7*: $\vdash (f \longrightarrow g \ \mathcal{W} \ h) \wedge (g \ \mathcal{W} \ h \longrightarrow g \ \mathcal{W} \ i) \longrightarrow (f \longrightarrow g \ \mathcal{W} \ i)$
by *auto*
have *8*: $\vdash \Box \ (\ (f \longrightarrow g \ \mathcal{W} \ h) \wedge (h \longrightarrow g \ \mathcal{W} \ i)) \longrightarrow$
$(f \longrightarrow g \ \mathcal{W} \ h) \wedge (g \ \mathcal{W} \ h \longrightarrow g \ \mathcal{W} \ i)$
using *1 2 3 4 5 6*
by (*metis Prop12 WaitLeftAbsorption inteq-reflection lift-imp-trans*)
from *7 8* **show** *?thesis* **by** *auto*
qed

lemma *LeftUntilWaitImp*:
$\vdash (f \ \mathcal{U} \ g) \ \mathcal{W} \ h \longrightarrow (f \ \mathcal{W} \ g) \ \mathcal{W} \ h$
by (*meson BoxGen MP UntilImpWait WaitLeftMono*)

lemma *RightWaitUntilImp*:
$\vdash f \ \mathcal{W} \ (g \ \mathcal{U} \ h) \longrightarrow f \ \mathcal{W} \ (g \ \mathcal{W} \ h)$
by (*meson BoxGen MP UntilImpWait WaitRightMono*)

lemma *RightUntilUntilImp*:
$\vdash f \ \mathcal{U} \ (g \ \mathcal{U} \ h) \longrightarrow f \ \mathcal{U} \ (g \ \mathcal{W} \ h)$
by (*meson BoxGen MP UntilImpWait UntilRightMono*)

lemma *LeftUntilUntilImp*:
$\vdash (f \ \mathcal{U} \ g) \ \mathcal{U} \ h \longrightarrow (f \ \mathcal{W} \ g) \ \mathcal{U} \ h$
by (*simp add*: *UntilImpUntil UntilImpWait*)

lemma *LeftUntilOrStrengthen*:
$\vdash (f \ \mathcal{U} \ g) \ \mathcal{U} \ h \longrightarrow (f \vee g) \ \mathcal{U} \ h$
by (*simp add*: *UntilImpOr UntilImpUntil*)

lemma *LeftWaitOrStrengthen*:
$\vdash (f \ \mathcal{W} \ g) \ \mathcal{W} \ h \longrightarrow (f \vee g) \ \mathcal{W} \ h$
by (*meson BoxGen MP WaitImpOr WaitLeftMono*)

lemma *RightWaitOrStrengthen*:
$\vdash f \; \mathcal{W} \; (g \; \mathcal{W} \; h) \longrightarrow f \; \mathcal{W} \; (g \vee h)$
by (*meson BoxGen MP WaitImpOr WaitRightMono*)

lemma *BoxImpBoxOr*:
$\vdash \Box \; f \longrightarrow \Box(f \vee g)$
by (*metis BoxImpWait BoxIntro UntilBoxEqvBox UntilBoxImphelp2 WaitImpOr inteq-reflection lift-imp-trans*)

lemma *RightWaitOrOrder*:
$\vdash f \; \mathcal{W} \; (g \; \mathcal{W} \; h) \longrightarrow (f \vee g) \; \mathcal{W} \; h$
proof –
have *1*: $\vdash f \; \mathcal{W} \; (g \; \mathcal{W} \; h) = (\Box \; f \vee f \; \mathcal{U} \; (\Box \; g \vee g \; \mathcal{U} \; h))$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash (f \vee g) \; \mathcal{W} \; h = (\Box \; (f \vee g) \vee (f \vee g) \; \mathcal{U} \; h)$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash \Box \; f \longrightarrow (\Box \; (f \vee g) \vee (f \vee g) \; \mathcal{U} \; h)$
using *BoxImpBoxOr* **by** *fastforce*
have *4*: $\vdash f \; \mathcal{U} \; (\Box \; g \vee g \; \mathcal{U} \; h) = (f \; \mathcal{U} \; (\Box \; g) \vee f \; \mathcal{U} \; (g \; \mathcal{U} \; h))$
using *UntilOrDist* **by** *blast*
have *5*: $\vdash f \; \mathcal{U} \; (g \; \mathcal{U} \; h) \longrightarrow (\Box \; (f \vee g) \vee (f \vee g) \; \mathcal{U} \; h)$
by (*simp add*: *Prop05 UntilRightOr*)
have *6*: $\vdash f \; \mathcal{U} \; (\Box \; g) \longrightarrow (\Box \; (f \vee g) \vee (f \vee g) \; \mathcal{U} \; h)$
by (*metis BoxImpBoxRule BoxImpWait UntilBoxImp UntilImpOr lift-imp-trans wait-d-def*)
have *7*: $\vdash (f \; \mathcal{U} \; (\Box \; g) \vee f \; \mathcal{U} \; (g \; \mathcal{U} \; h)) \longrightarrow (\Box \; (f \vee g) \vee (f \vee g) \; \mathcal{U} \; h)$
using *5 6* **by** *fastforce*
show *?thesis*
using *1 2 3 4 7* **by** *fastforce*
qed

lemma *RightWaitAndOrder*:
$\vdash f \; \mathcal{W} \; (g \wedge h) \longrightarrow f \; \mathcal{W} \; (g \; \mathcal{W} \; h)$
by (*metis Prop03 WaitAbsorptione WaitLeftDistOr inteq-reflection*)

lemma *UntilOrder*:
$\vdash ((\neg \; f) \; \mathcal{U} \; g \vee (\neg \; g) \; \mathcal{U} \; f) = \Diamond(f \vee g)$
proof –
have *1*: $\vdash ((\neg \; f) \; \mathcal{U} \; g \vee (\neg \; g) \; \mathcal{U} \; f) \longrightarrow \Diamond(f \vee g)$
using *UntilAbsorpAndDiamond UntilOrDist* **by** *fastforce*
have *2*: $\vdash \Diamond(f \vee g) = \# True \; \mathcal{U} \; (f \vee g)$
by (*metis DiamondEqvTrueUntil*)
have *3*: $\vdash \# True \; \mathcal{U} \; (f \vee g) = (\neg \; (f \vee g) \;) \; \mathcal{U} \; (f \vee g)$
using *2 UntilRule* **by** *fastforce*
have *4*: $\vdash (\neg \; (f \vee g) \;) \; \mathcal{U} \; (f \vee g) = (\neg f \wedge \neg g) \; \mathcal{U} \; (f \vee g)$
by (*metis UntilAbsorp-c int-eq int-simps*(*14*) *int-simps*(*33*))
have *5*: $\vdash \Diamond(f \vee g) \longrightarrow (\neg \; g) \; \mathcal{U} \; f \vee (\neg \; f) \; \mathcal{U} \; g$
by (*metis 2 3 4 UntilAndRule UntilOrDist int-iffD1 inteq-reflection lift-and-com*)
have *6*: $\vdash (\; (\neg \; g) \; \mathcal{U} \; f \vee (\neg \; f) \; \mathcal{U} \; g) = (\; (\neg \; f) \; \mathcal{U} \; g \vee (\neg \; g) \; \mathcal{U} \; f)$

by *fastforce*
show *?thesis* **using** *1 5 6* **by** *fastforce*
qed

lemma *WaitOrder*:
$\vdash (\neg f)\ \mathcal{W}\ g \lor (\neg g)\ \mathcal{W}\ f$
proof −
have *1*: $\vdash (\neg f)\ \mathcal{W}\ g = (\Box (\neg f) \lor (\neg f)\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash (\neg g)\ \mathcal{W}\ f = (\Box (\neg g) \lor (\neg g)\ \mathcal{U}\ f)$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash ((\Box (\neg f) \lor (\neg f)\ \mathcal{U}\ g) \lor (\Box (\neg g) \lor (\neg g)\ \mathcal{U}\ f)) =$
$((\Box (\neg f) \lor \Box (\neg g)) \lor ((\neg f)\ \mathcal{U}\ g \lor (\neg g)\ \mathcal{U}\ f))$
by *auto*
have *4*: $\vdash ((\neg f)\ \mathcal{U}\ g \lor (\neg g)\ \mathcal{U}\ f) = \Diamond(f \lor g)$
using *UntilOrder* **by** *blast*
show *?thesis*
using *1 2 4 WaitOrder-help* **by** *fastforce*
qed

lemma *WaitImpOrder*:
$\vdash f\ \mathcal{W}\ g \land (\neg g)\ \mathcal{W}\ h \longrightarrow f\ \mathcal{W}\ h$
proof −
have *1*: $\vdash f\ \mathcal{W}\ g = (\Box f \lor f\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash f\ \mathcal{W}\ h = (\Box f \lor f\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash (\neg g)\ \mathcal{W}\ h = (\Box (\neg g) \lor (\neg g)\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *4*: $\vdash \Box f \land (\Box (\neg g) \lor (\neg g)\ \mathcal{U}\ h) \longrightarrow (\Box f \lor f\ \mathcal{U}\ h)$
by *auto*
have *5*: $\vdash (\Box f \lor f\ \mathcal{U}\ g) \land \Box (\neg g) \longrightarrow (\Box f \lor f\ \mathcal{U}\ h)$
using *1 WaitAndBoxImpBox* **by** *fastforce*
have *6*: $\vdash f\ \mathcal{U}\ g \land (\neg g)\ \mathcal{U}\ h \longrightarrow (\Box f \lor f\ \mathcal{U}\ h)$
by (*simp add*: *Prop05 UntilNotImp*)
have *7*: $\vdash \Box f \land (\neg g)\ \mathcal{U}\ h \longrightarrow (\Box f \lor f\ \mathcal{U}\ h)$
by *auto*
have *8*: $\vdash (\Box f \lor f\ \mathcal{U}\ g) \land (\neg g)\ \mathcal{U}\ h \longrightarrow (\Box f \lor f\ \mathcal{U}\ h)$
using *6 7* **by** *fastforce*
have *9*: $\vdash (\Box f \lor f\ \mathcal{U}\ g) \land (\Box (\neg g) \lor (\neg g)\ \mathcal{U}\ h) \longrightarrow (\Box f \lor f\ \mathcal{U}\ h)$
using *5 8* **by** *fastforce*
show *?thesis* **by** (*simp add*: *9 wait-d-def*)
qed

end

22 Pi operator

theory *Pi*
imports *IntervalFilter UntilSince*

begin

This theory introduces the Pi operator [10, 7]. The Pi operator is defined in terms of the filter operator introduced in IntervalFilter.thy. We prove the soundness of the rules and axiom system. The until operator from UntilSince.thy is used as there is a striking similarity of the expressiveness of the until and the Pi operator [7].

22.1 Definitions

definition *sfxfilt* :: *'a interval* ⇒ (*'a*:: *world*) *formula* ⇒ *'a interval interval*
where *sfxfilt xs f* = (*filter* (λ *ys*. *f ys*) (*suffixes xs*))

definition *pifilt* :: *'a interval* ⇒ (*'a*:: *world*) *formula* ⇒ *'a interval*
where *pifilt xs f* = (*map* (λ*s*. *nth s 0*) (*sfxfilt xs f*))

definition *pi-d* :: (*'a*:: *world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *pi-d F G* ≡ λ*s*. ((∃ *i* ≤ *intlen s*. (*suffix i s*) ⊨ *F*) ∧ ((*pifilt s F*) ⊨ *G*))

syntax
-pi-d :: [*lift*,*lift*] ⇒ *lift* ((- Π -) [*84*,*84*] *83*)

syntax (*ASCII*)
-pi-d :: [*lift*,*lift*] ⇒ *lift* ((- *PI* -) [*84*,*84*] *83*)

translations
-pi-d ⇌ *CONST pi-d*

definition *upi-d* :: (*'a*:: *world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *upi-d F G* ≡ *LIFT*(¬(*F* Π (¬ *G*)))

syntax
-upi-d :: [*lift*,*lift*] ⇒ *lift* ((- Π^u -) [*84*,*84*] *83*)

syntax (*ASCII*)
-upi-d :: [*lift*,*lift*] ⇒ *lift* ((- *UPI* -) [*84*,*84*] *83*)

translations
-upi-d ⇌ *CONST upi-d*

22.2 Semantic Lemmas

lemma *sfxfilter-help*:
(∃ *ys* ∈ *set* (*suffixes xs*) . *f ys*) = (∃ *i* ≤ *intlen xs*. *f* (*suffix i xs*))
using *set-suffixes-sfx* **by** *auto*

lemma *pifiltinit-help*:

```
 (∃ y ∈ set (xs). w ⟨y⟩) = (∃ i ≤ intlen xs. w ⟨nth xs i⟩)
by (metis interval-nth-and-set)

lemma sfxfilt-state:
  sfxfilt ⟨x⟩ f = ⟨⟨x⟩⟩
by (auto simp:sfxfilt-def)

lemma pifilt-state:
 pifilt ⟨x⟩ f = ⟨x⟩
by (auto simp: pifilt-def sfxfilt-state)

lemma sfxfilt-cons:
shows (sfxfilt (x⊙xs) f) =
  (if (∃i≤intlen xs. f (suffix i xs)) then
             (if f (x⊙xs) then
                 (x⊙xs)⊙ (sfxfilt (xs) f)
              else   (sfxfilt xs f) )
            else ⟨x⊙xs⟩)
proof −
 have 1: (sfxfilt (x⊙xs) f)  =
         filter (λys. f ys) (suffixes (x⊙xs))

    by (simp add: sfxfilt-def)
 have 2: suffixes (x⊙xs) = (x⊙xs) ⊙ suffixes xs
   by simp
 have 3: filter (λys. f ys) (suffixes (x⊙xs)) =
         (if (∃ ys ∈ set (suffixes xs). f ys) then
           (if f (x⊙xs) then (x⊙xs)⊙ (filter (λys. f ys) (suffixes (xs)))
                         else (filter (λys. f ys) (suffixes (xs))) )
          else ⟨x⊙xs⟩)

   by auto
 have 4: (if (∃ ys ∈ set (suffixes xs). f ys) then
           (if f (x⊙xs) then (x⊙xs)⊙ (filter (λys. f ys) (suffixes (xs)))
                         else (filter (λys. f ys) (suffixes (xs))) )
          else ⟨x⊙xs⟩) =
          (if (∃i≤intlen xs. f (suffix i xs)) then
              (if f (x⊙xs) then
                  (x⊙xs)⊙ (sfxfilt (xs) f)
               else   (sfxfilt xs f) )
             else ⟨x⊙xs⟩)

    by (auto simp add: sfxfilt-def set-suffixes-sfx)
  show ?thesis by (simp add: 1 4)
qed

lemma pifilt-cons:
shows (pifilt (x⊙xs) f) =
  (if (∃i≤intlen xs. f (suffix i xs)) then
             (if f (x⊙xs) then
```

(*x*)⊙ (*pifilt* (*xs*) *f*)
else (*pifilt xs f*))
else ⟨*x*⟩)

proof −
have *1*: (*pifilt* (*x*⊙*xs*) *f*) = *map* (λ*s*. (*nth s 0*)) (*sfxfilt* (*x*⊙*xs*) *f*)
by (*simp add*: *pifilt-def*)
have *2*: *map* (λ*s*. (*nth s 0*)) (*sfxfilt* (*x*⊙*xs*) *f*) =
(*if* (∃ *i*≤*intlen xs*. *f* (*suffix i xs*)) *then*
(*if f* (*x*⊙*xs*) *then*
map (λ*s*. (*nth s 0*))((*x*⊙*xs*)⊙ (*sfxfilt* (*xs*) *f*))
else *map* (λ*s*. (*nth s 0*)) (*sfxfilt xs f*))
else map (λ*s*. (*nth s 0*)) ⟨*x*⊙*xs*⟩)

using *1*
by (*metis sfxfilt-cons*)
have *3*: *map* (λ*s*. (*nth s 0*))((*x*⊙*xs*)⊙ (*sfxfilt* (*xs*) *f*)) =
(*x*)⊙ (*pifilt* (*xs*) *f*)

by (*simp add*: *pifilt-def*)
have *4*: *map* (λ*s*. (*nth s 0*)) ⟨*x*⊙*xs*⟩ = ⟨*x*⟩
by *simp*
have *5*: *map* (λ*s*. (*nth s 0*)) (*sfxfilt xs f*) = (*pifilt* (*xs*) *f*)
by (*simp add*: *pifilt-def*)
have *6*: (*if* (∃ *i*≤*intlen xs*. *f* (*suffix i xs*)) *then*
(*if f* (*x*⊙*xs*) *then*
map (λ*s*. (*nth s 0*))((*x*⊙*xs*)⊙ (*sfxfilt* (*xs*) *f*))
else *map* (λ*s*. (*nth s 0*)) (*sfxfilt xs f*))
else map (λ*s*. (*nth s 0*)) ⟨*x*⊙*xs*⟩) =
(*if* (∃ *i*≤*intlen xs*. *f* (*suffix i xs*)) *then*
(*if f* (*x*⊙*xs*) *then*
(*x*)⊙ (*pifilt* (*xs*) *f*)
else (*pifilt xs f*))
else ⟨*x*⟩)
using *5* **by** *auto*
show *?thesis* **by** (*simp add*: *1 2 6*)
qed

lemma *sfxfilt-nth-cons*:
nth (*sfxfilt* (*x*⊙*xs*) *f*) *j* =
(*if* (∃ *i*≤*intlen xs*. *f* (*suffix i xs*)) *then*
(*if f* (*x*⊙*xs*) *then*
(*if j* =*0 then* (*x*⊙*xs*) *else nth* (*sfxfilt* (*xs*) *f*) (*j*−*1*))
else *nth* (*sfxfilt* (*xs*) *f*) *j*)
else (*x*⊙*xs*))
by (*metis Interval.nth.simps*(*1*) *add.commute add.left-neutral add-diff-inverse-nat diff-0-eq-0 interval-nth-cons-a interval-nth-zero sfxfilt-cons*)

lemma *pifilt-nth-cons*:
nth (*pifilt* (*x* ⊙ *xs*) *f*) *i* =
(*if* (∃ *i*≤*intlen xs*. *f* (*suffix i xs*)) *then*

(*if f* (*x*⊙*xs*) *then*
(*if i* =*0 then* *x else nth* (*pifilt* (*xs*) *f*) (*i*−*1*))
else *nth* (*pifilt* (*xs*) *f*) *i*)
else x)
by (*metis Interval.nth.simps*(*1*) *One-nat-def Suc-pred interval-nth-Suc interval-nth-zero not-gr0 pifilt-cons*)

lemma *sfxfilt-nth*:
assumes (∃ *i* ≤ *intlen σ*. (*suffix i σ*) ⊨ *f*)
i≤*intlen* (*sfxfilt σ f*)
shows (*nth* (*sfxfilt σ f*) *i*) ⊨ *f*
using *assms in-set-suffixes osfx-suffix sfxfilter-nth*
by (*simp add*: *sfxfilt-def*, *blast*)

lemma *pifilt-exists*:
assumes (∃ *i*≤*intlen σ*. *f* (*suffix i σ*))
shows (∃ *i* ≤ *intlen*(*sfxfilt σ f*). (*nth* (*sfxfilt σ f*) *i*) ⊨ *f*)
using *assms*
using *sfxfilt-nth* **by** *blast*

lemma *sfxfilt-pifilt-intlen*:
assumes (∃ *i* ≤ *intlen σ*. (*suffix i σ*) ⊨ *f*)
shows *intlen* (*pifilt σ f*) = *intlen* (*map* (λ*s*. *nth s 0*) (*sfxfilt σ f*))
using *assms* **by** (*simp add*: *pifilt-def*)

lemma *sfxfilt-pifilt-nth*:
assumes (∃ *i* ≤ *intlen σ*. (*suffix i σ*) ⊨ *f*)
j≤ *intlen* (*pifilt σ f*)
shows *nth* (*pifilt σ f*) *j* = *nth* (*map* (λ*s*. *nth s 0*) (*sfxfilt σ f*)) *j*
using *assms*
by (*simp add*: *pifilt-def*)

lemma *sfxfilt-pifilt*:
assumes (∃ *i* ≤ *intlen σ*. (*suffix i σ*) ⊨ *f*)
shows (*map* (λ*s*. *nth s 0*) (*sfxfilt σ f*)) = *pifilt σ f*
using *assms* **by** (*simp add*: *pifilt-def*)

lemma *sfxfilt-intlen-bound*:
assumes (∃ *i*≤*intlen xs*. *f* (*suffix i xs*))
shows *intlen* (*sfxfilt xs f*) ≤ *intlen xs*
using *assms*
by (*simp add*: *sfxfilt-def*)
(*metis IntervalFilter.length-filter-le intlen-suffixes*)

lemma *pifilt-intlen-bound*:
assumes (∃ *i*≤*intlen σ*. *f* (*suffix i σ*))
shows *intlen* (*pifilt σ f*) ≤ *intlen σ*
using *assms* **by** (*simp add*: *pifilt-def sfxfilt-intlen-bound*)

lemma *sfxfilt-intlen-nth-bound*:
assumes (∃ i≤intlen σ. f (suffix i σ))
 j≤ intlen (sfxfilt σ f)
shows intlen (nth (sfxfilt σ f) j) ≤ intlen σ
using *assms*
by (*simp add*: *sfxfilt-def sfxfilter-help sfxfilter-nth-bound*)

lemma *sfxfilt-pifilt-nth-suffix*:
assumes (∃ i ≤ intlen σ. (suffix i σ) ⊨ f)
 j≤ intlen (pifilt σ f)
shows nth (sfxfilt σ f) j = suffix (intlen σ − (intlen(nth (sfxfilt σ f) j))) σ
proof −
 have *1*: nth (sfxfilt σ f) j = nth (filter f (suffixes σ)) j
  by (*simp add*: *sfxfilt-def*)
 have *2*: suffix (intlen σ − intlen (nth (filter f (suffixes σ)) j)) σ =
  suffix (intlen σ − (intlen(nth (sfxfilt σ f) j))) σ
  by (*simp add*: *sfxfilt-def*)
 have *3*: j ≤ intlen (filter f (suffixes σ))
  using *assms* **by** (*simp add*: *pifilt-def sfxfilt-def*)
 have *4*: (∃ i≤intlen σ. f (suffix i σ))
  using *assms* **by** *auto*
 have *5*: (∃ ys ∈ set (suffixes σ). f ys)
  using *4*
  using *in-set-suffixes osfx-suffix* **by** *blast*
 have *6*: nth (filter f (suffixes σ)) j =
  suffix (intlen σ − intlen (nth (filter f (suffixes σ)) j)) σ

  using *3 5 sfxfilter-nth-suffix*[*of σ f j*] **by** *blast*
 show *?thesis* **using** *1 6* **by** *auto*
qed

lemma *pifilt-nth*:
 assumes (∃ i≤intlen σ. f (suffix i σ))
  i ≤ intlen (pifilt σ f)
shows (∃ k ≤ intlen σ. nth (pifilt σ f) i = nth σ k)
using *assms sfxfilt-pifilt-nth-suffix*[*of σ f i*]
by (*simp add*: *pifilt-def*)
 (*metis diff-le-self interval-intfirst-suffix interval-intlen-gr-zero interval-nth-map*
 interval-suffix-zero)

lemma *sfxfilt-intlen-imp*:
 assumes (∃ i≤intlen σ. f (suffix i σ) ∧ g (suffix i σ))
 shows intlen (sfxfilt σ (LIFT(f ∧ g))) ≤ intlen (sfxfilt σ f)
proof −
 have *1*: intlen (sfxfilt σ (LIFT(f ∧ g))) =
  intlen (filter (λys. f ys ∧ g ys) (suffixes σ))
  by (*simp add*: *sfxfilt-def*)
 have *2*: intlen (filter f (suffixes σ)) = intlen (sfxfilt σ f)
  by (*simp add*: *sfxfilt-def*)
 have *3*: ∃ x ∈ set (suffixes σ). f x ∧ g x

using *assms* **by** *auto*
have *4*: *intlen* (*filter* (λ*x*. *f x* ∧ *g x*) (*suffixes* σ)) ≤ *intlen* (*filter f* (*suffixes* σ))
using *3 filter-intlen-imp*[*of suffixes* σ *f g*] **by** *auto*
show *?thesis* **using** *1 2 4* **by** *auto*
qed

lemma *pifilt-intlen-imp*:
assumes (∃ *i*≤*intlen* σ. *f* (*suffix i* σ) ∧ *g* (*suffix i* σ))
shows *intlen* (*pifilt* σ (*LIFT*(*f* ∧ *g*))) ≤ *intlen* (*pifilt* σ *f*)
using *assms*
by (*simp add*: *sfxfilt-intlen-imp pifilt-def*)

lemma *interval-set-sfxfilt* [*simp*]:
assumes (∃ *i* ≤ *intlen xs*. *f* (*suffix i xs*))
shows (*set* (*sfxfilt xs f*)) = {*ys*. *ys* ∈ *set* (*suffixes xs*) ∧ *f* (*ys*)}
using *assms*
proof −
have *1*: *set* (*sfxfilt xs f*) = *set* (*filter f* (*suffixes xs*))
by (*simp add*: *sfxfilt-def*)
have *2*: ∃ *x* ∈ *set* (*suffixes xs*). *f x*
using *assms* **using** *in-set-suffixes osfx-suffix* **by** *blast*
have *3*: *set* (*filter f* (*suffixes xs*)) =
{*ys*. *ys* ∈ *set* (*suffixes xs*) ∧ *f ys*}

using *2 set-filter*[*of suffixes xs f*] **by** *auto*
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *interval-subset-sfxfilt* [*simp*]:
assumes (∃ *i* ≤ *intlen xs*. *f* (*suffix i xs*))
shows (*set* (*sfxfilt xs f*)) ≤ (*set* (*sfxfilt xs* (*LIFT*(*f* ∨ *g*))))
proof −
have *1*: ∃ *x* ∈ *set* (*suffixes xs*). *f x*
using *assms* **using** *in-set-suffixes osfx-suffix* **by** *blast*
have *2*: *set* (*sfxfilt xs f*) = *set* (*filter f* (*suffixes xs*))
by (*simp add*: *sfxfilt-def*)
have *3*: *set* (*sfxfilt xs* (*LIFT*(*f* ∨ *g*))) = *set* (*filter* (λ*x*. *f x* ∨ *g x*) (*suffixes xs*))
by (*simp add*: *sfxfilt-def*)
have *4*: *set* (*filter f* (*suffixes xs*)) ≤ *set* (*filter* (λ*x*. *f x* ∨ *g x*) (*suffixes xs*))
using *1 subset-filter*[*of suffixes xs f g*] **by** *auto*
show *?thesis* **using** *2 3 4* **by** *blast*
qed

lemma *interval-set-pifilt* [*simp*]:
assumes (∃ *i* ≤ *intlen xs*. *f* (*suffix i xs*))
shows (*set* (*pifilt xs f*)) = {(*nth ys 0*)| *ys*. *ys* ∈ *set*(*suffixes xs*) ∧ *f* (*ys*)}
proof −
have *1*: (*set* (*pifilt xs f*)) = (*set* (*map* (λ*s*. *nth s 0*) (*sfxfilt xs f*)))

by (*simp add*: *pifilt-def*)
have *2*: (*set* (*map* (λ*s*. *nth s 0*) (*sfxfilt xs f*))) =
{(*nth ys 0*) | *ys*. *ys* $\in$ *set* (*sfxfilt xs f*)}
by (*induction xs*) *auto*
have *3*: {(*nth ys 0*) | *ys*. *ys* $\in$ *set* (*sfxfilt xs f*)} =
{(*nth ys 0*)| *ys*. *ys* $\in$ *set*(*suffixes xs*) $\wedge$ *f* (*ys*)}

using *assms* **by** *auto*
show *?thesis* **using** *1 2 3* **by** *auto*
qed

lemma *interval-nth-sfxfilt-in-set*:
x $\in$ *set* (*sfxfilt* σ (*LIFT*(*f* $\vee$ *g*))) =
($\exists$ *i* $\leq$ *intlen* (*sfxfilt* σ (*LIFT*(*f* $\vee$ *g*))). *x* =(*nth* (*sfxfilt* σ (*LIFT*(*f* $\vee$ *g*))) *i*))
by (*metis interval-nth-and-set*)

lemma *sfxfilt-nth-or*:
assumes ($\exists$ *i* $\leq$ *intlen* σ. (*suffix i* σ) $\models$ *f*)
shows ($\exists$ *i* $\leq$ *intlen* (*sfxfilt* σ (*LIFT*(*f* $\vee$ *g*))). (*nth* (*sfxfilt* σ (*LIFT*(*f* $\vee$ *g*))) *i*) $\models$ *f*)
proof −
have *1*: $\exists$ *x* $\in$ *set* (*suffixes* σ). *f x*
using *assms* **by** *auto*
have *2*: *sfxfilt* σ (*LIFT*(*f* $\vee$ *g*)) = *filter* (λ *x*. *f x* $\vee$ *g x*) (*suffixes* σ)
by (*simp add*: *sfxfilt-def*)
have *3*: $\exists$ *x* $\in$ *set*(*filter* (λ *x*. *f x* $\vee$ *g x*) (*suffixes* σ)). *f x*
using *1 filter-nth-or*[*of suffixes* σ *f g*] **by** *auto*
from *2 3* **show** *?thesis* **by** (*metis interval-nth-sfxfilt-in-set*)
qed

lemma *NotPiFalse*:
σ $\models$ $\neg$ ((#*False*) Π *f*)
by (*simp add*: *pi-d-def*)

lemma *pifilt-true*:
pifilt σ (*LIFT*(#*True*)) = σ
by (*simp add*: *pifilt-def sfxfilt-def IntervalFilter.filter-True*)

lemma *pifilt-init-state*:
(*pifilt* $\langle x\rangle$ (*LIFT*(*init w*))) = $\langle x\rangle$
by (*auto simp add*: *pifilt-def sfxfilt-def*)

lemma *pifilt-init-cons*:
(*pifilt* (*x*$\odot$*xs*) (*LIFT*(*init w*))) =
(*if* ($\exists$*i*$\leq$*intlen xs*. *w* $\langle$*nth xs i*$\rangle$) *then*
(*if* *w* $\langle x\rangle$ *then x* $\odot$ (*pifilt xs* (*LIFT*(*init w*)))
else (*pifilt xs* (*LIFT*(*init w*))))
else $\langle x\rangle$)
by (*simp add*: *pifilt-def sfxfilt-def init-defs*)
(*metis interval-nth-and-set interval-nth-map intlen-suffixes map-first-suffixes*)

lemma *PiStatesem*:
$(\sigma \models (init\ w)\ \Pi\ f) =$
$((\exists\, i \le intlen\ \sigma.\ w\ \langle nth\ \sigma\ i\rangle) \wedge f\ (pifilt\ \sigma\ (LIFT(init\ w))))$
by (*simp add*: *pi-d-def init-defs*)

22.3 Soundness of Axioms

22.3.1 PiK

lemma *PiKsem*:
$\sigma \models f1\ \Pi^u\ (\ f \longrightarrow g) \longrightarrow (\ \ f1\ \Pi^u\ f \longrightarrow f1\ \Pi^u\ g\ )$
by (*simp add*: *upi-d-def init-defs pi-d-def*) *auto*

lemma *PiK*:
$\vdash f1\ \Pi^u\ (f \longrightarrow g) \longrightarrow (\ \ f1\ \Pi^u\ f \longrightarrow f1\ \Pi^u\ g\ )$
using *PiKsem Valid-def* **by** *blast*

22.3.2 PiDc

lemma *PiDcsem*:
$\sigma \models f\ \Pi\ g \longrightarrow f\ \Pi^u\ g$
by (*simp add*: *upi-d-def init-defs pi-d-def*)

lemma *PiDc*:
$\vdash f\ \Pi\ g \longrightarrow f\ \Pi^u\ g$
using *PiDcsem Valid-def* **by** *blast*

22.3.3 PiN

lemma *PiN*:
assumes $\vdash g$
shows $\vdash f\ \Pi^u\ g$
using *assms* **by** (*simp add*: *Valid-def pi-d-def upi-d-def*)

22.3.4 PiTrueEqvDiamond

lemma *PiTrueEqvDiamond*:
$\vdash f\ \Pi\ \#True = \Diamond\ f$
by (*simp add*: *Valid-def pi-d-def sometimes-defs*)

22.3.5 PiOr

lemma *PiOr*:
$\vdash f\ \Pi\ (g1 \vee g2) = (f\ \Pi\ g1 \vee f\ \Pi\ g2)$
by (*simp add*: *Valid-def pi-d-def*) *blast*

22.3.6 UPiFalseEqvBoxNot:

lemma *UPiFalseEqvBoxNot*:
$\vdash f\ \Pi^u\ \#False = \Box\ (\neg f)$
by (*simp add*: *Valid-def upi-d-def pi-d-def always-defs*)

22.3.7 BoxEqvImpPiEqv

lemma *BoxEqvImpPiEqvsem*:
assumes $(\sigma \models \Box\ (f1 = f2))$
shows $(\sigma \models (f1\ \Pi\ g = f2\ \Pi\ g))$
proof −
show $(\sigma \models (f1\ \Pi\ g = f2\ \Pi\ g))$
proof −
have *1*: $\forall\, n \le intlen\ \sigma.\ f1\ (suffix\ n\ \sigma) = f2\ (suffix\ n\ \sigma)$
using *assms* **by** (*simp add*: *always-defs*)
have *2*: $(\sigma \models (f1\ \Pi\ g)) = ((\exists\, i \le intlen\ \sigma.\ f1\ (suffix\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f1))$
by (*simp add*: *pi-d-def*)
have *3*: $(\exists\, i \le intlen\ \sigma.\ f1\ (suffix\ i\ \sigma)) = (\exists\, i \le intlen\ \sigma.\ f2\ (suffix\ i\ \sigma))$
using *1* **by** *blast*
have *4*: $(sfxfilt\ \sigma\ f1) = (sfxfilt\ \sigma\ f2)$
using *1*
proof (*induct* σ)
case (*St x*)
then show *?case* **by** (*simp add*: *sfxfilt-state*)
next
case (*Cons x1a* σ)
then show *?case*
proof −
have *41*: $(sfxfilt\ (x1a \odot \sigma)\ f1) =$
$(if\ (\exists\, i \le intlen\ \sigma.\ (suffix\ i\ \sigma) \models f1)\ then$
$(if\ ((x1a \odot \sigma) \models f1)\ then\ (x1a \odot \sigma) \odot (sfxfilt\ \sigma\ f1)$
$else\ (sfxfilt\ \sigma\ f1))$
$else\ \langle x1a \odot \sigma \rangle)$
using *sfxfilt-cons* **by** *blast*
have *42*: $sfxfilt\ \sigma\ f1 = sfxfilt\ \sigma\ f2$
by (*metis Cons.hyps Cons.prems Suc-le-mono interval-suffix-suc intlen.simps*(*2*) *plus-1-eq-Suc*)
have *43*: $((x1a \odot \sigma) \models f1) = ((x1a \odot \sigma) \models f2)$
using *Cons.prems* **by** *auto*
have *44*: $(\exists\, i \le intlen\ \sigma.\ (suffix\ i\ \sigma) \models f1) =$
$(\exists\, i \le intlen\ \sigma.\ (suffix\ i\ \sigma) \models f2)$
by (*metis Suc-le-mono interval-suffix-suc intlen.simps*(*2*) *local.Cons*(*2*) *plus-1-eq-Suc*)
have *45*: $(if\ (\exists\, i \le intlen\ \sigma.\ (suffix\ i\ \sigma) \models f1)\ then$
$(if\ ((x1a \odot \sigma) \models f1)\ then\ (x1a \odot \sigma) \odot (sfxfilt\ \sigma\ f1)$
$else\ (sfxfilt\ \sigma\ f1))$
$else\ \langle x1a \odot \sigma \rangle) =$
$(if\ (\exists\, i \le intlen\ \sigma.\ (suffix\ i\ \sigma) \models f2)\ then$
$(if\ ((x1a \odot \sigma) \models f2)\ then\ (x1a \odot \sigma) \odot (sfxfilt\ \sigma\ f2)$
$else\ (sfxfilt\ \sigma\ f2))$
$else\ \langle x1a \odot \sigma \rangle)$
using *42 43 44* **by** *auto*
have *46*: $(if\ (\exists\, i \le intlen\ \sigma.\ (suffix\ i\ \sigma) \models f2)\ then$
$(if\ ((x1a \odot \sigma) \models f2)\ then\ (x1a \odot \sigma) \odot (sfxfilt\ \sigma\ f2)$
$else\ (sfxfilt\ \sigma\ f2))$
$else\ \langle x1a \odot \sigma \rangle) = (sfxfilt\ (x1a \odot \sigma)\ f2)$
by (*simp add*: *sfxfilt-cons*)

show *?thesis*
using *41 45 46* **by** *presburger*
qed
qed
have *47*: $(pifilt\ \sigma\ f1) = (pifilt\ \sigma\ f2)$
by (*simp add*: *4 pifilt-def*)
have *5*: $((\exists\, i{\le}intlen\ \sigma.\ f1\ (suffix\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f1)) =$
$((\exists\, i{\le}intlen\ \sigma.\ f2\ (suffix\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f2))$
by (*simp add*: *3 47*)
show *?thesis* **by** (*simp add*: *5 pi-d-def*)
qed
qed

lemma *BoxEqvImpPiEqv*:
$\vdash \Box\ (f1 = f2) \longrightarrow (f1\ \Pi\ g = f2\ \Pi\ g)$
using *BoxEqvImpPiEqvsem* **by** (*simp add*: *Valid-def*, *auto*)

22.3.8 PiDiamondImpDiamond

lemma *PiDiamondImpDiamondsem*:
$\sigma \models f\ \Pi\ (\Diamond\ (init\ w)) \longrightarrow \Diamond\ (init\ w)$
using *pifilt-nth* **by** (*simp add*: *Valid-def pi-d-def sometimes-defs init-defs*) *fastforce*

lemma *PiDiamondImp*:
$\vdash f\ \Pi\ (\Diamond\ (init\ w)) \longrightarrow \Diamond\ (init\ w)$
using *PiDiamondImpDiamondsem Valid-def* **by** *blast*

22.3.9 PiAssoc

lemma *PiAssocsem1*:
assumes $i \le intlen\ \sigma$
$f\ (suffix\ i\ \sigma)$
$ia \le intlen\ (pifilt\ \sigma\ f)$
$w\ \langle nth\ (pifilt\ \sigma\ f)\ ia\rangle$
shows $\exists\, i{\le}intlen\ \sigma.\ f\ (suffix\ i\ \sigma) \wedge w\ \langle nth\ (suffix\ i\ \sigma)\ 0\rangle$
proof −
have *1*: $(nth\ (pifilt\ \sigma\ f)\ ia) = (nth\ (map\ (\lambda s.\ nth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ ia)$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *sfxfilt-pifilt-nth* **by** *blast*
have *2*: $(nth\ (map\ (\lambda s.\ nth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ ia) =$
$(\lambda s.\ nth\ s\ 0)\ (nth\ (sfxfilt\ \sigma\ f)\ ia)$
using *interval-nth-map* **by** *auto*
have *3*: $f\ (nth\ (sfxfilt\ \sigma\ f)\ ia)$
using *sfxfilt-nth*
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *interval-intlen-map pifilt-def*)
have *4*: $nth\ (sfxfilt\ \sigma\ f)\ ia = suffix\ (intlen\ \sigma - intlen\ (nth\ (sfxfilt\ \sigma\ f)\ ia))\ \sigma$
using *sfxfilt-pifilt-nth-suffix* *assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *blast*
have *5*: $w\ \langle nth\ (suffix\ (intlen\ \sigma - intlen\ (nth\ (sfxfilt\ \sigma\ f)\ ia))\ \sigma)\ 0\rangle$
using *1 2 4 assms*(*4*) **by** *auto*
show *?thesis*
by (*metis 3 4 5 diff-le-self*)
qed

lemma *PiAssocsem2*:
assumes $i \leq intlen\ \sigma$
$f\ (suffix\ i\ \sigma)$
$w\ \langle Interval.nth\ \sigma\ i\rangle$
shows $\exists j{\leq}intlen\ (pifilt\ \sigma\ f).\ w\ \langle nth\ (pifilt\ \sigma\ f)\ j\rangle$
proof −
have *1*: $\exists j{\leq}intlen\ (sfxfilt\ \sigma\ f).\ f\ (nth\ (sfxfilt\ \sigma\ f)\ j)$
using *assms pifilt-exists* **by** *blast*
have *2*: $(LIFT\ (init\ w))\ (suffix\ i\ \sigma)$
by (*simp add*: *assms init-defs*)
have *3*: $\exists j{\leq}intlen\ (sfxfilt\ \sigma\ (LIFT(init\ w))).\ (LIFT(init\ w))\ (nth\ (sfxfilt\ \sigma\ (LIFT(init\ w)))\ j)$
using *pifilt-exists 2 assms* **by** *blast*
have *4*: $(LIFT\ (f \wedge init\ w))\ (suffix\ i\ \sigma)$
by (*simp add*: *assms init-defs*)
have *5*: $\exists j{\leq}intlen\ (sfxfilt\ \sigma\ (LIFT(f \wedge init\ w))).$
$(LIFT(f \wedge init\ w))\ (nth\ (sfxfilt\ \sigma\ (LIFT(f \wedge init\ w)))\ j)$
using *pifilt-exists 4 assms* **by** *blast*
have *6*: $\exists i{\leq}intlen\ \sigma.\ suffix\ i\ \sigma \models f \wedge init\ w$
using *4 assms* **by** *blast*
have *7*: $\exists j{\leq}intlen\ (sfxfilt\ \sigma\ (LIFT((f \wedge init\ w) \vee (f \wedge \neg\ (init\ w)))\ \ )).$
$(LIFT(f \wedge init\ w))\ (nth\ (sfxfilt\ \sigma\ (LIFT((f \wedge init\ w) \vee (f \wedge \neg(init\ w)))\ ))\ j)$
using *6 sfxfilt-nth-or*[*of* σ $LIFT(f \wedge init\ w)$ $LIFT(f \wedge \neg(init\ w))$]
by *auto*
have *8*: $\bigwedge\sigma\ .\ (\sigma \models ((f \wedge init\ w) \vee (f \wedge \neg\ (init\ w)))) = (\sigma \models f)$
by *auto*
have *9*: $(sfxfilt\ \sigma\ (LIFT((f \wedge init\ w) \vee (f \wedge \neg\ (init\ w)))\ \ )) =$
$(sfxfilt\ \sigma\ f)$
using *8* **by** (*simp add*: *sfxfilt-def*)
have *10*: $\exists j{\leq}intlen\ (sfxfilt\ \sigma\ f).$
$(LIFT(f \wedge init\ w))\ (nth\ (sfxfilt\ \sigma\ f)\ j)$
using *7 9* **by** *auto*
have *11*: $intlen\ (sfxfilt\ \sigma\ f) = intlen\ (pifilt\ \sigma\ f)$
by (*simp add*: *pifilt-def*)
have *12*: $\exists j{\leq}intlen\ (pifilt\ \sigma\ f).$
$(LIFT(init\ w))\ (nth\ (sfxfilt\ \sigma\ f)\ j)$
using *10 11* **by** *auto*
from *12 11* **show** *?thesis*
by (*simp add*: *init-defs*)
(*metis interval-nth-map pifilt-def*)
qed

lemma *PiAssocsema*:
$((\exists i{\leq}intlen\ \sigma.\ f\ (suffix\ i\ \sigma)) \wedge$
$(\exists i{\leq}intlen\ (pifilt\ \sigma\ f).\ w\ \langle nth\ (suffix\ i\ (pifilt\ \sigma\ f))\ 0\rangle)) =$
$(\exists i{\leq}intlen\ \sigma.\ f\ (suffix\ i\ \sigma) \wedge w\ \langle nth\ (suffix\ i\ \sigma)\ 0\rangle)$
using *PiAssocsem1 PiAssocsem2* **by** *fastforce*

lemma *PiAssocsemb*:
$((\exists i{\leq}intlen\ \sigma.\ f\ (suffix\ i\ \sigma)) \wedge$

$(\exists\, i{\le}intlen\ (pifilt\ \sigma\ f).\ (LIFT(init\ w))\ (suffix\ i\ (pifilt\ \sigma\ f))\ )) =$
$(\exists\, i{\le}intlen\ \sigma.\ (LIFT(f \wedge init\ w))\ (suffix\ i\ \sigma))$
using *PiAssocsem1 PiAssocsem2*
by (*simp add*: *init-defs*) *fastforce*

lemma *pifilt-state-help*:
$(\exists\ x \in set\ (suffixes\ xs)\ .\ (LIFT(init\ w))\ x) = (\exists\ x \in set\ xs.\ w\ (\langle x\rangle))$
proof (*auto simp add*: *init-defs*)
show $\bigwedge x.\ osfx\ x\ xs \Longrightarrow w\ \langle Interval.nth\ x\ 0\rangle \Longrightarrow \exists x{\in}interval.set\ xs.\ w\ \langle x\rangle$
using *in-set-suffixes interval-sfx-1* **by** *blast*
show $\bigwedge x.\ x \in interval.set\ xs \Longrightarrow w\ \langle x\rangle \Longrightarrow \exists x{\in}interval.set\ (suffixes\ xs).\ w\ \langle Interval.nth\ x\ 0\rangle$
by (*metis in-set-suffixes interval-intfirst-suffix interval-nth-and-set interval-nth-zero-intfirst osfx-suffix*)
qed

lemma *pifilt-init*:
assumes $(\exists\, i{\le}intlen\ xs.\ (LIFT(init\ w))\ (suffix\ i\ xs))$
shows $(pifilt\ xs\ (LIFT(init\ w))) = filter\ (\lambda y.\ w\ (\langle y\rangle))\ xs$
using *assms*
proof (*induct xs*)
case (*St x*)
then show *?case*
by (*simp add*: *pifilt-init-state*)
next
case (*Cons x1a xs*)
then show *?case*
proof −
have *1*: $pifilt\ (x1a{\odot}xs)\ (LIFT(init\ w)) =$
$map\ (\lambda xs\ .\ (nth\ xs\ 0))\ (sfxfilt\ (x1a{\odot}xs)\ (LIFT(init\ w)))$

using *sfxfilt-pifilt* **by** (*simp add*: *pifilt-def*)
have *2*: $sfxfilt\ (x1a{\odot}xs)\ (LIFT(init\ w)) =$
$(filter\ (\lambda\ ys.\ (LIFT(init\ w))\ ys)\ (suffixes\ (x1a\ {\odot}xs)))$

using *sfxfilt-def* **by** *blast*
have *3*: $suffixes\ (x1a\ {\odot}xs) = (x1a{\odot}xs){\odot}\ (suffixes\ xs)$
by *simp*
have *4*: $(filter\ (\lambda\ ys.\ (LIFT(init\ w))\ ys)\ (suffixes\ (x1a\ {\odot}xs))) =$
$(if\ (\exists\ x \in set\ (suffixes\ xs)\ .\ (LIFT(init\ w))\ x)\ then$
$(if\ (LIFT(init\ w))\ (x1a{\odot}xs)\ then\ (x1a{\odot}xs){\odot}\ (filter\ (LIFT(init\ w))\ (suffixes\ xs))$
$else\ (filter\ (LIFT(init\ w))\ (suffixes\ xs)))$
$else\ \langle x1a \odot xs\rangle)$

by *simp*
have *5*: $map\ (\lambda xs\ .\ (nth\ xs\ 0))\ (filter\ (\lambda\ ys.\ (LIFT(init\ w))\ ys)\ (suffixes\ (x1a\ {\odot}xs))) =$
$(if\ (\exists\ x \in set\ (suffixes\ xs)\ .\ (LIFT(init\ w))\ x)\ then$
$(if\ (LIFT(init\ w))\ (x1a{\odot}xs)$
$then\ (x1a) \odot map\ (\lambda xs\ .\ (nth\ xs\ 0))\ (filter\ (LIFT(init\ w))\ (suffixes\ xs))$
$else\ map\ (\lambda xs\ .\ (nth\ xs\ 0))\ (filter\ (LIFT(init\ w))\ (suffixes\ xs)))$

else ⟨*x1a*⟩)

by *auto*
have *6*: *filter* (λ*y*. *w* ⟨*y*⟩) (*x1a* ⊙ *xs*) =
(*if* (∃ *x* ∈ *set xs*. *w* (⟨*x*⟩)) *then*
(*if w* (⟨*x1a*⟩) *then x1a*⊙ (*filter* (λ*y*. *w* ⟨*y*⟩) *xs*) *else* (*filter* (λ*y*. *w* ⟨*y*⟩) *xs*))
else ⟨*x1a*⟩)

by *simp*
have *61*: (∃ *x* ∈ *set* (*suffixes xs*) . (*LIFT*(*init w*)) *x*) =
(∃ *x* ∈ *set xs*. *w* (⟨*x*⟩))

by (*auto simp*: *init-defs interval-sfx-1*)
(*metis init-defs interval-prefix-zero-intfirst pifilt-state-help*)
have *62*: (*LIFT*(*init w*)) (*x1a*⊙*xs*) = *w* (⟨*x1a*⟩)
by (*simp add*: *init-defs*)
have *63*: (∃ *x* ∈ *set xs*. *w* (⟨*x*⟩)) ⟶
map (λ*xs* . (*nth xs 0*)) (*filter* (*LIFT*(*init w*)) (*suffixes xs*)) =
(*filter* (λ*y*. *w* ⟨*y*⟩) *xs*)
by (*auto simp add*: *interval-nth-and-set*)
(*metis Cons.hyps interval.distinct*(*1*) *pifilt-cons pifilt-def pifilt-init-cons sfxfilt-def*)
have *7*: (*if* (∃ *x* ∈ *set* (*suffixes xs*) . (*LIFT*(*init w*)) *x*) *then*
(*if* (*LIFT*(*init w*)) (*x1a*⊙*xs*)
then (*x1a*) ⊙ *map* (λ*xs* . (*nth xs 0*)) (*filter* (*LIFT*(*init w*)) (*suffixes xs*))
else map (λ*xs* . (*nth xs 0*)) (*filter* (*LIFT*(*init w*)) (*suffixes xs*)))
else ⟨*x1a*⟩) =
(*if* (∃ *x* ∈ *set xs*. *w* (⟨*x*⟩)) *then*
(*if w* (⟨*x1a*⟩) *then x1a*⊙ (*filter* (λ*y*. *w* ⟨*y*⟩) *xs*) *else* (*filter* (λ*y*. *w* ⟨*y*⟩) *xs*))
else ⟨*x1a*⟩)

by (*simp add*: *61 62 63*)
show *?thesis* **using** *1 2 5 7* **by** *auto*
qed
qed

lemma *pifilt-pifilt* :
assumes (∃ *i*≤*intlen xs*. *f* (*suffix i xs*))
(∃ *i*≤*intlen* (*pifilt xs f*). *w* ⟨*nth* (*suffix i* (*pifilt xs f*)) *0*⟩)
shows (*pifilt* (*pifilt xs f*) (*LIFT*(*init w*))) = *pifilt xs* (*LIFT*(*f* ∧ *init w*))
proof −
have *1*: ∃ *i*≤*intlen* (*pifilt xs f*). (*LIFT*(*init w*)) (*suffix i* (*pifilt xs f*))
using *assms* **by** (*simp add*: *init-defs*)
have *2*: (*pifilt* (*pifilt xs f*) (*LIFT*(*init w*))) =
filter (λ*y*. *w* (⟨*y*⟩)) (*pifilt xs f*)

using *1 pifilt-init*[*of* (*pifilt xs f*) *w*] **by** *auto*
have *3*: (*pifilt xs f*) =
map (λ*s*. *nth s 0*) (*sfxfilt xs f*)
by (*simp add*: *assms sfxfilt-pifilt*)
have *4*: (*sfxfilt xs f*) = *filter* (λ *ys*. *f ys*) (*suffixes xs*)

using *sfxfilt-def* **by** *blast*
have *5*: (*pifilt xs f*) = *map* (λ*s*. *nth s 0*) (*filter* (λ *ys*. *f ys*) (*suffixes xs*))
by (*simp add*: *3 4*)
have *6*: *filter* (λ*y*. *w* (⟨*y*⟩)) (*pifilt xs f*) =
filter (λ*y*. *w* (⟨*y*⟩)) (*map* (λ*s*. *nth s 0*) (*filter* (λ *ys*. *f ys*) (*suffixes xs*)))

using *5* **by** *simp*
have *7*: *filter* (λ*y*. *w* (⟨*y*⟩)) (*map* (λ*s*. *nth s 0*) (*filter* (λ *ys*. *f ys*) (*suffixes xs*))) =
map (λ*s*. *nth s 0*) (*filter* ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) (*filter* (λ *ys*. *f ys*) (*suffixes xs*)))
using *assms* **by** (*metis 3 4 filter-map in-set-suffixes interval-sfx-1 osfx-suffix*)
have *8*: ∃*x*∈*interval*.*set* (*filter* (λ *ys*. *f ys*) (*suffixes xs*)). ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *x*
using *assms 3 4*
by *simp-all*
(*metis interval-intlen-map interval-nth-map interval-nth-zero-intfirst nth-set*)
have *9*: ∃*x*∈*interval*.*set* (*suffixes xs*). (λ *ys*. *f ys*) *x*
using *assms in-set-suffixes osfx-suffix* **by** *blast*
have *10*: ∃*x*∈*interval*.*set* (*suffixes xs*). ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *x* ∧ (λ *ys*. *f ys*) *x*
using *8 9* **by** *auto*
have *11*: *filter* ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) (*filter* (λ *ys*. *f ys*) (*suffixes xs*)) =
filter (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) (*suffixes xs*)

using *filter-filter*[*of* (λ *ys*. *f ys*) (*suffixes xs*) ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*))]
8 10 **by** *blast*
have *12*: ∃ *i*≤*intlen xs*. (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) (*suffix i xs*)
by (*metis PiAssocsem1 assms comp-apply interval-intfirst-suffix interval-suffix-zero le0*)
have *13*: (*filter* (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) (*suffixes xs*))
= (*sfxfilt xs* (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*))

by (*simp add*: *sfxfilt-def*)
have *14*: ∃ *i*≤*intlen xs*. (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) (*suffix i xs*)
using *12* **by** *blast*
have *15*: *map* (λ*s*. *nth s 0*)
((*sfxfilt xs* (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*))) =
pifilt xs (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*)

using *14* **by** (*simp add*: *sfxfilt-pifilt*)
have *16*: ⋀*xs* . (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) *xs* =
(*LIFT*(*f* ∧ *init w*)) *xs*

by (*auto simp add*: *init-defs*)
have *17*: *pifilt xs* (λ *zs*. ((λ*y*. *w* (⟨*y*⟩))∘(λ*s*. *nth s 0*)) *zs* ∧ (λ *ys*. *f ys*) *zs*) =
pifilt xs (*LIFT*(*f* ∧ *init w*))

using *16* **by** *presburger*
show *?thesis*
using *11 13 15 17 2 3 4 7* **by** *auto*
qed

lemma *PiAssocsem*:

$\sigma \models f \ \Pi \ ((init\ w) \ \Pi \ \ g) = \ \ (f \wedge (init\ w)) \ \Pi \ g$
proof (*auto simp add*: *pi-d-def init-defs*)
show $\bigwedge i\ ia.$
$g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w))) \Longrightarrow$
$i \leq intlen\ \sigma \Longrightarrow$
$f\ (suffix\ i\ \sigma) \Longrightarrow$
$ia \leq intlen\ (pifilt\ \sigma\ f) \Longrightarrow$
$w\ \langle Interval.nth\ (pifilt\ \sigma\ f)\ ia\rangle \Longrightarrow$
$\exists\, i{\leq}intlen\ \sigma.\ f\ (suffix\ i\ \sigma) \wedge w\ \langle Interval.nth\ \sigma\ i\rangle$
using *PiAssocsem1* **by** *fastforce*
show $\bigwedge i\ ia.$
$g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w))) \Longrightarrow$
$i \leq intlen\ \sigma \Longrightarrow$
$f\ (suffix\ i\ \sigma) \Longrightarrow$
$ia \leq intlen\ (pifilt\ \sigma\ f) \Longrightarrow$
$w\ \langle Interval.nth\ (pifilt\ \sigma\ f)\ ia\rangle \Longrightarrow$
$g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$
by (*metis interval-intfirst-suffix interval-nth-zero-intfirst pifilt-pifilt*)
show $\bigwedge i.\ g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w))) \Longrightarrow$
$i \leq intlen\ \sigma \Longrightarrow f\ (suffix\ i\ \sigma) \Longrightarrow$
$w\ \langle Interval.nth\ \sigma\ i\rangle \Longrightarrow$
$\exists\, i{\leq}intlen\ (pifilt\ \sigma\ f).\ w\ \langle Interval.nth\ (pifilt\ \sigma\ f)\ i\rangle$
by (*metis PiAssocsem2*)
show $\bigwedge i.\ g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w))) \Longrightarrow$
$i \leq intlen\ \sigma \Longrightarrow$
$f\ (suffix\ i\ \sigma) \Longrightarrow$
$w\ \langle Interval.nth\ \sigma\ i\rangle \Longrightarrow$
$g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w)))$
by (*metis PiAssocsem2 interval-intfirst-suffix interval-nth-zero-intfirst pifilt-pifilt*)
qed

lemma *PiAssoc*:
$\vdash f \ \Pi \ ((init\ w) \ \Pi \ \ g) = \ \ (f \wedge (init\ w)) \ \Pi \ g$
using *PiAssocsem Valid-def* **by** *blast*

22.3.10 PiNotEqvDiamondAndNotPi

lemma *PiNotEqvDiamondAndNotPisem*:
$\sigma \models f \ \Pi \ (\neg\ g) = (\Diamond f \wedge \neg(f \ \Pi \ g))$
by (*simp add*: *pi-d-def sometimes-defs*) *blast*

lemma *PiNotEqvDiamondAndNotPi*:
$\vdash f \ \Pi \ (\neg\ g) = (\Diamond f \wedge \neg(f \ \Pi \ g))$
using *PiNotEqvDiamondAndNotPisem Valid-def* **by** *blast*

22.3.11 PiChopDist

lemma *set-fuse*:
assumes $intlast\ xs = intfirst\ ys$
shows $set\ (fuse\ xs\ ys) = set\ xs \cup set\ ys$

using *assms*
proof (*induction xs arbitrary*: *ys*)
case (*St x*)
then show *?case*
by (*metis fuse-St interval-fuse-rightneutral interval-intapp-assoc interval-set-intapp*
  opfx-code(*1*) *opfx-def sup.idem*)
next
case (*Cons x1a xs*)
then show *?case* **by** *simp*
qed

lemma *filter-chop*:
assumes *intlast xs* = *intfirst ys* ∧ *P* (*intlast xs*)
  (∃ *x* ∈ *set* (*fuse xs ys*). *P x*)
  (∃ *x* ∈ *set xs*. *P x*)
  (∃ *x* ∈ *set ys*. *P x*)
shows *filter P* (*fuse xs ys*) = *fuse* (*filter P xs*) (*filter P ys*)
using *assms*
proof (*induction xs arbitrary*: *ys*)
case (*St x*)
then show *?case*
by *simp*
next
case (*Cons x1a xs*)
then show *?case*
  proof (*cases* (∃ *x* ∈ *set xs*. *P x*))
  case *True*
  then show *?thesis*
    using *Cons.IH Cons.prems*(*1*) *Cons.prems*(*4*) *set-fuse* **by** *fastforce*
  next
  case *False*
  then show *?thesis*
   using *Cons.prems*(*1*) *nth-set order-refl* **by** *force*
  qed
qed

lemma *filter-chop1*:
assumes *n*≤ *intlen xs* ∧ *P* (*intlast* (*prefix n xs*))
  (∃ *x* ∈ *set xs*. *P x*)
  (∃ *x* ∈ *set* (*prefix n xs*). *P x*)
  (∃ *x* ∈ *set* (*suffix n xs*). *P x*)
shows *filter P xs* = *fuse* (*filter P* (*prefix n xs*)) (*filter P* (*suffix n xs*))
proof −
 have *1*: (∃ *x* ∈ *set xs*. *P x*) =
   (∃ *x* ∈ *set* (*fuse* (*prefix n xs*) (*suffix n xs*)). *P x*)

  by (*simp add*: *assms*(*1*) *interval-fuse-prefix-suffix*)
 have *2*: *intlast* (*prefix n xs*) = *intfirst* (*suffix n xs*)
  using *Interval.interval-intlast-intfirst* **by** *blast*
 have *3*: *xs* = *fuse* (*prefix n xs*) (*suffix n xs*)

by (*simp add*: *assms*(*1*) *interval-fuse-prefix-suffix*)
have *4*: *filter P* (*fuse* (*prefix n xs*) (*suffix n xs*)) =
fuse (*filter P* (*prefix n xs*)) (*filter P* (*suffix n xs*))
using *assms 2 3 filter-chop*[*of* (*prefix n xs*) (*suffix n xs*) *P*]
by *auto*
show *?thesis* **using** *3 4* **by** *auto*
qed

lemma *filter-chop1-prefix*:
assumes *n*≤ *intlen xs*
P (*intlast* (*prefix n xs*))
(∃ *x* ∈ *set xs*. *P x*)
(∃ *x* ∈ *set* (*prefix n xs*). *P x*)
(∃ *x* ∈ *set* (*suffix n xs*). *P x*)
shows *prefix* (*intlen* (*filter P* (*prefix n xs*))) (*filter P xs*) =
(*filter P* (*prefix n xs*))
proof −
have *2*: *filter P xs* = *fuse* (*filter P* (*prefix n xs*)) (*filter P* (*suffix n xs*))
using *assms filter-chop1* **by** *blast*
have *3*: *intlast* (*filter P* (*prefix n xs*)) = *intfirst* (*filter P* (*suffix n xs*))
using *assms* **by** (*metis Interval*.*interval-intlast-intfirst filter-intfirst filter-intlast*)
have *4*: *prefix* (*intlen* (*filter P* (*prefix n xs*)))
(*fuse* (*filter P* (*prefix n xs*)) (*filter P* (*suffix n xs*))) =
(*filter P* (*prefix n xs*))

using *interval-prefix-fuse* **using** *3* **by** *blast*
show *?thesis* **by** (*simp add*: *2 4*)
qed

lemma *filter-chop1-suffix*:
assumes *n*≤ *intlen xs*
P (*intlast* (*prefix n xs*))
(∃ *x* ∈ *set xs*. *P x*)
(∃ *x* ∈ *set* (*prefix n xs*). *P x*)
(∃ *x* ∈ *set* (*suffix n xs*). *P x*)
shows *suffix* (*intlen* (*filter P* (*prefix n xs*))) (*filter P xs*) =
(*filter P* (*suffix n xs*))
proof −
have *1*: *filter P xs* = *fuse* (*filter P* (*prefix n xs*)) (*filter P* (*suffix n xs*))
using *assms filter-chop1* **by** *blast*
have *3*: *intlast* (*filter P* (*prefix n xs*)) = *intfirst* (*filter P* (*suffix n xs*))
using *assms* **by** (*metis Interval*.*interval-intlast-intfirst filter-intfirst filter-intlast*)
have *4*: *suffix* (*intlen* (*filter P* (*prefix n xs*)))
(*fuse* (*filter P* (*prefix n xs*)) (*filter P* (*suffix n xs*))) =
(*filter P* (*suffix n xs*))

using *interval-suffix-fuse* **using** *3* **by** *blast*
show *?thesis* **by** (*simp add*: *1 4*)
qed

lemma *PiChopDistsema*:
assumes $(\sigma \models (init\ w)\ \Pi\ (g;h))$
shows $(\sigma \models ((init\ w)\ \Pi\ g);((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
proof −
have *1*: $(\sigma \models (init\ w)\ \Pi\ (g;h))$
using *assms* **by** *auto*
have *2*: $((\exists\, i \leq intlen\ \sigma.\ (LIFT(init\ w))\ (suffix\ i\ \sigma)) \wedge$
$((pifilt\ \sigma\ (LIFT(init\ w))) \models g;h)$
)
using *1* **by** (*simp add*: *pi-d-def*)
have *3*: $(\exists\, i \leq intlen\ \sigma.\ (LIFT(init\ w))\ (suffix\ i\ \sigma))$
using *2* **by** *auto*
have *4*: $((pifilt\ \sigma\ (LIFT(init\ w))) \models g;h)$
using *2* **by** *auto*
have *5*: $filter\ (\lambda y.\ w\ (\langle y\rangle))\ \sigma \models g;h$
using *pifilt-init*
using *2* **by** *fastforce*
have *6*: $\exists\, n \leq intlen\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma).$
$g\ (prefix\ n\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma)) \wedge$
$h\ (suffix\ n\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma))$
using *5* **by** (*simp add*: *chop-defs*)
obtain *n* **where** *7*: $n \leq intlen\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma) \wedge$
$g\ (prefix\ n\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma)) \wedge$
$h\ (suffix\ n\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma))$
using *6* **by** *auto*
have *8*: $\exists\, i \leq intlen\ \sigma.\ w\ \langle nth\ \sigma\ i\rangle$
using *3* **by** (*auto simp add*: *init-defs*)
have *9*: $\exists\ x \in set\ \sigma.\ w\ \langle x\rangle$
using *8 nth-set* **by** *blast*
have *10*: $(prefix\ n\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma)) =$
$(filter\ (\lambda y.\ w\ \langle y\rangle)\ (prefix\ ((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ )\ \sigma)\ )$
by (*simp add*: *7 9 filter-nfilter-prefix-1*)
have *11*: $(suffix\ n\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma)) =$
$(filter\ (\lambda y.\ w\ \langle y\rangle)\ (suffix\ ((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ )\ \sigma)\ )$
by (*simp add*: *7 9 filter-nfilter-suffix-1*)
have *12*: $g\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ (prefix\ ((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ )\ \sigma)\ )$
using *10 7* **by** *auto*
have *13*: $h\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ (suffix\ ((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ )\ \sigma)\ )$
using *11 7* **by** *auto*
have *14*: $((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ ) \leq intlen\ \sigma$
by (*metis 7 9 add-cancel-right-left nfilter-intlen nfilter-upper-bound*)
have *15*: $w\ \langle nth\ (suffix\ ((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ )\ \sigma)\ 0\rangle$
by (*metis* (*no-types*, *lifting*) *14 7 9 filter-nth-aa interval-intfirst-suffix*
interval-suffix-zero nfilter-filter nfilter-intlen nfilter-nth-n-zero zero-le)
have *16*: $(\exists\, i \leq intlen\ (prefix\ \ ((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ )\ \sigma).$
$w\ \langle nth\ (suffix\ i\ (prefix\ \ ((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ )\ \sigma))\ 0\rangle)$
using *14 15* **by** *auto*
have *17*: $(\exists\, i \leq intlen\ (suffix\ ((nth\ (nfilter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma\ 0)\ n)\ )\ \sigma).$

w ⟨*nth* (*suffix* (*i* + ((*nth* (*nfilter* (λ*y*. *w* ⟨*y*⟩) σ *0*) *n*))) σ) *0*⟩)
using *15* **by** *auto*
have *18*: (∃ *n*≤*intlen* σ.
(∃ *i*≤*intlen* (*prefix n* σ). *w* ⟨*nth* (*suffix i* (*prefix n* σ)) *0*⟩) ∧
g (*filter* (λ*y*. *w* (⟨*y*⟩)) (*prefix n* σ)) ∧
w ⟨*nth* (*suffix n* σ) *0*⟩ ∧
(∃ *i*≤*intlen* (*suffix n* σ). *w* ⟨*nth* (*suffix* (*i* + *n*) σ) *0*⟩) ∧
h (*filter* (λ*y*. *w* (⟨*y*⟩)) (*suffix n* σ)))
using *12 13 14 15 16 17* **by** *blast*
have *19*: (∃ *n*≤*intlen* σ.
(∃ *i*≤*intlen* (*prefix n* σ). *w* ⟨*nth* (*suffix i* (*prefix n* σ)) *0*⟩) ∧
g (*pifilt* (*prefix n* σ) (*LIFT*(*init w*))) ∧
w ⟨*nth* (*suffix n* σ) *0*⟩ ∧
(∃ *i*≤*intlen* (*suffix n* σ). *w* ⟨*nth* (*suffix* (*i* + *n*) σ) *0*⟩) ∧
h (*pifilt* (*suffix n* σ) (*LIFT*(*init w*))))
by (*metis 18 init-defs interval-prefix-zero-intfirst interval-suffix-suffix pifilt-init*)
have *20*: ((∃ *n*≤*intlen* σ.
(∃ *i*≤*intlen* (*prefix n* σ). (*LIFT*(*init w*)) (*suffix i* (*prefix n* σ))) ∧
g (*pifilt* (*prefix n* σ) (*LIFT*(*init w*))) ∧
(*LIFT*(*init w*)) (*suffix n* σ) ∧ (∃ *i*≤*intlen* (*suffix n* σ). (*LIFT*(*init w*)) (*suffix* (*i* + *n*) σ))
∧ *h* (*pifilt* (*suffix n* σ) (*LIFT*(*init w*)))))
by (*metis 19 init-defs interval-prefix-zero-intfirst*)
have *21*: (σ |= ((*init w*) Π *g*);((*init w*) ∧ ((*init w*) Π *h*)))
using *20* **by** (*simp add*: *chop-defs pi-d-def*)
show *?thesis*
using *21* **by** *auto*
qed

lemma *PiChopDistsemb*:
assumes (σ |= ((*init w*) Π *g*);((*init w*) ∧ ((*init w*) Π *h*)))
shows (σ |= (*init w*) Π (*g*;*h*))
proof −
have *1*: (σ |= ((*init w*) Π *g*);((*init w*) ∧ ((*init w*) Π *h*)))
using *assms* **by** *auto*
have *2*: ∃ *n* ≤ *intlen* σ.
((*prefix n* σ) |= ((*init w*) Π *g*)) ∧
((*suffix n* σ) |= ((*init w*) ∧ ((*init w*) Π *h*)))

using *assms chop-defs* **by** *blast*
obtain *n* **where** *3*: *n* ≤ *intlen* σ ∧ ((*prefix n* σ) |= ((*init w*) Π *g*)) ∧
((*suffix n* σ) |= ((*init w*) ∧ ((*init w*) Π *h*)))
using *2* **by** *auto*
have *4*: ((∃ *i*≤*intlen* (*prefix n* σ). (*LIFT*(*init w*)) (*suffix i* (*prefix n* σ))) ∧
((*pifilt* (*prefix n* σ) (*LIFT*(*init w*))) |= *g*)
)
by (*meson 3 pi-d-def*)
have *5*: (∃ *i*≤*intlen* (*prefix n* σ). (*LIFT*(*init w*)) (*suffix i* (*prefix n* σ)))
using *4* **by** *auto*
have *6*: *g* (*pifilt* (*prefix n* σ) (*LIFT*(*init w*)))

using *4* **by** *auto*
have *7*: *g* (*filter* (λ*y*. *w* (⟨*y*⟩)) (*prefix n σ*))
using *5 6 pifilt-init* **by** (*metis*)
have *8*: ((∃ *i*≤*intlen* (*suffix n σ*). (*LIFT*(*init w*)) (*suffix i* (*suffix n σ*))) ∧
((*pifilt* (*suffix n σ*) (*LIFT*(*init w*))) ⊨ *h*)
)
by (*metis 3 intensional-rews*(*3*) *pi-d-def*)
have *9*: (∃ *i*≤*intlen* (*suffix n σ*). (*LIFT*(*init w*)) (*suffix i* (*suffix n σ*)))
using *8* **by** *auto*
have *10*: *h* (*pifilt* (*suffix n σ*) (*LIFT*(*init w*)))
using *8* **by** *auto*
have *11*: *h* (*filter* (λ*y*. *w* (⟨*y*⟩)) (*suffix n σ*))
using *10 9 pifilt-init* **by** *metis*
have *12*: (λ*y*. *w* (⟨*y*⟩)) (*intlast* (*prefix n σ*))
by (*metis 3 init-defs intensional-rews*(*3*) *interval-intfirst-suffix interval-intlast-prefix interval-nth-zero-intfirst interval-prefix-zero-intfirst*)
have *13*: ∃ *x* ∈ *set σ*. (λ*y*. *w* (⟨*y*⟩)) *x*
using *12 3 nth-set* **using** *interval-intlast-prefix* **by** *fastforce*
have *14*: ∃ *x* ∈ *set* (*prefix n σ*) . (λ*y*. *w* (⟨*y*⟩)) *x*
using *12 nth-set* **by** (*metis interval-nth-intlen-intlast order-refl*)
have *15*: ∃ *x* ∈ *set* (*suffix n σ*) . (λ*y*. *w* (⟨*y*⟩)) *x*
by (*metis 12 3 interval-intfirst-suffix interval-intlast-prefix interval-intlen-gr-zero interval-nth-zero-intfirst nth-set*)
have *16*: (*filter* (λ*y*. *w* (⟨*y*⟩)) (*prefix n σ*)) =
prefix (*intlen* (*filter* (λ*y*. *w* (⟨*y*⟩)) (*prefix n σ*))) (*filter* (λ*y*. *w* (⟨*y*⟩)) *σ*)

using *12 13 14 15 3*
filter-chop1-prefix[*of n σ* (λ*y*. *w* ⟨*y*⟩)] **by** *auto*
have *17*: (*filter* (λ*y*. *w* (⟨*y*⟩)) (*suffix n σ*)) =
suffix (*intlen* (*filter* (λ*y*. *w* (⟨*y*⟩)) (*prefix n σ*))) (*filter* (λ*y*. *w* (⟨*y*⟩)) *σ*)

using *12 13 14 15 3*
filter-chop1-suffix[*of n σ* (λ*y*. *w* ⟨*y*⟩)] **by** *auto*
have *18*: ∃ *n*≤*intlen* (*filter* (λ*y*. *w* ⟨*y*⟩) *σ*).
g (*prefix n* (*filter* (λ*y*. *w* ⟨*y*⟩) *σ*)) ∧
h (*suffix n* (*filter* (λ*y*. *w* ⟨*y*⟩) *σ*))
by (*metis 11 16 17 7 interval-pref-intlen-bound*)
have *19*: *filter* (λ*y*. *w* (⟨*y*⟩)) *σ* ⊨ *g*;*h*
by (*simp add*: *18 chop-defs*)
have *20*: ((*pifilt σ* (*LIFT*(*init w*))) ⊨ *g*;*h*)
by (*metis* (*mono-tags*, *lifting*) *18 3 Interval.interval-intlast-intfirst intensional-rews*(*3*) *interval-chop-fuse interval-fuse-prefix-suffix pifilt-init*)
have *21*: (∃ *i*≤*intlen σ*. (*LIFT*(*init w*)) (*suffix i σ*))
using *3* **by** *auto*
show *?thesis*
by (*simp add*: *20 21 pi-d-def*)
qed

lemma *PiChopDistsem*:
σ ⊨ (*init w*) Π (*g*;*h*) = ((*init w*) Π *g*);((*init w*) ∧ ((*init w*) Π *h*))

using *PiChopDistsema PiChopDistsemb unl-lift2* **by** *blast*

lemma *PiChopDist*:
$\vdash$ (*init w*) Π (*g*;*h*) = ((*init w*) Π *g*);((*init w*) $\wedge$ ((*init w*) Π *h*))
using *PiChopDistsem Valid-def* **by** *blast*

22.3.12 PiProp

lemma *Pistate*:
(σ $\models$ (*init w*) Π *f*) =
(($\exists$ *x* $\in$ *set* σ. *w* $\langle x\rangle$) $\wedge$ ((*filter* (λy. *w* $\langle y\rangle$) σ) $\models$ *f*))
proof −
have *1*: (σ $\models$ (*init w*) Π *f*) =
(($\exists$ *i*$\leq$ *intlen* σ. *w* $\langle$*nth* σ *i*$\rangle$) $\wedge$ ((*pifilt* σ (*LIFT*(*init w*))) $\models$ *f*))
by (*auto simp add*: *pi-d-def init-defs*)
have *2*: ($\exists$ *i*$\leq$*intlen* σ. (*LIFT*(*init w*)) (*suffix i* σ)) =
($\exists$ *i*$\leq$ *intlen* σ. *w* $\langle$*nth* σ *i*$\rangle$)
by (*auto simp add*: *init-defs*)
have *3*: ($\exists$ *i*$\leq$ *intlen* σ. *w* $\langle$*nth* σ *i*$\rangle$) $\longrightarrow$
(*pifilt* σ (*LIFT*(*init w*))) = (*filter* (λy. *w* $\langle y\rangle$) σ)
using *pifilt-init* **using** *2* **by** *blast*
have *4*: ($\exists$ *i*$\leq$ *intlen* σ. *w* $\langle$*nth* σ *i*$\rangle$) =($\exists$ *x* $\in$ *set* σ. *w* $\langle x\rangle$)
using *interval-nth-and-set* **by** *force*
show *?thesis*
using *1 3 4* **by** *auto*
qed

lemma *PiPropsem1a*:
(σ $\models$ *f* Π \$*p*) =
(($\exists$ *i*$\leq$*intlen* σ. *f* (*suffix i* σ)) $\wedge$ *p* (*nth* σ (*nth* (*nfilter f* (*suffixes* σ) *0*) *0*)))

using *interval-nth-map*[*of* (λs. *nth s 0*) (*filter f* (*suffixes* σ)) *0*]
using *nfilter-filter*[*of suffixes* σ *f 0 0*]
by (*simp add*: *pi-d-def current-val-d-def pifilt-def sfxfilt-def*)
(*metis in-set-suffixes interval-nth-map map-first-suffixes osfx-suffix*)

lemma *PiPropsem2a*:
(σ $\models$ ($\neg$ *f*) $\mathcal{U}$ (*f* $\wedge$ \$*p*)) =
($\exists$ *k*$\leq$*intlen* σ. *f* (*suffix k* σ) $\wedge$ *p* (*nth* σ *k*) $\wedge$ ($\forall$ *j*<*k*. $\neg$ *f* (*suffix j* σ)))
by (*simp add*: *until-d-def current-val-d-def*)

lemma *PiPropsem3a*:
assumes (σ $\models$ *f* Π \$*p*)
shows (σ $\models$ ($\neg$ *f*) $\mathcal{U}$ (*f* $\wedge$ \$*p*))
proof −
have *1*: (($\exists$ *i*$\leq$*intlen* σ. *f* (*suffix i* σ)) $\wedge$ *p* (*nth* σ (*nth* (*nfilter f* (*suffixes* σ) *0*) *0*)))
using *assms PiPropsem1a* **by** *auto*
have *2*: $\exists$ *x* $\in$ *set*(*suffixes* σ). *f x*
using *1 in-set-suffixes osfx-suffix* **by** *blast*

have *3*: ∀ *x* ∈ *set*(*nfilter f* (*suffixes σ*) *0*). *f* (*nth* (*suffixes σ*) *x*)
  by (*simp add*: *2*)
have *4*: *f* (*suffix* (*nth* (*nfilter f* (*suffixes σ*) *0*) *0*) *σ*)
  by (*metis 2 3 add.left-neutral interval-intlen-gr-zero nfilter-upper-bound nth-set nth-suffixes*)
have *5*: (∀ *j*< (*nth* (*nfilter f* (*suffixes σ*) *0*) *0*). ¬ *f* (*suffix j σ*))
  using *nfilter-not-before*[*of suffixes σ f*] *2*
  proof −
  have *f1*: ∀ *n*.¬ *n* < *Interval.nth* (*nfilter f* (*suffixes σ*) *0*) *0* ∨ ¬ *f* (*Interval.nth* (*suffixes σ*) *n*)
  using ‹⋀*i*. ⟦∃ *x*∈*set* (*suffixes σ*). *f x*; *i* < *nth* (*nfilter f* (*suffixes σ*) *0*) *0*⟧ ⟹ ¬ *f* (*nth* (*suffixes σ*) *i*)› ‹∃ *x*∈*set* (*suffixes σ*). *f x*› **by** *blast*
  obtain *ii* :: *'a interval* **where**
  f2: *ii* ∈ *set* (*suffixes σ*) ∧ *f ii*
  using ‹∃ *x*∈*set* (*suffixes σ*). *f x*› **by** *blast*
  obtain *nn* :: *'a interval interval* ⇒ *'a interval* ⇒ *nat* **where**
  ∀ *x0 x1*. (∃ *v2*≤*intlen x0*. *nth x0 v2* = *x1*) = (*nn x0 x1* ≤ *intlen x0* ∧ *nth x0* (*nn x0 x1*) = *x1*)
  by *moura*
  then have *nn* (*suffixes σ*) *ii* ≤ *intlen* (*suffixes σ*) ∧ *nth* (*suffixes σ*) (*nn* (*suffixes σ*) *ii*) = *ii*
  using *f2* **by** (*meson interval-nth-and-set*)
  then have *nth* (*nfilter f* (*suffixes σ*) *0*) *0* ≤ *intlen* (*suffixes σ*)
  using *f2 f1* **by** (*metis* (*no-types*) *dual-order.trans le-less-linear*)
  then show *?thesis*
  using *f1* **by** (*metis* (*no-types*) *dual-order.trans less-or-eq-imp-le nth-suffixes*)
  qed
have *6*: (∃ *k*≤*intlen σ*. *f* (*suffix k σ*) ∧ *p* (*nth σ k*) ∧ (∀ *j*<*k*. ¬ *f* (*suffix j σ*)))
  by (*metis 1 4 5 interval-suf-first neqE*)
show *?thesis* **using** *6 PiPropsem2a* **by** *metis*
qed

lemma *PiPropsem3b*:
assumes (*σ* ⊨ (¬ *f*) $\mathcal{U}$ (*f* ∧ $*p*))
shows (*σ* ⊨ *f* Π $*p*)
proof −
have *1*: (∃ *k*≤*intlen σ*. *f* (*suffix k σ*) ∧ *p* (*nth σ k*) ∧ (∀ *j*<*k*. ¬ *f* (*suffix j σ*)))
  using *assms PiPropsem2a* **by** *auto*
obtain *k* **where** *2*: *k*≤*intlen σ* ∧ *f* (*suffix k σ*) ∧ *p* (*nth σ k*) ∧ (∀ *j*<*k*. ¬ *f* (*suffix j σ*))
  using *1* **by** *auto*
have *3*: (∃ *i*≤*intlen σ*. *f* (*suffix i σ*))
  using *2* **by** *blast*
have *31*: ∃ *x* ∈ *set*(*suffixes σ*). *f x*
  using *2* **by** *auto*
have *32*: *f* (*suffix* (*nth* (*nfilter f* (*suffixes σ*) *0*) *0*) *σ*)
  using *nfilter-holds*[*of suffixes σ f 0*] *nfilter-not-before*[*of suffixes σ f*]
  by (*metis 31 diff-zero interval-intlen-gr-zero nfilter-upper-bound nth-set nth-suffixes ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *4*: *p* (*nth σ* (*nth* (*nfilter f* (*suffixes σ*) *0*) *0*))
  by (*metis 1 31 32 intlen-suffixes linorder-neqE-nat nfilter-not-before nth-suffixes*)
show *?thesis* **using** *4 3* **by** (*simp add*: *PiPropsem1a*)
qed

lemma *PiPropsema*:
$\sigma \models f \; \Pi \; \$p = (\neg\, f) \; \mathcal{U} \; (f \wedge \$p)$
using *PiPropsem3a PiPropsem3b unl-lift2* **by** *blast*

lemma *PiProp*:
$\vdash f \; \Pi \; \$p = (\neg\, f) \; \mathcal{U} \; (f \wedge \$p)$
using *PiPropsema Valid-def* **by** *blast*

22.3.13 PiNext

lemma *PiNextsem1*:
$(\sigma \models f \; \Pi \; (\bigcirc\, g)) =$
$((\exists\, i \le intlen\ \sigma.\ f\ (suffix\ i\ \sigma)) \wedge$
$0 < intlen\ (filter\ f\ (suffixes\ \sigma)) \wedge$
$g\ (suffix\ (Suc\ 0)\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ \sigma)))))$
by (*simp add*: *pi-d-def next-defs pifilt-def sfxfilt-def*)

lemma *PiNextsem2*:
$(\sigma \models (\neg\, f) \; \mathcal{U} \; (f \wedge \bigcirc(f \; \Pi \; g))) =$
$(\exists\, k \le intlen\ \sigma.$
$f\ (suffix\ k\ \sigma) \wedge$
$k < intlen\ \sigma \wedge$
$(\exists\, i \le intlen\ \sigma - Suc\ k.\ f\ (suffix\ (Suc\ (i + k))\ \sigma)) \wedge$
$g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ (Suc\ k)\ \sigma)))) \wedge (\forall\, j < k.\ \neg\, f\ (suffix\ j\ \sigma)))$

by (*simp add*: *until-d-def next-defs pi-d-def pifilt-def sfxfilt-def*)

lemma *PiNextsem3*:
assumes $(\sigma \models f \; \Pi \; (\bigcirc\, g))$
shows $(\sigma \models (\neg\, f) \; \mathcal{U} \; (f \wedge \bigcirc(f \; \Pi \; g)))$
proof −
have *1*: $((\exists\, i \le intlen\ \sigma.\ f\ (suffix\ i\ \sigma)) \wedge$
$0 < intlen\ (filter\ f\ (suffixes\ \sigma)) \wedge$
$g\ (suffix\ (Suc\ 0)\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ \sigma)))))$
using *assms PiNextsem1* **by** *auto*
have *2*: $\exists\ x \in set(suffixes\ \sigma).\ f\,x$
using *1 in-set-suffixes osfx-suffix* **by** *blast*
have *3*: $\forall\ x \in set(nfilter\ f\ (suffixes\ \sigma)\ 0).\ f\ (nth\ (suffixes\ \sigma)\ x)$
by (*simp add*: *2*)
have *4*: $f\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)\ \ \sigma)$
by (*metis 2 add.left-neutral interval-intlen-gr-zero nfilter-filter nfilter-nth-n-zero nfilter-upper-bound nth-suffixes sfxfilter-nth*)
have *41*: $(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) \in set(nfilter\ f\ (suffixes\ \sigma)\ 0)$
by (*metis 1 2 One-nat-def Suc-leI nfilter-intlen nth-set*)
have *42*: $(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) \le intlen\ (suffixes\ \sigma)$
by (*metis 1 2 One-nat-def Suc-leI add-cancel-right-left nfilter-intlen nfilter-upper-bound*)
have *5*: $f\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1)\ \ \sigma)$
using *3 41 42 nth-suffixes* **by** *fastforce*

have *6*: $(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0) < (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1)$
by (*simp add*: *1 2 idx-nfilter-mono nfilter-intlen*)
have *7*: $Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0) \le intlen\ \sigma$
by (*metis 1 3 IntervalFilter.length-filter-le Suc-diff-Suc diff-is-0-eq′ filter-nfilter-suffix interval-intlen-gr-zero interval-suffix-length intlen-suffixes not-less-eq-eq nth-set*)
have *8*: $0 < intlen\ (nfilter\ f\ (suffixes\ \sigma)\ 0)$
by (*simp add*: *1 2 nfilter-intlen*)
have *9*: $(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) \le intlen\ \sigma$
using *nfilter-upper-bound*[*of suffixes* σ *f 1 0*]
by (*simp add*: *2 8 Suc-leI*)
have *10*: $(Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)) \le (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1)$
using *6 Suc-leI* **by** *blast*
have *11*: $(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) =$
$(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) - (Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)) +$
$(Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0))$

using *10* **by** *auto*
have *12*: $(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) - (Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)) \le$
$intlen\ \sigma - Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)$
using *9 diff-le-mono* **by** *blast*
have *13*: $\exists\ i \le intlen\ \sigma - Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0).$
$(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) = (Suc\ (i + (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)))$
using *11 12* **by** *auto*
have *14*: $(\exists\, i {\le} intlen\ \sigma - Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0).$
$f\ (suffix\ (Suc\ (i + (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)))\ \sigma))$
using *13 5* **by** *auto*
have *15*: $(suffix\ (Suc\ 0)\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ \sigma)))) =$
$(map\ (\lambda s.\ nth\ s\ 0)$
$(filter\ f\ (suffixes\ (suffix\ (Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0))\ \sigma))))$
using *2 8* **by** (*simp add*: *1 Suc-leI filter-suffixes-map*)
have *16*: $g\ (map\ (\lambda s\ .nth\ s\ 0)$
$(filter\ f\ (suffixes\ (suffix\ (Suc\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0))\ \sigma))))$
using *1 15* **by** *auto*
have *17*: $\forall\ j < (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0).\ \neg\ f\ (suffix\ j\ \sigma)$
by (*metis 7 Suc-leD Suc-leI in-set-suffixes intlen-suffixes le-trans nfilter-not-before nth-suffixes osfx-suffix*)
have *18*: $(\exists\, k {\le} intlen\ \sigma.$
$f\ (suffix\ k\ \sigma)\ \wedge$
$k < intlen\ \sigma\ \wedge$
$(\exists\, i {\le} intlen\ \sigma - Suc\ k.\ f\ (suffix\ (Suc\ (i + k))\ \sigma))\ \wedge$
$g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ (Suc\ k)\ \sigma))))\ \wedge\ (\forall\, j{<}k.\ \neg\ f\ (suffix\ j\ \sigma)))$
using *14 16 17 4 7 Suc-leD Suc-le-lessD* **by** *blast*
show *?thesis* **using** *18* **by** (*simp add*: *PiNextsem2*)
qed

lemma *PiNextsem4*:
assumes $(\sigma \models (\neg\ f)\ \mathcal{U}\ (f \wedge \bigcirc(f\ \Pi\ g)))$
shows $(\sigma \models f\ \Pi\ (\bigcirc\ g))$

proof −
have *1*: $(\exists\, k \le intlen\ \sigma.$
$f\ (suffix\ k\ \sigma) \wedge$
$k < intlen\ \sigma \wedge$
$(\exists\, i \le intlen\ \sigma - Suc\ k.\ f\ (suffix\ (Suc\ (i + k))\ \sigma)) \wedge$
$g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ (Suc\ k)\ \sigma)))) \wedge (\forall\, j<k.\ \neg\ f\ (suffix\ j\ \sigma)))$
using *assms* **by** (*simp add*: *PiNextsem2*)
obtain *k* **where** *2*: $k \le intlen\ \sigma \wedge f\ (suffix\ k\ \sigma) \wedge k < intlen\ \sigma \wedge (\exists\, i \le intlen\ \sigma - Suc\ k.$
$f\ (suffix\ (Suc\ (i + k))\ \sigma)) \wedge$
$g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ (Suc\ k)\ \sigma)))) \wedge (\forall\, j<k.\ \neg\ f\ (suffix\ j\ \sigma))$
using *1* **by** *auto*
have *3*: $\exists\ x \in set\ (suffixes\ \sigma).\ f\, x$
using *1 in-set-suffixes osfx-suffix* **by** *blast*
have *4*: $\forall\ x \in set(nfilter\ f\ (suffixes\ \sigma)\ 0).\ f\ (nth\ (suffixes\ \sigma)\ x)$
by (*simp add*: *3*)
have *5*: $f\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)\ \sigma)$
by (*metis 3 4 add.left-neutral interval-intlen-gr-zero nfilter-upper-bound nth-set nth-suffixes*)
have *6*: $0 < intlen\ (filter\ f\ (suffixes\ \sigma))$
using *filter-length-zero-conv-a*[*of suffixes* σ *f*]
by (*metis 2 3 Nat.le-diff-conv2 Suc-leI Suc-n-not-le-n add-Suc-right intlen-suffixes le-add2 neq0-conv nth-suffixes*)
have *61*: $(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) \in set(nfilter\ f\ (suffixes\ \sigma)\ 0)$
by (*metis 3 6 One-nat-def Suc-leI nfilter-intlen nth-set*)
have *62*: $(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1) \le intlen\ (suffixes\ \sigma)$
by (*metis 3 6 One-nat-def Suc-leI add-cancel-right-left nfilter-intlen nfilter-upper-bound*)
have *7*: $f\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 1)\ \sigma)$
using *4 61 62 nth-suffixes* **by** *fastforce*
have *8*: $(\exists\, i \le intlen\ \sigma.\ f\ (suffix\ i\ \sigma))$
using *1* **by** *blast*
have *9*: $g\ (suffix\ (Suc\ 0)\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ \sigma))))$
by (*metis 2 3 5 6 Suc-leI intlen-suffixes linorder-neqE-nat nfilter-not-before nth-suffixes filter-suffixes-map*)
have *10*: $((\exists\, i \le intlen\ \sigma.\ f\ (suffix\ i\ \sigma)) \wedge$
$0 < intlen\ (filter\ f\ (suffixes\ \sigma)) \wedge$
$g\ (suffix\ (Suc\ 0)\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ \sigma)))))$
using *6 8 9* **by** *blast*
show *?thesis* **using** *10*
by (*simp add*: *PiNextsem1*)
qed

lemma *PiNextsem*:
$(\sigma \models f\ \Pi\ (\bigcirc\ g) = (\neg\ f)\ \mathcal{U}\ (f \wedge \bigcirc(f\ \Pi\ g)))$
using *PiNextsem3 PiNextsem4*
using *unl-lift2* **by** *blast*

lemma *PiNext*:
$\vdash f\ \Pi\ (\bigcirc\ g) = (\neg\ f)\ \mathcal{U}\ (f \wedge \bigcirc(f\ \Pi\ g))$

using *PiNextsem Valid-def* **by** *blast*

22.3.14 PiUntil

lemma *PiUntilDistsem1*:
  $(\sigma \models f\ \Pi\ (g\ \mathcal{U}\ h)) =$
   $((\exists\, i \le intlen\ \sigma.\ f\ (suffix\ i\ \sigma)) \wedge$
    $(\exists\, k \le intlen\ (filter\ f\ (suffixes\ \sigma)).$
       $h\ (suffix\ k\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ \sigma)))) \wedge$
       $(\forall\, j<k.\ g\ (suffix\ j\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ \sigma)))))))$
by (*simp add*: *pi-d-def pifilt-def sfxfilt-def until-d-def*)

lemma *PiUntilDistsem2*:
  $(\sigma \models (\ f\ \Pi\ g\ )\ \mathcal{U}\ (\ f\ \Pi\ h\ )\ ) =$
   $(\exists\, k \le intlen\ \sigma.$
      $(\exists\, i \le intlen\ \sigma - k.\ f\ (suffix\ (i + k)\ \sigma)) \wedge$
      $h\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma)))) \wedge$
      $(\forall\, j<k.\ (\exists\, i \le intlen\ \sigma - j.\ f\ (suffix\ (i + j)\ \sigma)) \wedge$
          $g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ j\ \sigma))))))$
by (*simp add*: *until-d-def pi-d-def pifilt-def sfxfilt-def*)

lemma *cover*:
 assumes $(\exists\, i<k.\ ff(i) < (j{::}nat) \wedge j \le ff(Suc\ i))$
   $(\forall\, i<k.\ ff(i) < ff(Suc\ i))$
 shows $ff(0)<j \wedge j \le ff(k)$
using *assms*
proof (*induct k arbitrary*:*j*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
 proof −
  have *1*: $(\exists\, i<k.\ ff(i) < (j{::}nat) \wedge j \le ff(Suc\ i)) \vee (ff\ (k) < j \wedge j \le ff(Suc\ k))$
   using *Suc.prems*(*1*) *less-SucE* **by** *blast*
  have *2*: $(\forall\, i<k.\ ff(i)<ff(Suc\ i))$
   using *Suc.prems*(*2*) **by** *auto*
  have *3*: $ff\ k < ff\ (Suc\ k)$
   by (*simp add*: *Suc.prems*(*2*))
  have *4*: $(ff(0)<j \wedge j \le ff(k)) \vee\ (ff\ (k) < j \wedge j \le ff(Suc\ k))$
    using *1 2 Suc.hyps* **by** *blast*
  have *41*: $ff(0)<j$
   by (*metis 2 4 Suc.hyps less-antisym less-le-trans less-or-eq-imp-le zero-less-Suc*)
  have *42*: $j \le ff(Suc\ k)$
   using *3 4* **by** *linarith*
  have *5*: $ff(0)<j \wedge j \le ff(Suc\ k)$
   by (*simp add*: *41 42*)
  show *?thesis*

```
   by (simp add: 5)
  qed
qed


lemma cover-a:
 assumes (∀ j.
          (j≤ ff 0) ∨ (∃ i<k. ff(i) <(j::nat) ∧ j≤ ff(Suc i))
            ⟶ gg j)

         (∀ i<k. ff(i)<ff(Suc i))
 shows   (∀ j< ff k. gg j)
proof –
 have 1: (∀ j< ff 0. gg j)
   by (simp add: assms(1))
 have 2: (∀ j. ff 0 < j ∧ j≤ ff k ⟶ gg j)
   proof
     fix j
     show ff 0 < j ∧ j ≤ ff k ⟶ gg j
      using assms
       proof (induct k arbitrary: j)
       case 0
       then show ?case by simp
       next
       case (Suc k)
       then show ?case
         proof –
          have 21: (∀ j.
          (j≤ ff 0) ∨ (∃ i<k. ff(i) <(j::nat) ∧ j≤ ff(Suc i))
             ∨ (ff k < j ∧ j ≤ ff (Suc k)    )
            ⟶ gg j)
             using Suc.prems(1) less-SucI by blast
          have 22: (∀ i<k. ff(i)<ff(Suc i))
            using Suc.prems(2) by auto
          have 23: ff k < ff (Suc k)
             by (simp add: Suc.prems(2))
          have 24: (∀ j.
                    (j≤ ff 0) ∨ (ff 0 <j ∧ j ≤ ff k)
                     ∨ (ff k < j ∧ j ≤ ff (Suc k)    )
                   ⟶ gg j)
            using 21 22 Suc.hyps by blast
          have 25: ff 0 < j ∧ j ≤ ff (Suc k) ⟶ gg j
                 using 21 22 Suc.hyps not-le by blast
          show ?thesis using 25 by blast
         qed
       qed
    qed
  show ?thesis
  by (metis 2 assms(1) less-or-eq-imp-le linorder-neqE-nat)
qed
```

```
lemma PiUntilDistsem3:
 assumes (σ ⊨ f Π (g 𝒰 h))
 shows   (σ ⊨ ( f Π g ) 𝒰 ( f Π h ) )
proof −
  have 1: ((∃ i≤intlen σ. f (suffix i σ)) ∧
    (∃ k≤intlen (filter f (suffixes σ)).
        h (suffix k (map (λs. nth s 0) (filter f (suffixes σ)))) ∧
        (∀ j<k. g (suffix j (map (λs. nth s 0) (filter f (suffixes σ)))))))
   using assms PiUntilDistsem1 by blast
  have 2: ∃ x ∈ set(suffixes σ). f x
     using 1 in-set-suffixes osfx-suffix by blast
  have 3: ∀ x ∈ set(nfilter f (suffixes σ) 0). f (nth (suffixes σ) x)
   by (simp add: 2)
  have 4: (∃ k≤intlen (filter f (suffixes σ)).
        h (suffix k (map (λs. nth s 0) (filter f (suffixes σ)))) ∧
        (∀ j<k. g (suffix j (map (λs. nth s 0) (filter f (suffixes σ))))))
    using 1 by auto
  obtain k where 5: k≤intlen (filter f (suffixes σ)) ∧
                      h (suffix k (map (λs. nth s 0) (filter f (suffixes σ)))) ∧
                      (∀ j<k. g (suffix j (map (λs. nth s 0) (filter f (suffixes σ)))))
    using 4 by auto
  have 6: f (suffix (nth (nfilter f (suffixes σ) 0) k)  σ)
    by (metis (no-types, lifting) 2 3 5 add.left-neutral nfilter-intlen nfilter-upper-bound
        nth-set nth-suffixes)
  have 7: k=0 ⟶ (∃ i≤intlen σ − k. f (suffix (i + k) σ)) ∧
       h (map (λs. nth s 0) (filter f (suffixes (suffix k σ)))) ∧
       (∀ j<k. (∃ i≤intlen σ − j. f (suffix (i + j) σ)) ∧
              g (map (λs. nth s 0) (filter f (suffixes (suffix j σ)))))
   using 1 5 by auto
  have 71: k>0 ⟶ (nth (nfilter f (suffixes σ) 0) (k −1)) ∈ set(nfilter f (suffixes σ) 0)
    by (metis 2 5 diff-le-self le-trans nfilter-intlen nth-set)
  have 8: k>0 ⟶ f (suffix (nth (nfilter f (suffixes σ) 0) (k −1))  σ)
   by (metis 2 3 71 add.left-neutral interval-nth-and-set nfilter-upper-bound nth-suffixes)
  have 9: k>0 ⟶ (nth (nfilter f (suffixes σ) 0) (k −1)) < (nth (nfilter f (suffixes σ) 0) k)
     by (metis 2 5 One-nat-def Suc-diff-Suc Suc-le-lessD diff-zero idx-nfilter-mono
         nfilter-intlen)
  have 10: k>0 ⟶  (nth (nfilter f (suffixes σ) 0) (k −1)) ≤ intlen σ
    using  nfilter-upper-bound[of suffixes σ f k−1 0]
    by (simp add: 2 5 Suc-leD nfilter-intlen)
    have 11: k>0 ⟶ (nth (nfilter f (suffixes σ) 0) k) ≤ intlen σ
    using nfilter-upper-bound[of suffixes σ f k 0]
    by (simp add: 2 5 nfilter-intlen)
  have 12: k>0 ⟶
           h (map (λs. nth s 0)
                  (filter f (suffixes (suffix (nth (nfilter f (suffixes σ) 0) k) σ))))
     by (metis 11 5 6 filter-nfilter-suffix intlen-suffixes map-suffix nth-suffixes
         suffix-suffixes)
  have 121: k>0 ⟶
```

h (*map* (λ*s*. *nth s 0*)
(*filter f* (*suffixes* (*suffix* (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) (*k*−*1*))) σ))))
by (*metis 2 5 Suc-diff-1 filter-suffixes-map*)
have *13*: *k*>*0* ⟶ (∃ *i*≤*intlen* σ − (*nth* (*nfilter f* (*suffixes* σ) *0*) *k*).
f (*suffix* (*i* + (*nth* (*nfilter f* (*suffixes* σ) *0*) *k*)) σ))
using *6* **by** *auto*
have *131*: *k*>*0* ⟶ (∃ *i*≤*intlen* σ − (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) (*k*−*1*))).
f (*suffix* (*i* + (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) (*k*−*1*)))) σ))
using *11 6 9 diff-le-mono* **by** *fastforce*
have *14*: *k*>*0* ⟶ (∀ *j*< (*nth* (*nfilter f* (*suffixes* σ) *0*) *k*).
(∃ *i*≤*intlen* σ − *j*. *f* (*suffix* (*i* + *j*) σ)))
using *11 6 diff-le-mono* **by** *fastforce*
have *141*: *k*>*0* ⟶ (∀ *j*< (*Suc* (*nth* (*nfilter f* (*suffixes* σ) *0*) (*k*−*1*))).
(∃ *i*≤*intlen* σ − *j*. *f* (*suffix* (*i* + *j*) σ)))
using *14 9* **by** *auto*
have *15*: (∀ *j*<*k*. *g* (*suffix j* (*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes* σ)))))
using *5* **by** *blast*
have *151*: (∀ *j*<*k*. *g* (*map* (λ*s*. *nth s 0*) (*suffix j* (*filter f* (*suffixes* σ)))))
by (*simp add*: *5 less-le-trans less-or-eq-imp-le map-suffix*)
have *152*: (∀ *j*<*k*. (*suffix j* (*filter f* (*suffixes* σ))) =
(*filter f* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) *j*) (*suffixes* σ))))
by (*meson 2 5 filter-nfilter-suffix-1 le-trans less-imp-le-nat*)
have *16*: (∀ *j*<*k*. *g* (*map* (λ*s*. *nth s 0*)
(*filter f* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) *j*) (*suffixes* σ)))))
using *151 152 filter-nfilter-suffix* **by** *simp*
have *1610*: (*nth* (*nfilter f* (*suffixes* σ) *0*) *k*) ≤ *intlen*(*suffixes* σ)
by (*metis 2 5 diff-zero interval-intlen-gr-zero nfilter-intlen*
nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
have *1611*: (∀ *j*<*k*. (*nth* (*nfilter f* (*suffixes* σ) *0*) *j*) < (*nth* (*nfilter f* (*suffixes* σ) *0*) *k*))
by (*simp add*: *2 5 idx-nfilter-gr nfilter-intlen*)
have *1612*: (∀ *j*<*k*. (*nth* (*nfilter f* (*suffixes* σ) *0*) *j*) ≤ *intlen*(*suffixes* σ))
using *1610 1611* **by** *auto*
have *161*: (∀ *j*<*k*. ((*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) *j*) (*suffixes* σ))) =
((*suffixes* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) *j*) σ))))
using *suffix-suffixes* **using** *1612* **by** *blast*
have *17*: (∀ *j*<*k*. *g* (*map* (λ*s*. *nth s 0*)
(*filter f* (*suffixes* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) *j*) σ)))))
using *16 161* **by** *auto*
have *18*: *k*>*0* ⟶
g (*map* (λ*s*. *nth s 0*)
(*filter f* (*suffixes* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) (*k*−*1*)) σ))))
using *17* **by** *simp*
have *19*: *k*>*0* ⟶
g (*map* (λ*s*. *nth s 0*)
(*filter f* (*suffixes* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) *0*) σ))))
using *17* **by** *blast*
have *20*: *k*>*0* ⟶ (*filter f* (*suffixes* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) *0*) σ))) =
(*filter f* (*suffix* (*nth* (*nfilter f* (*suffixes* σ) *0*) *0*) (*suffixes* σ)))
using *161* **by** *auto*
have *21*: *k*>*0* ⟶ (∀ *j*≤ (*nth* (*nfilter f* (*suffixes* σ) *0*) *0*).

($(suffixes\ (suffix\ j\ \sigma))) =$
($(suffix\ j\ (suffixes\ \sigma))))$

using *1612 le-trans suffix-suffixes* **by** *fastforce*

have *22*: $k>0 \longrightarrow (\forall\ j\leq (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0\ )\ 0).$
$(filter\ f\ (suffixes\ (suffix\ j\ \sigma))) =$
$(filter\ f\ (suffix\ j\ (suffixes\ \sigma))))$

using *21* **by** *auto*

have *23*: $k>0 \longrightarrow$
$(\forall\ j\leq (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0\ )\ 0).$
$(map\ (\lambda s.\ nth\ s\ 0)$
$(filter\ f\ (suffixes\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)\ \sigma)))) =$
$(map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ j\ \sigma))))$
)

by (*simp add*: *2 21 filter-suffixes-map-help-0*)

have *24*: $k>0 \longrightarrow$
$(\forall\ j\leq (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0\ )\ 0).$
$g\ (map\ (\lambda s.\ nth\ s\ 0)$
$(filter\ f\ (suffixes\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ 0)\ \sigma)))) =$
$g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ j\ \sigma))))$
)

using *23* **by** *auto*

have *241*: $k>0 \longrightarrow (\forall\ j\leq (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0\ )\ 0).$
$g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ j\ \sigma))))\ )$

using *19 24* **by** *blast*

have *25*: $k>0 \longrightarrow (\forall\ i < k\ -1.$
$(\forall\ l.\ l\leq\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ (Suc\ i)\ ) \wedge$
$(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ i\ ) < l \longrightarrow$
$(filter\ f\ (suffix\ (nth(nfilter\ f\ (suffixes\ \sigma)\ 0)\ (Suc\ i))\ (suffixes\ \sigma)) =$
$(filter\ f\ (suffix\ l\ (suffixes\ \sigma)))\ \ )\ )\ \ )$

proof

assume $k>0$

show $(\forall\ i < k\ -1.$
$(\forall\ l.\ l\leq\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ (Suc\ i)\ ) \wedge$
$(nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ i\ ) < l \longrightarrow$
$(filter\ f\ (suffix\ (nth(nfilter\ f\ (suffixes\ \sigma)\ 0)\ (Suc\ i))\ (suffixes\ \sigma)) =$
$(filter\ f\ (suffix\ l\ (suffixes\ \sigma)))\ \ )\ )\ \ )$

proof

fix i

show $i < k - 1 \longrightarrow$
$(\forall l.\ l \leq nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ (Suc\ i) \wedge$
$nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ i < l \longrightarrow$
$filter\ f\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ (Suc\ i))\ (suffixes\ \sigma)) =$
$filter\ f\ (suffix\ l\ (suffixes\ \sigma)))$

proof –

have *251*: $k=1 \longrightarrow i < k - 1 \longrightarrow$
$(\forall l.\ l \leq nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ (Suc\ i) \wedge$
$nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ i < l \longrightarrow$
$filter\ f\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ \sigma)\ 0)\ (Suc\ i))\ (suffixes\ \sigma)) =$
$filter\ f\ (suffix\ l\ (suffixes\ \sigma)))$

by *auto*

```
      have 252: k>1 ⟶ i < k − 1 ⟶
                (∀ l. l ≤ nth (nfilter f (suffixes σ) 0) (Suc i) ∧
                     nth (nfilter f (suffixes σ) 0) i < l ⟶
                    filter f (suffix (nth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ)) =
                    filter f (suffix l (suffixes σ)))
        proof
         assume k>1
         show i < k − 1 ⟶
                    (∀ l. l ≤ nth (nfilter f (suffixes σ) 0) (Suc i) ∧
                          nth (nfilter f (suffixes σ) 0) i < l ⟶
                     filter f (suffix (nth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ)) =
                     filter f (suffix l (suffixes σ))
                    )
         proof
           assume i < k − 1
           show (∀ l. l ≤ nth (nfilter f (suffixes σ) 0) (Suc i) ∧
                    nth (nfilter f (suffixes σ) 0) i < l ⟶
                 filter f (suffix (nth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ)) =
                 filter f (suffix l (suffixes σ)))
           proof
           fix l
            show l ≤ nth (nfilter f (suffixes σ) 0) (Suc i) ∧
                nth (nfilter f (suffixes σ) 0) i < l ⟶
                 filter f (suffix (nth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ)) =
                 filter f (suffix l (suffixes σ))
             using  filter-suffixes-map-help-j[of l f suffixes σ  i]
              using 2 5 ‹i < k − 1› by linarith
            qed
        qed
      qed
    show ?thesis
    by (simp add: 252)
   qed
  qed
 qed
have 261: k>0 ⟶ (∀  i < k −1.
                (∀  l. l≤  (nth (nfilter f (suffixes σ) 0) (Suc i) ) ∧
                      (nth (nfilter f (suffixes σ) 0) i ) < l ⟶
                      l ≤ intlen (suffixes σ) ))

    by (metis 1612 Suc-diff-1 Suc-mono le-eq-less-or-eq less-le-trans)
have 262: k>0 ⟶ (∀  i < k −1.
                (∀  l. l≤  (nth (nfilter f (suffixes σ) 0) (Suc i) ) ∧
                      (nth (nfilter f (suffixes σ) 0) i ) < l ⟶
                      (suffix l (suffixes  σ)) = (suffixes (suffix l σ)) ))
   using 261 suffix-suffixes by blast
 have 26: k>0 ⟶ (∀  i < k −1.
                (∀  l. l≤  (nth (nfilter f (suffixes σ) 0) (Suc i) ) ∧
                      (nth (nfilter f (suffixes σ) 0) i ) < l ⟶
                 (map (λs. nth s 0)
```

```
                  (filter f (suffix (nth(nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ))))=
                 (map (λs. nth s 0) (filter f (suffixes (suffix l σ)))    ) ) )
    using 25 262 by auto
  have 27: k>0 ⟶ (∀ i < k −1.
                 (∀ l. l≤ (nth (nfilter f (suffixes σ) 0) (Suc i) ) ∧
                      (nth (nfilter f (suffixes σ) 0) i ) < l ⟶
                 g (map (λs. nth s 0)
                   (filter f
                     (suffix (nth(nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ))))))
    by (simp add: 16)
  have 28: k>0 ⟶ (∀ i < k −1.
                 (∀ l. l≤ (nth (nfilter f (suffixes σ) 0) (Suc i) ) ∧
                      (nth (nfilter f (suffixes σ) 0) i ) < l ⟶
                 g (map (λs. nth s 0) (filter f (suffixes (suffix l σ)))    )))
   using 25 262 27 by auto
  have 281: k>0 ⟶
         (∀j.
           (j ≤ (nth (nfilter f (suffixes σ) 0) 0 ) ∨
           (∃ i. i < k −1 ∧
              j≤ (nth (nfilter f (suffixes σ) 0) (Suc i) ) ∧
              (nth (nfilter f (suffixes σ) 0) i ) < j) )
              ⟶ g (map (λs. nth s 0) (filter f (suffixes (suffix j σ)))    ))
      using 241 28 by blast
  have 282: k>0 ⟶ (∀i<k − 1.
                  nth (nfilter f (suffixes σ) 0) i < nth (nfilter f (suffixes σ) 0) (Suc i))
    by (metis 2 5 Suc-diff-1 Suc-leI idx-nfilter-gr le-SucI le-trans lessI nfilter-intlen)
  have 29: k>0 ⟶ (∀j< (Suc (nth (nfilter f (suffixes σ) 0) (k−1))).
                  g (map (λs. nth s 0) (filter f (suffixes (suffix j σ)))))
    using 281 282 cover-a[of λ x. nth (nfilter f (suffixes σ) 0) x k−1
                        (λ j. g (map (λs. nth s 0) (filter f (suffixes (suffix j σ)))))]
    using 18 less-antisym by blast
  have 30: k>0 ⟶ (∃k≤intlen σ.
      (∃i≤intlen σ − k. f (suffix (i + k) σ)) ∧
      h (map (λs. nth s 0) (filter f (suffixes (suffix k σ)))) ∧
      (∀j<k. (∃i≤intlen σ − j. f (suffix (i + j) σ)) ∧
           g (map (λs. nth s 0) (filter f (suffixes (suffix j σ))))))
   using 29 121 131 141
   by (meson 11 9 Suc-leI less-le-trans)
 have 31: (∃k≤intlen σ.
      (∃i≤intlen σ − k. f (suffix (i + k) σ)) ∧
      h (map (λs. nth s 0) (filter f (suffixes (suffix k σ)))) ∧
      (∀j<k. (∃i≤intlen σ − j. f (suffix (i + j) σ)) ∧
           g (map (λs. nth s 0) (filter f (suffixes (suffix j σ))))))
  using 30 7 by blast
  show ?thesis using 31
  by (simp add: PiUntilDistsem2)
qed
```

lemma *PiUntilDistsem4*:
 assumes $(\sigma \models (\ f\ \Pi\ g\ )\ \mathcal{U}\ (\ f\ \Pi\ h\ )\ )$
 shows $(\sigma \models f\ \Pi\ (g\ \mathcal{U}\ h))$
proof −
 have *1*: $(\exists\, k {\le} intlen\ \sigma.$
 $\quad (\exists\, i {\le} intlen\ \sigma - k.\ f\ (suffix\ (i + k)\ \sigma)) \wedge$
 $\quad h\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma)))) \wedge$
 $\quad (\forall\, j{<}k.\ (\exists\, i {\le} intlen\ \sigma - j.\ f\ (suffix\ (i + j)\ \sigma)) \wedge$
 $\qquad g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ j\ \sigma))))))$
  using *assms* **by** (*simp add*: *PiUntilDistsem2*)
 obtain *k* **where** *2*: $k{\le}intlen\ \sigma \wedge (\exists\, i{\le}intlen\ \sigma - k.\ f\ (suffix\ (i + k)\ \sigma)) \wedge$
 $\quad h\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma)))) \wedge$
 $\quad (\forall\, j{<}k.\ (\exists\, i{\le}intlen\ \sigma - j.\ f\ (suffix\ (i + j)\ \sigma)) \wedge$
 $\qquad g\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ j\ \sigma)))))$
  using *1* **by** *auto*
 have *3*: $(\exists\, i{\le}intlen\ \sigma - k.\ f\ (suffix\ (i + k)\ \sigma))$
   using *2* **by** *auto*
 have *4*: $k \le intlen\ \sigma$
  using *2* **by** *auto*
 obtain *i* **where** *5*: $i{\le}intlen\ \sigma - k \wedge f\ (suffix\ (i + k)\ \sigma)$
  using *3* **by** *auto*
 have *6*: $i{+}k \le intlen\ \sigma$
   using *4 5 Nat.le-diff-conv2* **by** *blast*
 have *61*: $\exists\, x \in set\ (suffixes\ (suffix\ k\ \sigma)).\ f\, x$
  by (*metis 4 5 in-set-suffixes interval-suffix-length-good interval-suffix-suffix osfx-suffix*)
 have *7*: $(\exists\, i{\le}intlen\ \sigma.\ f\ (suffix\ i\ \sigma))$
  using *5 6* **by** *blast*
 have *71*: $\exists\, x \in set\ (suffixes\ \sigma).\ f\, x$
  using *5 6 in-set-suffixes osfx-suffix* **by** *blast*
 have *72*: $\exists\ x \in set(nfilter\ f\ (suffixes\ \sigma)\ 0).\ f\ (nth\ (suffixes\ \sigma)\ x)$
  by (*metis 5 6 cancel-comm-monoid-add-class.diff-cancel intlen-suffixes le-refl*
   nfilter-holds-not-a nth-set nth-suffixes
   ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
 have *73*: $f\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ (suffix\ k\ \sigma))\ 0)\ 0)\ \ (suffix\ k\ \sigma))$
   using *nfilter-holds*[*of suffixes* $(suffix\ k\ \sigma)$ *f 0*]
   by (*metis 61 diff-zero interval-intlen-gr-zero nfilter-upper-bound nth-set nth-suffixes*
    ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
 have *74*: $f\ (suffix\ ((nth\ (nfilter\ f\ (suffixes\ (suffix\ k\ \sigma))\ 0)\ 0){+}k)\ \ \sigma)$
   using *73* **by** *auto*
 have *75*: $f\ (suffix\ k\ (suffix\ (nth\ (nfilter\ f\ (suffixes\ (suffix\ k\ \sigma))\ 0)\ 0)\ \sigma))$
  by (*metis 73 add.commute interval-suffix-suffix*)
 have *8*: $h\ (map\ (\lambda s.\ nth\ s\ 0)\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma))))$
   using *2* **by** *auto*
 have *9*: $(filter\ f\ (suffixes\ (suffix\ k\ \sigma))) = (filter\ f\ (suffix\ k\ (suffixes\ \sigma)))$
  by (*simp add*: *4 suffix-suffixes*)
 have *10*: $h\ (map\ (\lambda s.\ nth\ s\ 0)$
 $\quad (suffix\ (intlen(filter\ f\ (suffixes\ \sigma)) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma))))$
 $\qquad (filter\ f\ (suffixes\ \sigma))))$
  using *2 61 sfxfilter-suffix-suffix* **by** *fastforce*
 have *11*: $h\ (suffix\ (intlen(filter\ f\ (suffixes\ \sigma)) - intlen\ (filter\ f\ (suffixes\ (suffix\ k\ \sigma))))$

(*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes σ*))))
by (*metis 10 diff-le-self map-suffix*)
have *12*: (∀ *j*<*k*. (∃ *i*≤*intlen σ* − *j*. *f* (*suffix* (*i* + *j*) *σ*)) ∧
g (*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes* (*suffix j σ*)))))
using *2* **by** *blast*
have *13*: (∀ *j*<*k*. (∃ *x* ∈ *set*(*suffixes* (*suffix j σ*)). *f x*) ∧
g (*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes* (*suffix j σ*)))))
by (*metis 2 interval-suffix-length interval-suffix-suffix intlen-suffixes nth-set nth-suffixes*)
have *14*: (∀ *j*<*k*. (∃ *x* ∈ *set*(*suffixes* (*suffix j σ*)). *f x*) ∧
g (*map* (λ*s*. *nth s 0*)
(*suffix* (*intlen*(*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix j σ*))))
(*filter f* (*suffixes σ*)))))
using *13 sfxfilter-suffix-suffix*
by (*metis* (*no-types*, *lifting*) *2 dual-order.strict-iff-order less-le-trans*)
have *15*: (∀ *j*<*k*. (∃ *x* ∈ *set*(*suffixes* (*suffix j σ*)). *f x*) ∧
g (*suffix* (*intlen*(*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix j σ*))))
(*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes σ*)))))
by (*metis 14 diff-le-self map-suffix*)
have *150*: *k*=*0* ⟶(∀ *jj*<(*intlen*(*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*)))).
g (*suffix jj* (*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes σ*)))))
by *auto*
have *151*: (∀ *jj*<(*intlen*(*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*)))).
(*suffix jj* (*filter f* (*suffixes σ*))) =
filter f (*suffixes* (*suffix* ((*intlen σ*) − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*)) *σ*))
)
by (*simp add*: *71 sfxfilter-suffix-suffix-a*)
have *153*: (∀ *jj*<(*intlen*(*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*)))).
g (*map* (λ*s*. *nth s 0*)
(*filter f*
(*suffixes* (*suffix* ((*intlen σ*) − *intlen* (*nth* (*filter f* (*suffixes σ*)) *jj*)) *σ*))))
)
using *13 sfx-suffix-upperbound* **by** *blast*
have *1530*: (∀ *jj*<(*intlen*(*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*)))).
g (*map* (λ*s*. *nth s 0*) (*suffix jj* (*filter f* (*suffixes σ*)))))
by (*simp add*: *151 153*)
have *1531*: (∀ *jj*<(*intlen*(*filter f* (*suffixes σ*)) − *intlen* (*filter f* (*suffixes* (*suffix k σ*)))).
g (*suffix jj* (*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes σ*)))))
by (*metis 1530 diff-le-self interval-suffix-length interval-suffix-length-code less-le-trans
less-numeral-extra*(*3*) *map-suffix zero-less-diff*)
have *154*: ((∃ *i*≤*intlen σ*. *f* (*suffix i σ*)) ∧
(∃ *k*≤*intlen* (*filter f* (*suffixes σ*)).
h (*suffix k* (*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes σ*)))) ∧
(∀ *j*<*k*. *g* (*suffix j* (*map* (λ*s*. *nth s 0*) (*filter f* (*suffixes σ*)))))))
using *11 1531 7 diff-le-self* **by** *blast*
show *?thesis*
by (*simp add*: *154 PiUntilDistsem1*)
qed

lemma *PiUntilDistsem*:
$\sigma \models f\ \Pi\ (g\ \mathcal{U}\ h) = (\ f\ \Pi\ g\ )\ \mathcal{U}\ (\ f\ \Pi\ h\ )$

using *PiUntilDistsem3 PiUntilDistsem4* **using** *unl-lift2* **by** *blast*

lemma *PiUntilDist*:
$\vdash\ f\ \Pi\ (g\ \mathcal{U}\ h) = (\ f\ \Pi\ g\ )\ \mathcal{U}\ (\ f\ \Pi\ h\ )$
using *PiUntilDistsem Valid-def* **by** *blast*

22.3.15 PiChopstar

lemma *wnextboxnotstatesem*:
assumes $k \le intlen\ \sigma$
shows $(\forall\ j \le intlen\ \sigma.\ k{<}j \longrightarrow \neg\ (\lambda y.\ w\ \langle y\rangle)\ (nth\ \sigma\ j)) =$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (suffix\ k\ \sigma)$

using *assms*
proof (*auto simp add*: *always-defs init-defs wnext-defs*)
show $\bigwedge j.\ j \le intlen\ \sigma \Longrightarrow k < j \Longrightarrow w\ \langle nth\ \sigma\ j\rangle \Longrightarrow$
$\forall\ n{\le}intlen\ \sigma - Suc\ k.\ \neg\ w\ \langle nth\ \sigma\ (Suc\ (n + k))\rangle \Longrightarrow False$
proof (*cases k*)
show $\bigwedge j.\ j \le intlen\ \sigma \Longrightarrow k < j \Longrightarrow w\ \langle nth\ \sigma\ j\rangle \Longrightarrow$
$\forall\ n{\le}intlen\ \sigma - Suc\ k.\ \neg\ w\ \langle nth\ \sigma\ (Suc\ (n + k))\rangle \Longrightarrow k = 0 \Longrightarrow False$
by *simp*
(*metis Suc-le-mono Suc-pred less-le-trans*)
show $\bigwedge j\ nat.\ j \le intlen\ \sigma \Longrightarrow k < j \Longrightarrow w\ \langle nth\ \sigma\ j\rangle \Longrightarrow$
$\forall\ n{\le}intlen\ \sigma - Suc\ k.\ \neg\ w\ \langle nth\ \sigma\ (Suc\ (n + k))\rangle \Longrightarrow k = Suc\ nat \Longrightarrow False$
proof −
fix *j* :: *nat* **and** *nat* :: *nat*
assume *a1*: $k < j$
assume *a2*: $\forall\ n{\le}intlen\ \sigma - Suc\ k.\ \neg\ w\ \langle Interval.nth\ \sigma\ (Suc\ (n + k))\rangle$
assume *a3*: $j \le intlen\ \sigma$
assume *a4*: $w\ \langle nth\ \sigma\ j\rangle$
have $Suc\ (k + (j - Suc\ k)) = j$
using *a1* **by** *simp*
then show *False*
using *a4 a3 a2* **by** (*metis diff-add-inverse2 diff-le-mono*
linordered-semidom-class.add-diff-inverse not-add-less2)
qed
qed
qed

lemma *NotStateUntilStateAndsem*:
$(\sigma \models (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge f)\ ) =$
$(\exists\ k{\le}intlen\ \sigma.\ w\ \langle nth\ \sigma\ k\rangle \wedge f\ (suffix\ k\ \sigma) \wedge (\forall\ j{<}k.\ \neg\ w\ \langle nth\ \sigma\ j\rangle))$

by (*auto simp add*: *until-d-def init-defs*)

lemma *StateUntilEqvWPrevChopsem*:
$\sigma \models (init\ w)\ \mathcal{U}\ f = (wprev\ (\Box\ (init\ w)));f$
by (*auto simp add*: *min.absorb1 until-d-def wprev-defs always-defs init-defs chop-defs*)

lemma *StateUntilEqvWPrevChop*:
$\vdash$ (*init w*) $\mathcal{U}$ *f* = (*wprev* ($\Box$ (*init w*)));*f*
using *StateUntilEqvWPrevChopsem Valid-def* **by** *blast*

lemma *UntilChopDist*:
$\vdash$ (*init w*) $\mathcal{U}$ (*g*;*h*) = ((*init w*) $\mathcal{U}$ *g*) ;*h*
by (*metis ChopAssoc StateUntilEqvWPrevChop inteq-reflection*)

lemma *PiEmptysem*:
$\sigma \models$ (*init w*) Π *empty* = (*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$ *w*))))
proof −
have *1*: ($\sigma \models$ (*init w*) Π *empty*) =
((∃*x*∈*interval.set* σ. *w* ⟨*x*⟩) ∧ *intlen* (*IntervalFilter.filter* (λ*y*. *w* ⟨*y*⟩) σ) = *0*)
by (*simp add*: *init-defs empty-defs Pistate*)
have *2*: ((∃*x*∈*interval.set* σ. *w* ⟨*x*⟩) ∧ *intlen* (*IntervalFilter.filter* (λ*y*. *w* ⟨*y*⟩) σ) = *0*) =
(∃ *k* ≤ *intlen* σ . (λ*y*. *w* ⟨*y*⟩) (*nth* σ *k*) ∧
(∀ *j*. *j*<*k* ⟶ ¬ (λ*y*. *w* ⟨*y*⟩) (*nth* σ *j*)) ∧
(∀ *j* ≤ *intlen* σ. *k*<*j* ⟶ ¬ (λ*y*. *w* ⟨*y*⟩) (*nth* σ *j*)))

by (*simp add*: *filter-length-zero-conv-2*)
have *3*: (∃ *k* ≤ *intlen* σ . (λ*y*. *w* ⟨*y*⟩) (*nth* σ *k*) ∧
(∀ *j*. *j*<*k* ⟶ ¬ (λ*y*. *w* ⟨*y*⟩) (*nth* σ *j*)) ∧
(∀ *j* ≤ *intlen* σ. *k*<*j* ⟶ ¬ (λ*y*. *w* ⟨*y*⟩) (*nth* σ *j*))) =
(∃*k*≤*intlen* σ.
w ⟨*nth* σ *k*⟩ ∧
(*LIFT*(*wnext* ($\Box$ (*init* (¬ *w*))))) (*suffix k* σ) ∧
(∀*j*<*k*. ¬ *w* ⟨*nth* σ *j*⟩))
using *wnextboxnotstatesem*
by *metis*
have *4*: (∃*k*≤*intlen* σ.
w ⟨*nth* σ *k*⟩ ∧
(*LIFT*(*wnext* ($\Box$ (*init* (¬ *w*))))) (*suffix k* σ) ∧
(∀*j*<*k*. ¬ *w* ⟨*nth* σ *j*⟩)) =
($\sigma \models$ (*init* (¬ *w*)) $\mathcal{U}$ ((*init w*) ∧ *wnext* ($\Box$ (*init* (¬ *w*)))))

by (*simp add*: *NotStateUntilStateAndsem*)
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *PiEmpty*:
$\vdash$ (*init w*) Π *empty* = (*init* (¬ *w*)) $\mathcal{U}$ ((*init w*) ∧ *wnext* ($\Box$ (*init* (¬ *w*))))
using *PiEmptysem Valid-def* **by** *blast*

lemma *StatePiBoxStatesem*:
$\sigma \models$ (*init w*) Π *f* = (*init w*) Π (*f* ∧ $\Box$ (*init w*))

proof −
have *1*: $(\sigma \models (init\ w)\ \Pi\ f) =$
$((\exists\ x \in set\ \sigma.\ w\ \langle x\rangle) \wedge\ (\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma) \models\ f))$

by (*metis Pistate*)
have *2*: $(\sigma \models (init\ w)\ \Pi\ (f \wedge \Box\ (init\ w))) =$
$((\exists\ x \in set\ \sigma.\ w\ \langle x\rangle) \wedge\ (\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma) \models f \wedge \Box\ (init\ w)))$

by (*metis Pistate*)
have *3*: $(\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma) \models f \wedge \Box\ (init\ w))$
$= (f\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma) \wedge$
$(\forall\ n \leq intlen\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma).\ w\ \langle nth\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma)\ n\rangle))$
by (*simp add*: *always-defs init-defs*)
have *4*: $((\exists\ x \in set\ \sigma.\ w\ \langle x\rangle) \wedge (f\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma) \wedge$
$(\forall\ n \leq intlen\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma).\ w\ \langle nth\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma)\ n\rangle))) =$
$((\exists\ x \in set\ \sigma.\ w\ \langle x\rangle) \wedge f\ (filter\ (\lambda y.\ w\ \langle y\rangle)\ \sigma))$

by (*meson filter-nth-aa*)
show *?thesis* **using** *1 2 3 4* **by** *auto*
qed

lemma *StatePiBoxState*:
$\vdash (init\ w)\ \Pi\ f = (init\ w)\ \Pi\ (f \wedge \Box\ (init\ w))$
using *StatePiBoxStatesem Valid-def* **by** *blast*

lemma *StatePiUntil1*:
$\vdash (\ (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(wprev\ (\Box\ (init\ (\neg\ w))));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )$
using *StateUntilEqvWPrevChop* **by** *blast*

lemma *StatePiUntilsem2*:
$(\sigma \models (wprev\ (\Box\ (init\ (\neg\ w))));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models ((wprev\ (\Box\ (init\ (\neg\ w))));\ ((init\ w) \wedge empty))\ ;\ ((init\ w) \wedge (init\ w)\ \Pi\ f\ )\ )$
by (*auto simp add*: *chop-defs init-defs empty-defs min.absorb1*)
fastforce

lemma *StatePiUntil2*:
$\vdash (wprev\ (\Box\ (init\ (\neg\ w))));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ ) =$
$(\ ((wprev\ (\Box\ (init\ (\neg\ w))));\ ((init\ w) \wedge empty))\ ;\ ((init\ w) \wedge (init\ w)\ \Pi\ f\ )\ )$
by (*simp add*: *StatePiUntilsem2 Valid-def*)

lemma *StatePiUntil3*:
$\vdash ((wprev\ (\Box\ (init\ (\neg\ w))));\ ((init\ w) \wedge empty)) =$
$(\ ((init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w)))))\ ;\ ((init\ w) \wedge empty))$
proof −
have *1*: $\vdash (wprev\ (\Box\ (init\ (\neg\ w))));\ ((init\ w) \wedge empty) =$
$(init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge empty)$
by (*meson Prop11 StateUntilEqvWPrevChop*)
have *2*: $\vdash ((init\ w) \wedge empty) = ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w))));((init\ w) \wedge empty)$
by (*auto simp add*: *min.absorb1 Valid-def init-defs empty-defs wnext-defs always-defs chop-defs*)

(*metis Suc-pred eq-imp-le interval-intlen-gr-zero le-neq-trans*)
show *?thesis* **by** (*metis 1 2 UntilChopDist inteq-reflection*)
qed

lemma *StatePiUntilsem4*:
$(\sigma \models ((init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w))))) \ ; ((init\ w) \wedge empty)) =$
$(\sigma \models ((init\ w)\ \Pi\ empty); ((init\ w) \wedge empty))$
by (*metis PiEmpty inteq-reflection*)

lemma *StatePiUntil4*:
$\vdash ((init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w))))) \ ; ((init\ w) \wedge empty) =$
$(\ ((init\ w)\ \Pi\ empty); ((init\ w) \wedge empty))$
by (*simp add*: *StatePiUntilsem4 Valid-def*)

lemma *StatePiUntilsem*:
$\sigma \models (init\ w)\ \Pi\ f = (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )$
proof −
have *2*: $(\sigma \models (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models (wprev\ (\Box\ (init\ (\neg\ w))));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ ))$

using *StateUntilEqvWPrevChopsem*[*of LIFT*$(\neg\ w)$ *LIFT*$((init\ w) \wedge (init\ w)\ \Pi\ f)\ \sigma$]
by *simp*
have *7*: $(\sigma \models (wprev\ (\Box\ (init\ (\neg\ w))));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models (((init\ w)\ \Pi\ empty); ((init\ w) \wedge empty));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ ))$

by (*metis PiEmpty StatePiUntil2 StatePiUntil3 inteq-reflection*)
have *8*: $(\sigma \models (((init\ w)\ \Pi\ empty); ((init\ w) \wedge empty));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models (((init\ w)\ \Pi\ empty));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ ))$

by (*auto simp add*: *chop-defs init-defs empty-defs min.absorb1*)
fastforce
have *9*: $(\sigma \models (((init\ w)\ \Pi\ empty));(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models (init\ w)\ \Pi\ (empty;f))$

using *PiChopDistsema PiChopDistsemb* **by** *blast*
have *10*: $(\sigma \models (init\ w)\ \Pi\ (empty;f)) = (\sigma \models (init\ w)\ \Pi\ f)$
by (*metis chop-defs empty-defs interval-prefix-length-good interval-suffix-zero pi-d-def zero-order*(*1*))
show *?thesis*
by (*simp add*: *10 2 7 8 9*)
qed

lemma *StatePiUntil*:
$\vdash (init\ w)\ \Pi\ f = (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )$
using *StatePiUntilsem* **by** *blast*

lemma *StateAndPiEmpty*:
$\vdash\ ((init\ w) \wedge (init\ w)\ \Pi\ empty) = (w \wedge empty)\ ; (wnext\ (\Box\ (init\ (\neg\ w))))$

proof −
have *1*: ⊢ ((*init w*) ∧ (*init w*) Π *empty*) =
((*init w*) ∧ (*init* (¬ *w*)) 𝒰 ((*init w*) ∧ (*wnext* (□ (*init* (¬ *w*))))))
using *PiEmpty* **by** *fastforce*
have *2*: ⊢ ((*init w*) ∧ (*init* (¬ *w*)) 𝒰 ((*init w*) ∧ (*wnext* (□ (*init* (¬ *w*))))))
= ((*init w*) ∧ (*wnext* (□ (*init* (¬ *w*)))))
by (*auto simp add*: *until-d-def Valid-def init-defs*)
force
have *3*: ⊢ ((*init w*) ∧ (*wnext* (□ (*init* (¬ *w*))))) = (*w* ∧ *empty*) ; (*wnext* (□ (*init* (¬ *w*))))
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptyChop inteq-reflection*)
show *?thesis*
using *1 2 3* **by** *fastforce*
qed

lemma *PiPowerExpandsem*:
(σ ⊨ (∃ *k*. (*init w*) Π (*power f k*))) =

(σ ⊨ (*init w*) Π *empty* ∨ (∃ *k*. (*init w*) Π (*f*;(*power f* (*k*)))))
proof −
have *1*: (σ ⊨ (∃ *k*. (*init w*) Π (*power f k*))) =
(∃ *k*. (σ ⊨ (*init w*) Π (*power f k*)))
by *simp*
have *2*: (∃ *k*. (σ ⊨ (*init w*) Π (*power f k*))) =
((σ ⊨ (*init w*) Π (*power f 0*)) ∨ (∃ *k*. *1* ≤ *k* ∧ (σ ⊨ (*init w*) Π (*power f* (*k*)))))
by (*metis One-nat-def diff-Suc-1 le-SucE le-add1 plus-1-eq-Suc*)
have *3*: (σ ⊨ (*init w*) Π (*power f 0*)) = (σ ⊨ (*init w*) Π *empty*)
by *simp*
have *4*: (∃ *k*. *1* ≤ *k* ∧ (σ ⊨ (*init w*) Π (*power f* (*k*)))) =
(∃ *k*. (σ ⊨ (*init w*) Π (*power f* (*Suc k*))))
by (*metis le-add1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
have *5*: (∃ *k*. (σ ⊨ (*init w*) Π (*power f* (*Suc k*)))) =
(σ ⊨ (∃ *k*. (*init w*) Π (*power f* (*Suc k*))))

by *simp*
have *6*: ((σ ⊨ (*init w*) Π *empty*) ∨ (σ ⊨ (∃ *k*. (*init w*) Π (*power f* (*Suc k*))))) =
(σ ⊨ (*init w*) Π *empty* ∨ (∃ *k*. (*init w*) Π (*f*;*power f* (*k*))))
by *auto*
show *?thesis*
using *1 2 3 4 5 6* **by** *blast*
qed

lemma *PiPowerExpandsem1*:
∀ σ. σ ⊨ (∃ *k*. (*init w*) Π (*power f k*)) =
((*init w*) Π *empty* ∨ (∃ *k*. (*init w*) Π (*power f* (*Suc k*))))
proof
fix σ
show σ ⊨ (∃ *k*. (*init w*) Π (*power f k*)) =
((*init w*) Π *empty* ∨ (∃ *k*. (*init w*) Π (*power f* (*Suc k*))))
proof −

have *1*: ($\sigma \models$ ($\exists$ *k*. (*init w*) Π (*power f k*)) =
((*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*power f* (*Suc k*)))))
= (($\sigma \models$ ($\exists$ *k*. (*init w*) Π (*power f k*))) =
($\sigma \models$ (*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*power f* (*Suc k*)))))

by *auto*
have *2*: (($\sigma \models$ ($\exists$ *k*. (*init w*) Π (*power f k*))) =
($\sigma \models$ (*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*power f* (*Suc k*)))))
using *PiPowerExpandsem*[*of w f* σ] **by** *simp*
show *?thesis*
using *1 2* **by** *blast*
qed
qed

lemma *PiPowerExpand*:
$\vdash$ ($\exists$ *k*. (*init w*) Π (*power f k*)) =
((*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*power f* (*Suc k*))))
using *PiPowerExpandsem1*[*of w f*] **by** (*auto simp add*: *Valid-def PiPowerExpandsem1*)

lemma *exists-expand-sem*:
($\sigma \models$ ($\exists$ *k*. (*power* ((*init w*) Π *f* $\wedge$ *fin w*) *k*))) =
(($\sigma \models$ (*power* ((*init w*) Π *f* $\wedge$ *fin w*) *0*)) $\vee$
($\sigma \models$ ($\exists$ *k*. (*power* ((*init w*) Π *f* $\wedge$ *fin w*) (*Suc k*)))))
by (*metis* (*no-types*, *lifting*) *not0-implies-Suc unl-Rex*)

lemma *exists-expand*:
$\vdash$ ($\exists$ *k*. (*power* ((*init w*) Π *f* $\wedge$ *fin w*) *k*)) =
((*power* ((*init w*) Π *f* $\wedge$ *fin w*) *0*) $\vee$ ($\exists$ *k*. (*power* ((*init w*) Π *f* $\wedge$ *fin w*) (*Suc k*))))
using *exists-expand-sem Valid-def* **by** *fastforce*

22.3.16 TruePiEqv

lemma *TruePiEqvsem*:
$\sigma \models \#True\ \Pi\ f = f$
by (*simp add*: *pi-d-def pifilt-true*) *auto*

lemma *TruePiEqv*:
$\vdash (\#True)\ \Pi\ f = f$
using *TruePiEqvsem* **by** (*auto simp add*: *Valid-def*)

22.3.17 BoxImpEqvPi

lemma *BoxImpEqvPi*:
$\vdash \Box\ f \longrightarrow g = f\ \Pi\ g$
by (*simp add*: *Valid-def always-defs pi-d-def pifilt-def sfxfilt-def*)
(*metis filter-True interval-intlen-gr-zero interval-nth-and-set intlen-suffixes*
map-first-suffixes nth-suffixes)

22.3.18 PiEqvDiamondUPi

lemma *PiEqvDiamondUPi*:
$\vdash f\ \Pi\ g = (\ \Diamond\ f \wedge f\ \Pi^u\ g)$
by (*simp add*: *Valid-def upi-d-def sometimes-defs pi-d-def*, *blast*)

22.3.19 PiEqvUntilPi

lemma *PiEqvUntilPi*:
$\vdash (init\ w)\ \Pi\ g = (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w)\ \Pi\ g)$
by (*metis StatePiUntil UntilUntilsem Valid-def inteq-reflection*)

22.3.20 UPiEqvBoxOrPi

lemma *UPiEqvBoxOrPi*:
$\vdash f\ \Pi^u\ g = (\Box\ (\neg\ f) \vee f\ \Pi\ g)$
by (*simp add*: *Valid-def upi-d-def always-defs pi-d-def*, *blast*)

22.4 Theorems

lemma *UPiImpRule*:
assumes $\vdash g1 \longrightarrow g2$
shows $\vdash f\ \Pi^u\ g1 \longrightarrow f\ \Pi^u\ g2$
using *assms*
by (*meson MP PiK PiN*)

lemma *UPiEqvRule*:
assumes $\vdash g1 = g2$
shows $\vdash f\ \Pi^u\ g1 = f\ \Pi^u\ g2$
proof −
have *1*: $\vdash\ g1 \longrightarrow g2$
using *assms* **by** (*simp add*: *int-iffD1*)
have *2*: $\vdash f\ \Pi^u\ g1 \longrightarrow f\ \Pi^u\ g2$
using *1 UPiImpRule* **by** *blast*
have *3*: $\vdash g2 \longrightarrow g1$
using *assms* **by** (*simp add*: *int-iffD2*)
have *4*: $\vdash f\ \Pi^u\ g2 \longrightarrow f\ \Pi^u\ g1$
using *3 UPiImpRule* **by** *blast*
from *3 4* **show** *?thesis*
by (*simp add*: *2 int-iffI*)
qed

lemma *PiEqvNotUPiNot*:
$\vdash\ f\ \Pi\ g = (\neg\ (f\ \Pi^u\ (\neg\ g)))$
by (*simp add*: *upi-d-def*)

lemma *NotPiEqvNotUPi*:
$\vdash\ f\ \Pi\ (\neg\ g) = (\neg\ (f\ \Pi^u\ g))$
by (*simp add*: *upi-d-def*)

lemma *UPiEqvNotPiNot*:

$\vdash f\ \Pi^u\ g = (\neg\ (f\ \Pi\ (\neg\ g)))$
by (*simp add*: *upi-d-def*)

lemma *NotUPiEqvNotPi*:
$\vdash f\ \Pi^u\ (\neg g) = (\neg\ (f\ \Pi\ g))$
by (*simp add*: *upi-d-def*)

lemma *PiImpRule*:
 assumes $\vdash g1 \longrightarrow g2$
 shows $\vdash f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2$
proof −
 have *1*: $\vdash \neg\ g2 \longrightarrow \neg\ g1$
  by (*simp add*: *assms*)
 have *2*: $\vdash f\ \Pi^u\ (\neg\ g2) \longrightarrow f\ \Pi^u\ (\neg g1)$
   using *1 UPiImpRule* **by** *blast*
 have *3*: $\vdash \neg(f\ \Pi^u\ (\neg g1)) \longrightarrow \neg(f\ \Pi^u\ (\neg g2))$
  using *2* **by** *fastforce*
 from *3* **show** *?thesis* **using** *PiEqvNotUPiNot* **by** *fastforce*
qed

lemma *PiEqvRule*:
 assumes $\vdash g1 = g2$
 shows $\vdash f\ \Pi\ g1 = f\ \Pi\ g2$
proof −
 have *1*: $\vdash g1 \longrightarrow g2$
  using *assms* **by** (*simp add*: *int-iffD1*)
 have *2*: $\vdash f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2$
   using *1 PiImpRule* **by** *blast*
 have *3*: $\vdash g2 \longrightarrow g1$
   using *assms* **by** (*simp add*: *int-iffD2*)
 have *4*:$\vdash f\ \Pi\ g2 \longrightarrow f\ \Pi\ g1$
  using *3 PiImpRule* **by** *blast*
 from *2 4* **show** *?thesis* **by** (*simp add*: *int-iffI*)
qed

lemma *UPiAndPiImpPiAnd*:
$\vdash f1\ \Pi^u\ f \land f1\ \Pi\ (\neg\ g) \longrightarrow f1\ \Pi\ (f \land \neg g)$
proof −
 have *1*: $\vdash (\neg(f \longrightarrow g)) = (f \land \neg g)$
  by *fastforce*
 have *2*: $\vdash (\neg\ (f1\ \Pi^u\ (f \longrightarrow g))) = f1\ \Pi\ (\neg(f \longrightarrow g))$
   by (*simp add*: *NotPiEqvNotUPi int-iffD1 int-iffD2 int-iffI*)
 have *3*: $\vdash\ \neg(\ f1\ \Pi^u\ f \longrightarrow f1\ \Pi^u\ g\ ) \longrightarrow \neg\ (f1\ \Pi^u\ (f \longrightarrow g))$
   by (*simp add*: *PiK*)
 have *4*: $\vdash (\neg(\ f1\ \Pi^u\ f \longrightarrow f1\ \Pi^u\ g\ )\ ) = (f1\ \Pi^u\ f \land f1\ \Pi\ (\neg\ g))$
   using *NotPiEqvNotUPi*[*of f1 g*] **by** *fastforce*
 have *5*:$\vdash f1\ \Pi\ (\neg(f \longrightarrow g)) = f1\ \Pi\ (f \land \neg g)$
    using *1* **by** (*simp add*: *PiEqvRule*)
 from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*

qed

lemma *UPiAndPiImpPiAndA*:
$\vdash f1\ \Pi^u\ f \wedge f1\ \Pi\ g \longrightarrow f1\ \Pi\ (f \wedge g)$
using *UPiAndPiImpPiAnd*[*of f1 f LIFT*($\neg g$)] **by** *fastforce*

lemma *PiAndPiImpPiAnd*:
$\vdash f1\ \Pi\ f \wedge f1\ \Pi\ g \longrightarrow f1\ \Pi\ (f \wedge g)$
proof –
 have *1*: $\vdash\ f1\ \Pi^u\ f \wedge f1\ \Pi\ g \longrightarrow f1\ \Pi\ (f \wedge g)$
  using *UPiAndPiImpPiAndA* **by** *fastforce*
 have *2*: $\vdash f1\ \Pi\ f \longrightarrow f1\ \Pi^u\ f$
  using *PiDc* **by** *blast*
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *PiAnd*:
$\vdash f\ \Pi\ (g1 \wedge g2) = (f\ \Pi\ g1 \wedge f\ \Pi\ g2)$
proof –
 have *1*: $\vdash f\ \Pi\ (g1 \wedge g2) \longrightarrow f\ \Pi\ g1$
  by (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)
 have *2*: $\vdash f\ \Pi\ (g1 \wedge g2) \longrightarrow f\ \Pi\ g2$
  by (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)
 have *3*: $\vdash f\ \Pi\ (g1 \wedge g2) \longrightarrow f\ \Pi\ g1 \wedge f\ \Pi\ g2$
  using *1 2* **by** *fastforce*
 have *4*: $\vdash f\ \Pi\ g1 \wedge f\ \Pi\ g2 \longrightarrow f\ \Pi\ (g1 \wedge g2)$
  by (*simp add*: *PiAndPiImpPiAnd*)
 from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *UPiAnd*:
$\vdash f\ \Pi^u\ (g1 \wedge g2) = (\ f\ \Pi^u\ g1 \wedge f\ \Pi^u\ g2\ )$
proof –
 have *1*: $\vdash f\ \Pi\ (\neg g1 \vee \neg\ g2) = (f\ \Pi\ (\neg g1) \vee f\ \Pi\ (\neg\ g2))$
  by (*simp add*: *PiOr*)
 have *2*: $\vdash (\neg(f\ \Pi\ (\neg g1 \vee \neg\ g2))) = (\neg(f\ \Pi\ (\neg g1) \vee f\ \Pi\ (\neg\ g2)))$
  using *1* **by** *fastforce*
 have *3*: $\vdash (\neg(f\ \Pi\ (\neg g1 \vee \neg\ g2))) = f\ \Pi^u\ (\neg(\neg g1 \vee \neg\ g2))$
  by (*meson NotUPiEqvNotPi Prop11*)
 have *4*: $\vdash (\neg(\neg g1 \vee \neg\ g2)) = (g1 \wedge g2)$
  by *fastforce*
 have *5*: $\vdash f\ \Pi^u\ (\neg(\neg g1 \vee \neg\ g2)) = f\ \Pi^u\ (g1 \wedge g2)$
  using *4* **by** (*simp add*: *UPiEqvRule*)
 have *6*: $\vdash (\neg(f\ \Pi\ (\neg g1) \vee f\ \Pi\ (\neg\ g2))) = (\neg(f\ \Pi\ (\neg g1)) \wedge \neg(f\ \Pi(\neg g2)))$
  by *fastforce*

have *7*: $\vdash\neg(f\ \Pi\ (\neg g1)) = f\ \Pi^u\ g1$
  by (*simp add*: *NotPiEqvNotUPi*)
have *8*: $\vdash \neg(f\ \Pi\ (\neg g2)) = f\ \Pi^u\ g2$
  by (*simp add*: *NotPiEqvNotUPi*)
have *9*: $\vdash (\neg(f\ \Pi\ (\neg g1)) \wedge \neg(f\ \Pi(\neg g2))) = (f\ \Pi^u\ g1 \wedge f\ \Pi^u\ g2)$
  using *6 7 8* **by** *fastforce*
from *2 3 5 6 9* **show** *?thesis* **by** *fastforce*
qed

lemma *UpiAndImp*:
$\vdash f\ \Pi^u\ (g1 \longrightarrow g2) \wedge f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2$
proof −
have *2*: $\vdash f\ \Pi^u\ ((\neg g2) \longrightarrow (\neg g1)) \longrightarrow (\ f\ \Pi^u\ (\neg\ g2) \longrightarrow f\ \Pi^u\ (\neg\ g1)\ )$
  using *PiK* **by** *blast*
have *3*: $\vdash (f\ \Pi^u\ (\neg g2) \longrightarrow f\ \Pi^u\ (\neg g1)) = ((\neg\ (f\ \Pi^u\ (\neg g1))) \longrightarrow (\neg(f\ \Pi^u\ (\neg g2))))$
  by *auto*
have *4*: $\vdash (\neg\ (f\ \Pi^u\ (\neg g2))) = f\ \Pi\ g2$
  by (*simp add*: *upi-d-def*)
have *5*: $\vdash (\neg\ (f\ \Pi^u\ (\neg g1))) = f\ \Pi\ g1$
  by (*simp add*: *upi-d-def*)
have *6*: $\vdash f\ \Pi^u\ ((\neg g2) \longrightarrow (\neg g1)) = f\ \Pi^u\ (g1 \longrightarrow g2)$
  by *simp*
have *7*: $\vdash (f\ \Pi^u\ (g1 \longrightarrow g2) \wedge f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2) =$
  $(f\ \Pi^u\ (g1 \longrightarrow g2) \ \longrightarrow (f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2))$
  by *auto*
show *?thesis*
using *2 4 5* **by** *fastforce*
qed

lemma *BoxImpUPiBox*:
$\vdash \Box\ (init\ w) \longrightarrow f\ \Pi^u\ (\Box\ (init\ w))$
proof −
have *1*: $\vdash f\ \Pi\ (\Diamond\ (init\ (\neg w))) \longrightarrow \Diamond\ (init\ (\neg w))$
  by (*simp add*: *PiDiamondImp*)
have *2*: $\vdash \neg\ \ \Diamond\ (init\ (\neg w)) \longrightarrow \neg\ (f\ \Pi\ (\Diamond\ (init\ (\neg w))))$
  using *1* **by** *auto*
have *3*: $\vdash (\neg\ \ \Diamond\ (init\ (\neg w))) = \Box\ (init\ w)$
  by (*metis 2 Initprop*(*2*) *Prop10 always-d-def inteq-reflection*)
have *4*: $\vdash (\neg\ (f\ \Pi\ (\Diamond\ (init\ (\neg w))))) = f\ \Pi^u\ \ (\Box\ (init\ w))$
  by (*simp add*: *upi-d-def*)
  (*metis 3 int-simps*(*4*) *inteq-reflection*)
show *?thesis*
using *2 3 4* **by** *fastforce*
qed

lemma *WPrevPi*:
$\vdash (init\ w)\ \Pi\ f = (wprev\ (\Box\ (init\ (\neg\ w))));\ ((init\ w) \wedge (init\ w)\ \Pi\ f)$
  using *StatePiUntil StatePiUntil1* **by** *fastforce*

lemma *PiChopstarhelp2a*:
$\vdash\ (w \wedge empty);(power\ (\ (f;(w \wedge empty)) \wedge more)\ k) =$
$\quad (power\ (((w \wedge empty);f) \wedge more)\ k\ )\ ;(w \wedge empty)$
proof (*induction k*)
case *0*
then show *?case*
by (*metis ChopEmpty EmptyChop inteq-reflection pow-0*)
next
case (*Suc k*)
then show *?case*
proof −
have *1*: $\vdash (w \wedge empty);power\ (f;(w \wedge empty) \wedge more)\ (Suc\ k)\ =$
$\quad (w \wedge empty);((f;(w \wedge empty) \wedge more);\ power\ (f;(w \wedge empty) \wedge more)\ k)$
by *simp*
have *11*: $\vdash (f \wedge more);(w \wedge empty) = (fin\ w \wedge (f \wedge more))$
by (*metis FinEqvTrueChopAndEmpty TrueChopAndEmptyEqvChopAndEmpty inteq-reflection*)
have *12*: $\vdash (f;(w \wedge empty)) = (fin\ w \wedge f)$
by (*meson AndFinEqvChopAndEmpty Prop04 lift-and-com*)
have *13*: $\vdash (f;(w \wedge empty) \wedge more) = ((fin\ w \wedge f) \wedge more)$
using *12* **by** *auto*
have *14*: $\vdash ((fin\ w \wedge f) \wedge more) = (fin\ w \wedge (f \wedge more))$
by *fastforce*
have *2*: $\vdash (f;(w \wedge empty) \wedge more) = (f \wedge more);(w \wedge empty)$
using *11 13* **by** *fastforce*
have *21*: $\vdash ((f;(w \wedge empty) \wedge more);\ power\ (f;(w \wedge empty) \wedge more)\ k) =$
$\quad ((f \wedge more);\ (power\ ((w \wedge empty);f \wedge more)\ k;(w \wedge empty)))$
by (*metis 2 ChopAssocB Suc.IH int-eq*)
have *3*: $\vdash (w \wedge empty);((f;(w \wedge empty) \wedge more);\ power\ (f;(w \wedge empty) \wedge more)\ k) =$
$\quad (w \wedge empty);((f \wedge more);\ (power\ ((w \wedge empty);f \wedge more)\ k;(w \wedge empty)))$
by (*simp add*: *21 RightChopEqvChop*)
have *4*: $\vdash (w \wedge empty);((f \wedge more);\ (power\ ((w \wedge empty);f \wedge more)\ k;(w \wedge empty))) =$
$\quad ((w \wedge empty);(f \wedge more));\ ((power\ ((w \wedge empty);f \wedge more)\ k;(w \wedge empty)))$
using *ChopAssoc* **by** *blast*
have *5*: $\vdash (w \wedge empty);(f \wedge more) = ((w \wedge empty);f \wedge more)$
by (*auto simp add*: *Valid-def chop-defs empty-defs more-defs*)
have *6*: $\vdash ((w \wedge empty);(f \wedge more));\ ((power\ ((w \wedge empty);f \wedge more)\ k;(w \wedge empty))) =$
$\quad ((w \wedge empty);f \wedge more);\ ((power\ ((w \wedge empty);f \wedge more)\ k;(w \wedge empty)))$
using *5 LeftChopEqvChop* **by** *blast*
have *7*: $\vdash ((w \wedge empty);f \wedge more);\ ((power\ ((w \wedge empty);f \wedge more)\ k;(w \wedge empty))) =$
$\quad ((power\ ((w \wedge empty);f \wedge more)\ (Suc\ k);(w \wedge empty)))$
by (*simp add*: *ChopAssoc*)
show *?thesis*
by (*metis 1 3 4 5 7 int-eq*)
qed
qed

lemma *PiChopstarhelp2*:
$\vdash (w \wedge empty);(f;(w \wedge empty))^{\star} = ((w \wedge empty);f)^{\star};(w \wedge empty)$
proof −

have *1*: $\vdash$ $(w \wedge empty);(f;(w \wedge empty))^{\star} = (w \wedge empty);(\exists\, k.\ power\ (\ (f;(w \wedge empty)) \wedge more)\ k)$
by (*simp add*: *chopstar-d-def powerstar-d-def*)
have *2*: $\vdash$ $(w \wedge empty);(\exists\, k.\ power\ (\ (f;(w \wedge empty)) \wedge more)\ k) =$
$(\exists\, k.\ (w \wedge empty);(power\ (\ (f;(w \wedge empty)) \wedge more)\ k))$
using *ChopExist* **by** *fastforce*
have *3*: $\vdash$ $(\exists\, k.\ (w \wedge empty);(power\ (\ (f;(w \wedge empty)) \wedge more)\ k)) =$
$(\exists\, k.\ (power\ ((w \wedge empty);f \wedge more)\ k);(w \wedge empty))$

by (*simp add*: *ExEqvRule PiChopstarhelp2a*)
have *4*: $\vdash$ $(\exists\, k.\ (power\ ((w \wedge empty);f \wedge more)\ k);(w \wedge empty)) =$
$(\exists\, k.\ (power\ ((w \wedge empty);f \wedge more)\ k));(w \wedge empty)$
using *ExistChop* **by** *fastforce*
have *5*: $\vdash$ $(\exists\, k.\ (power\ ((w \wedge empty);f \wedge more)\ k));(w \wedge empty) =$
$((w \wedge empty);f)^{\star};(w \wedge empty)$
by (*simp add*: *chopstar-d-def powerstar-d-def*)
show *?thesis*
using *1 2 3 4 5* **by** *fastforce*
qed

lemma *PiPowerSuca*:
$\vdash$ $(init\ w)\ \Pi\ (power\ f\ (Suc\ k)) = ((init\ w)\ \Pi\ f);((init\ w) \wedge (init\ w)\ \Pi\ (power\ f\ k))$
by (*simp add*: *PiChopDist*)

lemma *PiPowerSucb*:
$\vdash$ $(init\ w)\ \Pi\ (power\ f\ (Suc\ k)) = ((init\ w)\ \Pi\ f \wedge fin\ w);((init\ w) \wedge\ (init\ w)\ \Pi\ (power\ f\ k))$
by (*metis* (*no-types*, *lifting*) *AndFinChopEqvStateAndChop ChopAssoc ChopImpDiamond FinChopEqvDiamond FinEqvTrueChopAndEmpty InitAndEmptyEqvAndEmpty PiPowerSuca Prop10 StateAndEmptyChop inteq-reflection*)

lemma *PiPowerSucc*:
$\vdash$ $(init\ w)\ \Pi\ (power\ f\ (Suc\ k)) =$
$(power\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ (Suc\ k));((init\ w) \wedge (init\ w)\ \Pi\ empty)$
proof (*induction k*)
case *0*
then show *?case*
by (*metis ChopEmpty PiPowerSucb inteq-reflection pow-0 pow-Suc*)
next
case (*Suc k*)
then show *?case*
by (*metis AndFinChopEqvStateAndChop AndFinEqvChopAndEmpty ChopAssocB InitAndEmptyEqvAndEmpty PiPowerSuca inteq-reflection pow-Suc*)
qed

lemma *PiPowerSucd*:
$\vdash$ $(init\ w)\ \Pi\ (power\ f\ (Suc\ k)) = (power\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ (Suc\ k));((init\ w)\ \Pi\ empty)$
proof (*induction k*)
case *0*
then show *?case*
by (*metis* (*no-types*, *lifting*) *AndFinChopEqvStateAndChop AndFinEqvChopAndEmpty*

InitAndEmptyEqvAndEmpty PiPowerSuca int-eq pow-0 pow-Suc)
next
case (*Suc k*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *ChopAssoc PiPowerSucc inteq-reflection pow-Suc*)
qed

lemma *PiChopstar*:
$\vdash (\mathit{init}\ w)\ \Pi\ (f^\star) = (\mathit{init}\ (\neg\ w))\ \mathcal{U}\ ((\mathit{init}\ w) \wedge (((\mathit{init}\ w)\ \Pi\ f) \wedge \mathit{fin}\ w)^\star;\mathit{wnext}(\Box\ (\mathit{init}\ (\neg\ w))))$
proof −
have *1*: $\vdash (\mathit{init}\ w)\ \Pi\ (f^\star) = (\mathit{init}\ w)\ \Pi\ (\exists\, k.\ \mathit{power}\ f\ k)$
by (*metis ChopstarEqvPowerstar PiEqvRule powerstar-d-def*)
have *2*: $\vdash (\mathit{init}\ w)\ \Pi\ (\exists\, k.\ \mathit{power}\ f\ k) = (\exists\, k.\ (\mathit{init}\ w)\ \Pi\ (\mathit{power}\ f\ k))$
by (*simp add*: *Valid-def pi-d-def*)
have *3*: $\vdash (\exists\, k.\ (\mathit{init}\ w)\ \Pi\ (\mathit{power}\ f\ k)) =$
$(\ (\mathit{init}\ w)\ \Pi\ \mathit{empty} \vee (\exists\, k.\ (\mathit{init}\ w)\ \Pi\ (\mathit{power}\ f\ (\mathit{Suc}\ k))))$
using *PiPowerExpand* **by** *auto*
have *4*: $\vdash (\exists\, k.\ (\mathit{init}\ w)\ \Pi\ (\mathit{power}\ f\ (\mathit{Suc}\ k))) =$
$(\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k));((\mathit{init}\ w)\ \Pi\ \mathit{empty}))$
by (*meson ExEqvRule PiPowerSucd*)
have *5*: $\vdash (\mathit{init}\ w)\ \Pi\ \mathit{empty} = \mathit{empty}\ ;\ ((\mathit{init}\ w)\ \Pi\ \mathit{empty})$
by (*simp add*: *EmptyChop int-iffD1 int-iffD2 int-iffI*)
have *6*: $\vdash (\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k));((\mathit{init}\ w)\ \Pi\ \mathit{empty})) =$
$(\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k)));((\mathit{init}\ w)\ \Pi\ \mathit{empty})$
by (*simp add*: *Semantics.ExistChop*)
have *7*: $\vdash (\ (\mathit{init}\ w)\ \Pi\ \mathit{empty} \vee (\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k));((\mathit{init}\ w)\ \Pi\ \mathit{empty})))=$
$(\ \mathit{empty};\ ((\mathit{init}\ w)\ \Pi\ \mathit{empty}) \vee$
$(\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k)));((\mathit{init}\ w)\ \Pi\ \mathit{empty}))$
using *5 6* **by** *fastforce*
have *8*: $\vdash (\ \mathit{empty};\ ((\mathit{init}\ w)\ \Pi\ \mathit{empty}) \vee$
$(\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k)));((\mathit{init}\ w)\ \Pi\ \mathit{empty})) =$
$(\ \mathit{empty} \vee (\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k))));((\mathit{init}\ w)\ \Pi\ \mathit{empty})$
by (*meson OrChopEqv Prop11*)
have *9*: $\vdash \mathit{empty} = (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ 0)$
by *simp*
have *10*: $\vdash (\ \mathit{empty} \vee (\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k)))) =$
$(\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ 0) \vee (\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k))))$
by *simp*
have *11*: $\vdash (\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ 0) \vee (\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k)))) =$
$(\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ k))$
using *exists-expand*[*of w f*] **by** *fastforce*
have *12*: $\vdash (\ (\mathit{init}\ w)\ \Pi\ \mathit{empty} \vee$
$(\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\mathit{Suc}\ k));((\mathit{init}\ w)\ \Pi\ \mathit{empty}))) =$
$(\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\ k)));\ ((\mathit{init}\ w)\ \Pi\ \mathit{empty})$
by (*metis 11 7 8 9 inteq-reflection*)
have *13*: $\vdash (\exists\, k.\ (\mathit{power}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)\ (\ k)));\ ((\mathit{init}\ w)\ \Pi\ \mathit{empty}) =$
$((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)^\star\ ;\ \ ((\mathit{init}\ w)\ \Pi\ \mathit{empty})$
by (*metis ChopstarEqvPowerstar LeftChopEqvChop inteq-reflection powerstar-d-def*)
have *14*: $\vdash ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{fin}\ w)^\star\ ;\ \ ((\mathit{init}\ w)\ \Pi\ \mathit{empty}) =$
$(\ ((\mathit{init}\ w)\ \Pi\ \mathit{empty}) \vee$

$(((init\ w)\ \Pi\ f \wedge fin\ w);((init\ w)\ \Pi\ f \wedge fin\ w)^\star);((init\ w)\ \Pi\ empty))$
using *CSChopEqvOrChopPlusChop* **by** *blast*
have *15*: $\vdash ((init\ w)\ \Pi\ empty) = (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w))))$
by (*simp add*: *PiEmpty*)
have *16*: $\vdash (((init\ w)\ \Pi\ f \wedge fin\ w);((init\ w)\ \Pi\ f \wedge fin\ w)^\star);((init\ w)\ \Pi\ empty) =$
$(((init\ w)\ \Pi\ f \wedge fin\ w);((init\ w)\ \Pi\ f \wedge fin\ w)^\star);\ wnext\ (\Box\ (init\ (\neg\ w)))$
by (*metis* (*no-types*, *lifting*) *AndFinEqvChopAndEmpty ChopAssoc InitAndEmptyEqvAndEmpty PiChopstarhelp2 StateAndEmptyChop StateAndPiEmpty inteq-reflection*)
have *17*: $\vdash ((init\ w)\ \Pi\ f \wedge fin\ w) = ((init\ w)\ \Pi\ f);((init\ w) \wedge empty)$
by (*metis AndFinEqvChopAndEmpty InitAndEmptyEqvAndEmpty inteq-reflection*)
have *18*: $\vdash ((init\ w)\ \Pi\ f) = wprev(\Box\ (init\ (\neg\ w)));((init\ w) \wedge (init\ w)\ \Pi\ f)$
by (*simp add*: *WPrevPi*)
have *19*: $\vdash wprev(\Box\ (init\ (\neg\ w)));((init\ w) \wedge (init\ w)\ \Pi\ f) =$
$wprev(\Box\ (init\ (\neg\ w)));(((init\ w) \wedge empty)\ ;\ ((init\ w)\ \Pi\ f))$
by (*metis RightChopEqvChop StateAndEmptyChop inteq-reflection*)
have *20*: $\vdash wprev(\Box\ (init\ (\neg\ w)));(((init\ w) \wedge empty)\ ;\ ((init\ w)\ \Pi\ f)) =$
$(init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge ((init\ w)\ \Pi\ f))$
using *18 19 StatePiUntil* **by** *fastforce*
have *21*: $\vdash (((init\ w)\ \Pi\ f \wedge fin\ w);((init\ w)\ \Pi\ f \wedge fin\ w)^\star);\ wnext\ (\Box\ (init\ (\neg\ w))) =$
$(init\ (\neg\ w))\ \mathcal{U}\ ($
$(init\ w) \wedge$
$((((init\ w)\ \Pi\ f) \wedge fin\ w);((init\ w)\ \Pi\ f \wedge fin\ w)^\star);wnext\ (\Box\ (init\ (\neg\ w)))$
$)$
by (*metis 17 18 19 20 StateAndChop UntilChopDist inteq-reflection*)
have *22*: $\vdash ((init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w)))) \vee$
$(init\ (\neg\ w))\ \mathcal{U}\ ($
$(init\ w) \wedge$
$((((init\ w)\ \Pi\ f) \wedge fin\ w);((init\ w)\ \Pi\ f \wedge fin\ w)^\star);wnext\ (\Box\ (init\ (\neg\ w)))$
$)\ ) =$
$(init\ (\neg\ w))\ \mathcal{U}\ \ ($
$((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w)))) \vee$
$(\ (init\ w) \wedge ((((init\ w)\ \Pi\ f) \wedge fin\ w);((init\ w)\ \Pi\ f \wedge fin\ w)^\star);wnext\ (\Box\ (init\ (\neg\ w))))$
$)$
using *UntilOrDist* **by** *fastforce*
have *23*: $\vdash ($
$((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w)))) \vee$
$(\ (init\ w) \wedge ((((init\ w)\ \Pi\ f) \wedge fin\ w);((init\ w)\ \Pi\ f \wedge fin\ w)^\star);wnext\ (\Box\ (init\ (\neg\ w))))$
$) =$
$(\ (init\ w) \wedge (((init\ w)\ \Pi\ f) \wedge fin\ w)^\star;wnext(\Box\ (init\ (\neg\ w))))$
by (*metis* (*no-types*, *lifting*) *CSChopEqvOrChopPlusChop ChopOrEqv StateAndEmptyChop inteq-reflection*)
have *24*: $\vdash (init\ w)\ \Pi\ (f^\star) = (\ (init\ w)\ \Pi\ empty \vee (\exists\, k.\ (init\ w)\ \Pi\ (power\ f\ (Suc\ k))))$
using *1 2 3* **by** *fastforce*
have *25*: $\vdash (\ (init\ w)\ \Pi\ empty \vee (\exists\, k.\ (init\ w)\ \Pi\ (power\ f\ (Suc\ k)))) =$
$(\ (init\ w)\ \Pi\ empty \vee (\exists\, k.\ (power\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ (Suc\ k));((init\ w)\ \Pi\ empty)))$
using *4* **by** *fastforce*
have *26*: $\vdash (\ (init\ w)\ \Pi\ empty \vee$
$(\exists\, k.\ (power\ ((init\ w)\ \Pi\ f \wedge fin\ w)\ (Suc\ k));((init\ w)\ \Pi\ empty))) =$
$((init\ w)\ \Pi\ f \wedge fin\ w)^\star\ ;\ \ ((init\ w)\ \Pi\ empty)$
using *12 13* **by** *fastforce*

have *27*:$\vdash$ ((*init w*) Π *f* $\wedge$ *fin w*)* ; ((*init w*) Π *empty*) =
(*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ (((*init w*) Π *f*) $\wedge$ *fin w*)*;*wnext*($\Box$ (*init* ($\neg$ *w*))))
by (*metis 14 15 16 21 22 23 inteq-reflection*)
from *24 25 26 27* **show** *?thesis* **by** *fastforce*
qed

end

23 Interval Temporal Algebra

theory *ITA*
imports *Fuse Semantics TimeReversal*
begin

23.1 Definition of Set of intervals and Operations on them

type-synonym *'a intervals* = *'a interval set*

definition *lan*:: (*'a*:: *world*) *formula* $\Rightarrow$ *'a intervals*
where *lan f* = { σ . ($\sigma \models f$) }

definition *fusion* :: *'a intervals* $\Rightarrow$ *'a intervals* $\Rightarrow$ *'a intervals* (**infixl** $\cdot$ *70*)
where $X \cdot Y$ = {*fuse* $\sigma 1$ $\sigma 2$| $\sigma 1$ $\sigma 2$. $\sigma 1 \in X \wedge \sigma 2 \in Y \wedge$ *intlast* $\sigma 1$ = *intfirst* $\sigma 2$}

definition *reverse* :: *'a intervals* $\Rightarrow$ *'a intervals* ((*SRev* -) [*85*] *85*)
where (*SRev X*) = {*intrev* σ |σ. $\sigma \in X$ }

definition *sempty* :: *'a intervals* (*SEmpty*)
where
SEmpty $\equiv$ *range St*

definition *smore* :: *'a intervals* (*SMore*)
where
SMore $\equiv$ $-$ *SEmpty*

definition *sskip* :: *'a intervals* (*SSkip*)
where
SSkip $\equiv$ $-$(*SEmpty* $\cup$ (*SMore*$\cdot$*SMore*))

definition *sfalse* :: *'a intervals* (*SFalse*)
where
SFalse $\equiv$ {}

definition *strue* :: *'a intervals* (*STrue*)
where
STrue ≡ −{}

definition *sinit* :: *'a intervals* ⇒ *'a intervals* ((*SInit* -) [*85*] *85*)
where
SInit X ≡ (*X* ∩ *SEmpty*)·*STrue*

definition *sfin* :: *'a intervals* ⇒ *'a intervals* ((*SFin* -) [*85*] *85*)
where
SFin X ≡ *STrue*·(*X* ∩ *SEmpty*)

definition *ssometime* :: *'a intervals* ⇒ *'a intervals* ((*SSometime* -) [*85*] *85*)
where
SSometime X ≡ *STrue*·*X*

definition *salways* :: *'a intervals* ⇒ *'a intervals* ((*SAlways* -) [*85*] *85*)
where
SAlways X ≡ −(*SSometime* (−*X*))

definition *sdi* :: *'a intervals* ⇒ *'a intervals* ((*SDi* -) [*85*] *85*)
where
SDi X ≡ *X*·*STrue*

definition *sbi* :: *'a intervals* ⇒ *'a intervals* ((*SBi* -) [*85*] *85*)
where
SBi X ≡ −(*SDi* (−*X*))

definition *sda* :: *'a intervals* ⇒ *'a intervals* ((*SDa* -) [*85*] *85*)
where
SDa X ≡ *STrue*·*X*·*STrue*

definition *sba* :: *'a intervals* ⇒ *'a intervals* ((*SBa* -) [*85*] *85*)
where
SBa X ≡ −(*SDa* (−*X*))

definition *snext* :: *'a intervals* ⇒ *'a intervals* ((*SNext* -) [*85*] *85*)
where
SNext X ≡ *SSkip*·*X*

definition *swnext* :: *'a intervals* ⇒ *'a intervals* ((*SWnext* -) [*85*] *85*)
where
SWnext X ≡ (−(*SSkip*·(−*X*)))

definition *sprev* :: *'a intervals* ⇒ *'a intervals* ((*SPrev* -) [*85*] *85*)
where
SPrev X ≡ *X*·*SSkip*

definition *swprev* :: *'a intervals* ⇒ *'a intervals* ((*SWprev* -) [*85*] *85*)
where

$SWprev\ X \equiv (-((-X)\cdot SSkip))$

primrec *spower* :: *'a intervals* ⇒ *nat* ⇒ *'a intervals* ((*SPower* - -) [*88*,*88*] *87*)
where
  pwr-0 : $SPower\ X\ 0 \quad = SEmpty$
| *pwr-Suc*: $SPower\ X\ (Suc\ n) = ((X \cap SMore)\cdot(SPower\ X\ n))$

definition *sstar* :: *'a intervals* ⇒ *'a intervals* ((*SStar* -) [*85*] *85*)
where
  $SStar\ X \equiv (\bigcup\ n.\ SPower\ X\ n)$

23.2 Simplification Lemmas

lemma *snot-elim* :
$x \in -X \longleftrightarrow x \notin X$
by *simp*

lemma *sor-elim* :
$x\in (X \cup Y) \longleftrightarrow (x \in X \vee x\in Y)$
by *simp*

lemma *sand-elim* :
$x\in (X \cap Y) \longleftrightarrow (x \in X \wedge x\in Y)$
by *simp*

lemma *sfalse-elim* :
$\sigma \notin SFalse$
by (*simp add*: *sfalse-def*)

lemma *strue-elim* :
$\sigma \in STrue$
by (*simp add*: *strue-def*)

lemma *sempty-elim* :
$\sigma \in SEmpty \longleftrightarrow intlen\ \sigma = 0$
by (*simp add*: *image-iff interval-st-intlen sempty-def*)

lemma *smore-elim* :
$\sigma \in SMore \longleftrightarrow intlen\ \sigma > 0$
by (*simp add*: *sempty-elim smore-def*)

lemma *fusion-iff*:
$\sigma \in X\cdot Y \longleftrightarrow (\exists \sigma 1\ \sigma 2.\ \sigma = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \wedge intlast\ \sigma 1 = intfirst\ \sigma 2)$
by (*unfold fusion-def*) *auto*

lemma *fusion-iff-1*:
$\sigma \in X\cdot Y \longleftrightarrow (\exists\ i\leq intlen\ \sigma.\ (prefix\ i\ \sigma)\in X \wedge (suffix\ i\ \sigma)\in Y\ )$
by (*metis fusion-iff interval-fuse-intlen interval-fuse-prefix-suffix interval-intlast-intfirst*
interval-prefix-fuse interval-suffix-fuse le-add1)

lemma *smore-fusion-smore* :
$\sigma \in (SMore \cdot SMore) \longleftrightarrow intlen\ \sigma > 1$
using *fusion-iff-1*
by (*metis interval-prefix-length-good interval-suffix-length-good less-one not-less not-less-iff-gr-or-eq smore-elim zero-less-diff*)

lemma *sskip-elim* :
$\sigma \in SSkip \longleftrightarrow intlen\ \sigma = 1$
using *sskip-def smore-fusion-smore*
by (*metis One-nat-def Suc-lessI Un-iff less-numeral-extra*(*4*) *sempty-elim smore-def smore-elim snot-elim zero-neq-one*)

lemma *spower-elim-zero* :
$\sigma \in SPower\ X\ 0 \longleftrightarrow \sigma \in SEmpty$
by *simp*

lemma *spower-elim-suc* :
$\sigma \in SPower\ X\ (Suc\ n) \longleftrightarrow \sigma \in (X \cap SMore) \cdot (SPower\ X\ n)$
by *simp*

lemma *spower-elim-suc-1* :
$\sigma \in (X \cap SMore) \cdot (SPower\ X\ n) \longleftrightarrow$
$(\exists \sigma 1\ \sigma 2.\ \sigma = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in X \wedge intlen\ \sigma 1 > 0 \wedge \sigma 2 \in (SPower\ X\ n) \wedge$
$intlast\ \sigma 1 = intfirst\ \sigma 2)$
by (*meson IntD1 IntD2 IntI smore-elim fusion-iff*)

lemma *sstar-elim* :
$\sigma \in SStar\ X \longleftrightarrow (\exists\ n.\ \sigma \in SPower\ X\ n)$
by (*simp add*: *sstar-def*)

lemma *sstar-elim-1* :
$(\exists\ n\ .\ \sigma \in SPower\ X\ n) \longleftrightarrow$
$(\ \sigma \in SPower\ X\ 0 \vee (\exists\ n.\ \sigma \in SPower\ X\ (Suc\ n)))$
by (*metis not0-implies-Suc*)

lemma *spower-suc* :
$(\exists\ n.\ \sigma \in SPower\ X\ (Suc\ n)) \longleftrightarrow$
$(\exists\ n.\ \sigma \in (X \cap SMore) \cdot (SPower\ X\ n))$
by *simp*

lemma *spower-suc-1* :
$(\exists\ n.\ \sigma \in (X \cap SMore) \cdot (SPower\ X\ n)) \longleftrightarrow$
$\sigma \in (X \cap SMore) \cdot (SStar\ X)$
by (*metis fusion-iff sstar-elim*)

lemma *sstar-eqv* :
$\sigma \in SStar\ X \longleftrightarrow$
$(\ \sigma \in SEmpty \vee \sigma \in (X \cap SMore) \cdot (SStar\ X))$
by (*metis spower.simps*(*1*) *spower-elim-suc spower-suc-1 sstar-elim sstar-elim-1*)

lemma *spower-sskip-elim* :
$(\sigma \in SPower\ SSkip\ n) \longleftrightarrow intlen\ \sigma = n$
proof (*induct n arbitrary*: σ)
case *0*
then show *?case* **by** (*auto simp add*: *sempty-elim*)
next
case (*Suc n*)
then show *?case*
  proof *auto*
  show $(\bigwedge\sigma.\ (\sigma \in SPower\ SSkip\ n) = (intlen\ \sigma = n)) \Longrightarrow$
      $\sigma \in SSkip \cap SMore \cdot SPower\ SSkip\ n \Longrightarrow intlen\ \sigma = Suc\ n$
   by (*metis Suc.hyps interval-fuse-intlen plus-1-eq-Suc spower-elim-suc-1 sskip-elim*)
  show $(\bigwedge\sigma.\ (\sigma \in SPower\ SSkip\ n) = (intlen\ \sigma = n)) \Longrightarrow$
      $intlen\ \sigma = Suc\ n \Longrightarrow \sigma \in SSkip \cap SMore \cdot SPower\ SSkip\ n$
  by (*metis Suc.hyps Compl-Un Int-commute add-diff-cancel-left′ fusion-iff-1 inf-sup-aci*(*4*)
    interval-prefix-length-good interval-suffix-length-good le-add1 plus-1-eq-Suc smore-def
    sskip-def sskip-elim)
  qed
qed

lemma *srev-elim*:
$\sigma \in (SRev\ X) \longleftrightarrow intrev\ \sigma \in X$
using *interval-rev-swap* **by** (*auto simp add*: *reverse-def*)

23.3 Algebraic Laws

23.3.1 Commutative Additive Monoid

lemma *UnionCommute*:
$(X{::}'a\ intervals) \cup Y = Y \cup X$
by (*simp add*: *Un-commute*)

lemma *UnionSFalse*:
$X \cup SFalse = X$
by (*simp add*: *sfalse-def*)

lemma *UnionAssoc*:
$(X{::}'a\ intervals) \cup (Y \cup Z) = (X \cup Y) \cup Z$
by (*simp add*: *sup-assoc*)

23.3.2 Boolean algebra

lemma *Huntington*:
$(X{::}'a\ intervals) = -(-X \cup -Y) \cup -(-X \cup Y)$
by *auto*

lemma *Morgan*:
$(X{::}'a\ intervals) \cap Y = -(-X \cup -Y)$
by *auto*

— identities

lemma *STrueTop*:
  STrue $= X \cup -X$
by (*simp add*: *strue-def*)

lemma *SFalseBottom*:
  SFalse $= X \cap -X$
by (*simp add*: *sfalse-def*)

23.3.3 multiplicative monoid

lemma *FusionSEmptyL* :
    SEmpty $\cdot X = X$
using *fusion-iff-1 set-eqI*[*of SEmpty*$\cdot$*X X*]
by (*metis interval-intlen-gr-zero interval-prefix-length-good interval-suffix-zero sempty-elim*)

lemma *FusionSEmptyR* :
    $X \cdot$ *SEmpty* $= X$
using *fusion-iff-1 set-eqI*[*of X*$\cdot$*SEmpty X*]
proof *auto*
 show $\bigwedge x.\ (\bigwedge \sigma\ X\ Y.\ (\sigma \in X \cdot Y) = (\exists\, i \le intlen\ \sigma.\ prefix\ i\ \sigma \in X \wedge suffix\ i\ \sigma \in Y)) \Longrightarrow$
        $((\bigwedge x.\ (x \in X \cdot SEmpty) = (x \in X)) \Longrightarrow X \cdot SEmpty = X) \Longrightarrow x \in X \cdot SEmpty \Longrightarrow x \in X$
 by (*metis diff-diff-cancel diff-zero fusion-iff-1 interval-prefix-intlen*
      interval-suffix-length-good sempty-elim)
  show $\bigwedge x.\ (\bigwedge \sigma\ X\ Y.\ (\sigma \in X \cdot Y) = (\exists\, i \le intlen\ \sigma.\ prefix\ i\ \sigma \in X \wedge suffix\ i\ \sigma \in Y)) \Longrightarrow$
        $((\bigwedge x.\ (x \in X \cdot SEmpty) = (x \in X)) \Longrightarrow X \cdot SEmpty = X) \Longrightarrow x \in X \Longrightarrow x \in X \cdot SEmpty$
   using *sempty-elim fusion-iff-1* **by** *fastforce*
qed

lemma *FusionAssocA*:
assumes $x \in X \cdot (Y \cdot Z)$
shows $x \in (X \cdot Y) \cdot Z$
proof −
 have *1*: $(\exists \sigma 1\ \sigma 2.\ x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge intlast\ \sigma 1 = intfirst\ \sigma 2)$
  using *assms fusion-iff*[*of x X Y* $\cdot$ *Z*] **by** *auto*
 obtain $\sigma 1\ \sigma 2$ **where** *2*: $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge intlast\ \sigma 1 = intfirst\ \sigma 2$
   using *1* **by** *auto*
 have *3*: $(\exists \sigma 3\ \sigma 4.\ \sigma 2 = fuse\ \sigma 3\ \sigma 4 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge intlast\ \sigma 3 = intfirst\ \sigma 4)$
  using *2 fusion-iff*[*of* $\sigma 2$ *Y Z*] **by** *auto*
 obtain $\sigma 3\ \sigma 4$ **where** *4*: $\sigma 2 = fuse\ \sigma 3\ \sigma 4 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge intlast\ \sigma 3 = intfirst\ \sigma 4$
   using *3* **by** *auto*
 have *5*: $x = fuse\ \sigma 1\ (fuse\ \sigma 3\ \sigma 4)$
   using *2 4* **by** *auto*
 have *6*: $x = fuse\ (fuse\ \sigma 1\ \sigma 3)\ \sigma 4$
   using *5 2 4 interval-FusionAssoc interval-intfirst-fuse* **by** *fastforce*
 show *?thesis*
   by (*metis 2 4 6 fusion-iff interval-intfirst-fuse interval-intlast-fuse*)
qed

lemma *FusionAssocB*:
assumes $x \in (X \cdot Y) \cdot Z$
shows $x \in X \cdot (Y \cdot Z)$
proof −
have *1*: $(\exists \sigma 1\ \sigma 2.\ x = \mathit{fuse}\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in X{\cdot}Y \wedge \sigma 2 \in Z \wedge \mathit{intlast}\ \sigma 1 = \mathit{intfirst}\ \sigma 2)$
using *assms fusion-iff* [*of x X·Y Z*] **by** *auto*
obtain $\sigma 1\ \sigma 2$ **where** *2*: $x = \mathit{fuse}\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in X{\cdot}Y \wedge \sigma 2 \in Z \wedge \mathit{intlast}\ \sigma 1 = \mathit{intfirst}\ \sigma 2$
using *1* **by** *auto*
have *3*: $(\exists \sigma 3\ \sigma 4.\ \sigma 1 = \mathit{fuse}\ \sigma 3\ \sigma 4 \wedge \sigma 3 \in X \wedge \sigma 4 \in Y \wedge \mathit{intlast}\ \sigma 3 = \mathit{intfirst}\ \sigma 4)$
using *2 fusion-iff* [*of σ1 X Y*] **by** *auto*
obtain $\sigma 3\ \sigma 4$ **where** *4*: $\sigma 1 = \mathit{fuse}\ \sigma 3\ \sigma 4 \wedge \sigma 3 \in X \wedge \sigma 4 \in Y \wedge \mathit{intlast}\ \sigma 3 = \mathit{intfirst}\ \sigma 4$
using *3* **by** *auto*
have *5*: $x = \mathit{fuse}\ (\mathit{fuse}\ \sigma 3\ \sigma 4)\ \sigma 2$
using *2 4* **by** *auto*
have *6*: $x = \mathit{fuse}\ \sigma 3\ (\mathit{fuse}\ \sigma 4\ \sigma 2)$
using *2 4 interval-FusionAssoc interval-intlast-fuse* **by** *force*
show *?thesis*
by (*metis 2 4 6 fusion-iff interval-intfirst-fuse interval-intlast-fuse*)
qed

lemma *FusionAssoc* :
$X{\cdot}(Y{\cdot}Z) = (X{\cdot}Y){\cdot}Z$
using *set-eqI* [*of X·(Y·Z) (X·Y)·Z*]
FusionAssocA FusionAssocB **by** *blast*

— left and right distributivity

lemma *FusionUnionDistL*:
$(X \cup Y){\cdot}Z = (X{\cdot}Z) \cup (Y{\cdot}Z)$
using *fusion-iff set-eqI* [*of* $(X \cup Y){\cdot}Z$ $(X{\cdot}Z) \cup (Y{\cdot}Z)$]
by (*metis* (*no-types*, *lifting*) *sor-elim*)

lemma *FusionUnionDistR*:
$X{\cdot}(Y \cup Z) = (X{\cdot}Y) \cup (X{\cdot}Z)$
using *fusion-iff set-eqI* [*of* $X{\cdot}(Y \cup Z)$ $(X{\cdot}Y) \cup (X{\cdot}Z)$]
by (*metis* (*no-types*, *lifting*) *sor-elim*)

— left and right annihilation
lemma *SFalseFusion*:
SFalse · *X* = *SFalse*
by (*simp add*: *fusion-def sfalse-def*)

lemma *FusionSFalse*:
X·*SFalse* = *SFalse*
by (*simp add*: *fusion-def sfalse-def*)

— idempotency
lemma *UnionIdem*:
$(X{::}'a\ \mathit{intervals}) \cup X = X$

by *simp*

23.3.4 Subsumption order

lemma *Subsumption*:
$((X :: 'a\ intervals) \subseteq Y) = (X \cup Y = Y)$
by *auto*

23.3.5 Helper lemmas

lemma *FusionRuleR*:
assumes $X \subseteq Y$
shows $Z \cdot X \subseteq Z \cdot Y$
using *assms FusionUnionDistR* **by** (*metis Subsumption*)

lemma *FusionRuleL*:
assumes $X \subseteq Y$
shows $X \cdot Z \subseteq Y \cdot Z$
using *assms* **by** (*metis FusionUnionDistL subset-Un-eq*)

lemma *spower-commutes*:
$(X \cap SMore) \cdot (SPower\ X\ n) = (SPower\ X\ n) \cdot (X \cap SMore)$
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyL FusionSEmptyR*)
next
case (*Suc n*)
then show *?case* **by** (*simp add*: *FusionAssoc*)
qed

lemma *fusion-inductl*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $(SPower\ X\ n) \cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyL*)
next
case (*Suc n*)
then show *?case*
proof −
have *f1*: $X \cdot (SPower\ X\ n \cdot Y) \cup X \cdot Z = X \cdot Z$
by (*metis FusionUnionDistR Suc.hyps assms subset-Un-eq*)
have $X \cdot SPower\ X\ n \cdot Y \cup X \cap SMore \cdot SPower\ X\ n \cdot Y = X \cdot (SPower\ X\ n \cdot Y)$
by (*metis* (*no-types*) *FusionAssoc FusionUnionDistL sup-inf-absorb*)
then have $SPower\ X\ (Suc\ n) \cdot Y \cup Z = X \cdot Z \cup (Y \cup Z)$
using *f1 assms* **by** *auto*
then show *?thesis*
using *assms* **by** *auto*
qed
qed

lemma *fusion-inductr*:
assumes $Y \cup Z{\cdot}X \subseteq Z$
shows $Y{\cdot}(SPower\ X\ n) \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyR*)
next
case (*Suc n*)
then show *?case*
proof −
have *f1*: $Y \cdot SPower\ X\ n \cup Z = Z$
using *Suc.hyps assms* **by** *blast*
have $Y \cdot (X \cap SMore) \cdot SPower\ X\ n = Y \cdot (SPower\ X\ n \cdot (X \cap SMore))$
by (*metis* (*no-types*) *FusionAssoc spower-commutes*)
then have $Y \cdot (X \cap SMore) \cdot SPower\ X\ n \subseteq Z$
using *f1* **by** (*metis* (*no-types*) *FusionAssoc FusionUnionDistL FusionUnionDistR Un-subset-iff assms sup-inf-absorb*)
then show *?thesis*
by (*simp add*: *FusionAssoc*)
qed
qed

lemma *sstar-contlA*:
assumes $x \in Y \cdot (SStar\ X)$
shows $x \in (\bigcup n.\ Y \cdot (SPower\ X\ n))$
proof −
have *1*: $(\exists \sigma 1\ \sigma 2.\ x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in Y \wedge \sigma 2 \in (SStar\ X) \wedge intlast\ \sigma 1 = intfirst\ \sigma 2)$
using *assms* **by** (*simp add*: *fusion-iff*)
obtain $\sigma 1\ \sigma 2$ **where** *2*: $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in Y \wedge \sigma 2 \in (SStar\ X) \wedge intlast\ \sigma 1 = intfirst\ \sigma 2$
using *1* **by** *auto*
have *3*: $(\exists\ n.\ \sigma 2 \in SPower\ X\ n)$
using *2 sstar-elim* **by** *blast*
obtain *n* **where** *4*: $\sigma 2 \in SPower\ X\ n$
using *3* **by** *auto*
have *5*: $(\exists\ n.\ x \in Y{\cdot}SPower\ X\ n)$
using *2 4 fusion-iff* **by** *blast*
from *5* **show** *?thesis* **by** *blast*
qed

lemma *sstar-contlB*:
assumes $x \in (\bigcup n.\ Y \cdot (SPower\ X\ n))$
shows $x \in Y \cdot (SStar\ X)$
proof −
have *1*: $\exists n.\ x \in Y{\cdot}\ (SPower\ X\ n)$
using *assms* **by** *blast*
obtain *n* **where** *2*: $x \in Y{\cdot}\ (SPower\ X\ n)$
using *1* **by** *auto*
have *3*: $(\exists \sigma 1\ \sigma 2.\ x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in Y \wedge \sigma 2 \in (SPower\ X\ n) \wedge intlast\ \sigma 1 = intfirst\ \sigma 2)$

using 2 **by** (*simp add*: *fusion-iff*)
obtain $\sigma 1$ $\sigma 2$ **where** 4: $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in Y \wedge \sigma 2 \in (SPower\ X\ n) \wedge intlast\ \sigma 1 = intfirst\ \sigma 2$
using 3 **by** *auto*
have 5: $\sigma 2 \in (SStar\ X)$
using 4 *sstar-elim* **by** *auto*
from 5 4 **show** *?thesis* **using** *fusion-iff* **by** *blast*
qed

lemma *sstar-contl*:
$Y \cdot (SStar\ X) = (\bigcup n.\ Y \cdot (SPower\ X\ n))$
using *set-eqI*[*of* $Y \cdot (SStar\ X)$ $(\bigcup n.\ Y \cdot (SPower\ X\ n))$]
by (*metis sstar-contlA sstar-contlB*)

lemma *sstar-contrA*:
assumes $x \in (SStar\ X) \cdot Y$
shows $x \in (\bigcup n.\ (SPower\ X\ n) \cdot Y)$
proof −
have 1: $(\exists \sigma 1\ \sigma 2.\ x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in (SStar\ X) \wedge \sigma 2 \in Y \wedge intlast\ \sigma 1 = intfirst\ \sigma 2)$
using *assms* **by** (*simp add*: *fusion-iff*)
obtain $\sigma 1$ $\sigma 2$ **where** 2: $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in (SStar\ X) \wedge \sigma 2 \in Y \wedge intlast\ \sigma 1 = intfirst\ \sigma 2$
using 1 **by** *auto*
have 3: $\exists n.\ \sigma 1 \in (SPower\ X\ n)$
using 2 *sstar-elim* **by** *blast*
obtain *n* **where** 4: $\sigma 1 \in (SPower\ X\ n)$
using 3 **by** *auto*
have 5: $(\exists n.\ x \in (SPower\ X\ n) \cdot Y)$
by (*metis 2 4 fusion-iff-1 interval-fuse-intlen interval-prefix-fuse interval-suffix-fuse le-add1*)
from 5 **show** *?thesis* **by** *blast*
qed

lemma *sstar-contrB*:
assumes $x \in (\bigcup n.\ (SPower\ X\ n) \cdot Y)$
shows $x \in (SStar\ X) \cdot Y$
proof −
have 1: $\exists n.\ x \in (SPower\ X\ n) \cdot Y$
using *assms* **by** *blast*
obtain *n* **where** 2: $x \in (SPower\ X\ n) \cdot Y$
using 1 **by** *auto*
have 3: $(\exists \sigma 1\ \sigma 2.\ x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in (SPower\ X\ n) \wedge \sigma 2 \in Y \wedge intlast\ \sigma 1 = intfirst\ \sigma 2)$
using 2 **by** (*simp add*: *fusion-iff*)
obtain $\sigma 1$ $\sigma 2$ **where** 4: $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in (SPower\ X\ n) \wedge \sigma 2 \in Y \wedge intlast\ \sigma 1 = intfirst\ \sigma 2$
using 3 **by** *auto*
have 5: $\sigma 1 \in (SStar\ X)$
using 4 *sstar-elim* **by** *auto*
from 5 4 **show** *?thesis* **using** *fusion-iff* **by** *blast*
qed

lemma *sstar-contr*:
  $(SStar\ X) \cdot Y = (\bigcup n.\ (SPower\ X\ n) \cdot Y)$
using *set-eqI*[*of* $(SStar\ X) \cdot Y\ (\bigcup n.\ (SPower\ X\ n) \cdot Y)$]
by (*metis sstar-contrA sstar-contrB*)

23.3.6 Kleene Algebra

— left unfold
lemma *UnfoldL*:
 $SEmpty \cup X \cdot (SStar\ X) = (SStar\ X)$
proof −
 have *1*: $(SStar\ X) = SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$
  by (*meson Un-iff set-eqI sstar-eqv*)
 have *2*: $(X \cap SMore) \cdot (SStar\ X) \subseteq X \cdot (SStar\ X)$
  by (*simp add*: *FusionRuleL*)
 have *3*: $(SStar\ X) \subseteq SEmpty \cup X \cdot (SStar\ X)$
  using *1 2* **by** *blast*
 have *4*: $SEmpty \subseteq (SStar\ X)$
  using *1* **by** *auto*
 have *5*: $X \subseteq SEmpty \cup (X \cap SMore)$
  by (*simp add*: *Un-Int-distrib smore-def*)
 have *6*: $X \cdot (SStar\ X) \subseteq (SStar\ X) \cup (X \cap SMore) \cdot (SStar\ X)$
  using *5* **by** (*metis FusionRuleL FusionUnionDistL FusionSEmptyL*)
 have *7*: $(SStar\ X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$
  using *1* **by** *auto*
 have *8*: $X \cdot (SStar\ X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$
  using *6 7* **by** *blast*
 hence *9*: $X \cdot (SStar\ X) \subseteq (SStar\ X)$
  using *1* **by** *auto*
 have *10*: $SEmpty \cup X \cdot (SStar\ X) \subseteq (SStar\ X)$
  using *9 4* **by** *simp*
 from *3 10* **show** *?thesis* **by** *auto*
qed

— Left induction
lemma *SStarInductL*:
 assumes $Y \cup X \cdot Z \subseteq Z$
 shows $(SStar\ X) \cdot Y \subseteq Z$
by (*metis UN-least assms fusion-inductl sstar-contr*)

— Right induction
lemma *SStarInductR*:
 assumes $Y \cup Z \cdot X \subseteq Z$
 shows $Y \cdot (SStar\ X) \subseteq Z$
using *sstar-contl assms fusion-inductr* **by** *blast*

23.3.7 ITL specific Laws

lemma *PwrFusionInterL*:
 $((((SPower\ SSkip\ n) \cap X) \cdot V) \cap (((SPower\ SSkip\ n) \cap Y) \cdot W)) =$

$(((SPower\ SSkip\ n) \cap X \cap Y)\cdot(V \cap W))$
using *set-eqI*[*of* $((((SPower\ SSkip\ n) \cap X)\cdot V) \cap (((SPower\ SSkip\ n) \cap Y)\cdot W))$
$(((SPower\ SSkip\ n) \cap X \cap Y)\cdot(V \cap W))$]
by (*simp add*: *fusion-iff-1 spower-sskip-elim*)
(*metis min.absorb1*)

lemma *PwrFusionInterR*:
$((V\cdot((SPower\ SSkip\ n) \cap X)) \cap ((W\cdot((SPower\ SSkip\ n) \cap Y)))) =$
$((V \cap W)\cdot((SPower\ SSkip\ n) \cap X \cap Y))$
using *set-eqI*[*of* $((V\cdot((SPower\ SSkip\ n) \cap X)) \cap ((W\cdot((SPower\ SSkip\ n) \cap Y))))$
$((V \cap W)\cdot((SPower\ SSkip\ n) \cap X \cap Y))$]
by (*simp add*: *fusion-iff-1 spower-sskip-elim*)
(*metis diff-diff-cancel*)

lemma *SSkipFusionImpSMore*:
$SSkip\cdot STrue \subseteq SMore$
using *subsetI*[*of* $SSkip\cdot STrue\ SMore$]
by (*auto simp add*: *fusion-iff-1 sskip-elim smore-elim strue-elim*)

lemma *SStarSkip*:
$(SStar\ SSkip) = STrue$
using *set-eqI*[*of* $(SStar\ SSkip)\ STrue$]
by (*simp add*: *strue-def spower-sskip-elim sstar-elim*)

23.4 Derived Laws

23.4.1 Helper Lemmas

lemma *B01*:
assumes $(X{::}\ 'a\ intervals) \subseteq Y$
shows $-Y \subseteq -X$
using *assms* **by** *auto*

lemma *B04*:
$((X{::}\ 'a\ intervals) = Y\ ) \longleftrightarrow (X \subseteq Y) \wedge (Y \subseteq X)$
by *auto*

lemma *B09*:
assumes $-X \cup Y = STrue$
shows $(X{::}\ 'a\ intervals) \subseteq Y$
using *assms* **using** *strue-def* **by** *auto*

lemma *B20*:
$(X{::}\ 'a\ intervals) \subseteq Y \cup Z \longleftrightarrow X \cap -Y \subseteq Z$
by *auto*

lemma *B28*:
$((X{::}\ 'a\ intervals) \cap Y) \cup (X \cap Z) = X \cap (Y \cup Z)$
by *auto*

lemma *CH01*:
$STrue \cdot STrue = STrue$
by (*metis FusionSEmptyR FusionUnionDistR Int-commute SStarSkip STrueTop UnfoldL inf-sup-absorb*)

lemma *CH07*:
$(((SSkip \cap X) \cdot V) \cap ((SSkip \cap Y) \cdot W)) = ((SSkip \cap X \cap Y) \cdot (V \cap W))$
using *PwrFusionInterL*[*of 1 X V Y W*]
by (*simp add*: *FusionSEmptyR inf-commute smore-def sskip-def*)

lemma *CH08*:
$((V \cdot (SSkip \cap X)) \cap ((W \cdot (SSkip \cap Y)))) = ((V \cap W) \cdot (SSkip \cap X \cap Y))$
using *PwrFusionInterR*[*of V 1 X W Y*]
by (*simp add*: *FusionSEmptyR inf-commute smore-def sskip-def*)

lemma *CH09*:
$(((X \cap SEmpty) \cdot V) \cap ((Y \cap SEmpty) \cdot W)) = (((X \cap Y) \cap SEmpty) \cdot (V \cap W))$
using *PwrFusionInterL*[*of 0 X V Y W*]
by (*metis* (*no-types*, *lifting*) *inf-assoc inf-commute pwr-0*)

lemma *CH10*:
$((V \cdot (X \cap SEmpty)) \cap ((W \cdot (Y \cap SEmpty)))) = ((V \cap W) \cdot ((X \cap Y) \cap SEmpty))$
using *PwrFusionInterR*[*of V 0 X W Y*]
by (*metis* (*no-types*, *lifting*) *inf-assoc inf-commute pwr-0*)

lemma *ST13*:
$((X \cap SEmpty) \cdot Z) \cap ((Y \cap SEmpty) \cdot Z) = ((X \cap Y) \cap SEmpty) \cdot Z$
by (*simp add*: *CH09*)

lemma *ST15*:
$(SStar\ (X \cap SEmpty)) = SEmpty$
by (*metis FusionSEmptyL inf.right-idem inf-le2 UnfoldL SStarInductR sup.orderE sup-inf-absorb*)

lemma *ST21*:
$((-X) \cap SEmpty) \cup (X \cap SEmpty) = SEmpty$
by *blast*

lemma *ST24*:
$(SInit\ X) \cap (SInit\ Y) = (SInit\ (X \cap Y))$
by (*simp add*: *ST13 sinit-def*)

lemma *ST25*:
$(SInit\ STrue) = STrue$
by (*simp add*: *sinit-def strue-def FusionSEmptyL*)

lemma *ST26*:
$(SInit\ (-X)) \cup (SInit\ X) = STrue$
by (*metis Compl-disjoint2 ST21 ST25 Un-Int-distrib compl-bot-eq FusionUnionDistL sinit-def strue-def sup-bot.right-neutral sup-top-right*)

lemma *ST28*:
$(SDi\ (SInit\ X)) = (SInit\ X)$
by (*metis compl-bot-eq FusionAssoc FusionUnionDistR FusionSEmptyR sdi-def sinit-def strue-def sup-top-right UnionCommute*)

lemma *ST33*:
$(STrue \cap SEmpty) \cdot SEmpty = SEmpty$
by (*simp add*: *strue-def FusionSEmptyL*)

lemma *ST36*:
$(SInit\ (-X)) \subseteq -(SInit\ X)$
by (*metis Compl-disjoint ST24 compl-bot-eq disjoint-eq-subset-Compl double-complement inf.coboundedI2 inf.orderE sfalse-def SFalseFusion sinit-def strue-def*)

lemma *ST37*:
$-(SInit\ X) \subseteq (SInit\ (-X))$
using *B09 ST26* **by** *auto*

lemma *ST38*:
$-(SInit\ X) = (SInit\ (-X))$
using *ST37 ST36* **by** *auto*

lemma *ST47*:
$X \cup Y \cdot X = (SEmpty \cup Y) \cdot X$
by (*simp add*: *FusionUnionDistL FusionSEmptyL*)

lemma *SStar01*:
assumes $X \cdot (SStar\ Y) \cup SEmpty \subseteq (SStar\ Y)$
shows $(SStar\ X) \subseteq (SStar\ Y)$
using *assms*
by (*metis Un-commute FusionSEmptyR SStarInductL*)

lemma *SStar03*:
$(SStar\ X) \cdot (SStar\ X) \subseteq (SStar\ X)$
by (*metis SStarInductL Un-absorb UnfoldL sup.absorb-iff1 sup.right-idem*)

lemma *SStar05*:
assumes $((SStar\ X) \cdot (SStar\ X)) \cup SEmpty \subseteq (SStar\ X)$
shows $(SStar\ (SStar\ X)) \subseteq (SStar\ X)$
using *assms*
by (*simp add*: *SStar01*)

lemma *SStar12*:
$(SEmpty \cup (X \cdot (SStar\ X))) \subseteq (SStar\ X)$
using *UnfoldL* **by** *blast*

lemma *SStar06*:
$((SStar\ X) \cdot (SStar\ X)) \cup SEmpty \subseteq (SStar\ X)$
using *SStar03 SStar12* **by** *force*

lemma *SStar07*:
  $(SStar\ X) \subseteq (SStar\ (SStar\ X))$
by (*metis FusionUnionDistR FusionSEmptyR Subsumption Un-commute UnfoldL ST47 sup.right-idem*)

lemma *SStar08*:
  $(SStar\ X) = (SStar\ (SStar\ X))$
by (*meson B04 SStar05 SStar06 SStar07*)

lemma *SStar15*:
  $SEmpty \subseteq (SStar\ SSkip)$
by (*simp add*: *SStarSkip strue-def*)

lemma *SStar16*:
  $SSkip \subseteq (SStar\ SSkip)$
by (*simp add*: *SStarSkip strue-def*)

lemma *SStar22*:
  $(SEmpty \cap X) \cdot (SStar\ (SEmpty \cap X)) = (SEmpty \cap X)$
by (*metis ST15 FusionSEmptyR inf-commute*)

lemma *SStar23*:
  $(SStar\ (SEmpty \cap X)) = SEmpty$
using *SStar22 UnfoldL* **by** *auto*

lemma *SStar25*:
  $(SStar\ STrue) = STrue$
by (*metis SStar08 SStarSkip*)

lemma *SStar28*:
  $(SStar\ X) \cdot X \subseteq X \cdot (SStar\ X)$
by (*metis B04 FusionUnionDistR FusionSEmptyR UnfoldL SStarInductL*)

lemma *SStar29*:
  $X \cdot (SStar\ X) \subseteq (SStar\ X) \cdot X$
by (*metis B04 SStar28 SStarInductR UnfoldL FusionRuleL ST47 sup.mono*)

lemma *SStar17*:
  $(SStar\ SSkip) \cdot SSkip \subseteq SSkip \cdot (SStar\ SSkip)$
by (*simp add*: *SStar28*)

lemma *SStar18*:
  $SSkip \cdot (SStar\ SSkip) \subseteq (SStar\ SSkip) \cdot SSkip$
by (*simp add*: *SStar29*)

lemma *SStar19*:
  $(SStar\ SSkip) \cdot SSkip = SSkip \cdot (SStar\ SSkip)$
using *SStar17 SStar18* **by** *auto*

lemma *SStar30*:
  $X \cdot (SStar\ X) = (SStar\ X) \cdot X$

using *SStar28 SStar29* **by** *auto*

lemma *SStar34*:
assumes *SEmpty* ∪ (*X* ∪ *Y*)·((*SStar X*)·(*SStar* (*Y*·(*SStar X*)))) ⊆ (*SStar X*)·(*SStar* (*Y*·(*SStar X*)))
shows (*SStar* (*X* ∪ *Y*)) ⊆ (*SStar X*)·(*SStar* (*Y*·(*SStar X*)))
by (*metis assms FusionSEmptyR SStarInductL*)

lemma *SStar35*:
SEmpty ∪ (*X* ∪ *Y*)·((*SStar X*)·(*SStar* (*Y*·(*SStar X*)))) ⊆ (*SStar X*)·(*SStar* (*Y*·(*SStar X*)))
by (*simp add*: *FusionAssoc FusionUnionDistL ST47 UnfoldL UnionAssoc UnionCommute*)

lemma *SStar36*:
(*SStar* (*X* ∪ *Y*)) ⊆ (*SStar X*)·(*SStar* (*Y*·(*SStar X*)))
using *SStar34 SStar35* **by** *blast*

lemma *SStar46*:
(*SStar X*)·(*SStar* (*Y*·(*SStar X*))) ⊆ (*SStar* (*X* ∪ *Y*))
proof −
have (*SEmpty* ∪ *SStar* (*X* ∪ *Y*) · *Y*) · *SStar X* ⊆ *SStar* (*X* ∪ *Y*)
by (*metis* (*no-types*) *FusionUnionDistR SStar12 SStar30 SStarInductR sup.bounded-iff*)
then show *?thesis* **by** (*simp add*: *SStarInductR ST47 FusionAssoc*)
qed

lemma *SStar47*:
(*SStar Z*) = (*SStar* (*Z* ∩ *SMore*))
proof −
have *1*: (*SStar Z*) = (*SStar* ((*SEmpty* ∩ *Z*) ∪ (*SMore* ∩ *Z*)))
by (*metis Int-Un-distrib2 compl-bot-eq inf-top.left-neutral smore-def strue-def STrueTop*)
have *2*: (*SStar* ((*SEmpty* ∩ *Z*) ∪ (*SMore* ∩ *Z*))) =
(*SStar* (*SEmpty* ∩ *Z*))·(*SStar* ((*SMore* ∩ *Z*)·(*SStar* (*SEmpty* ∩ *Z*))))
by (*simp add*: *SStar36 SStar46 subset-antisym*)
have *3*: (*SStar* (*SEmpty* ∩ *Z*))·(*SStar* ((*SMore* ∩ *Z*)·(*SStar* (*SEmpty* ∩ *Z*)))) =
(*SStar* (*Z* ∩ *SMore*))
by (*simp add*: *FusionSEmptyL FusionSEmptyR SStar23 inf-commute*)
from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *SStar48*:
(*SStar SMore*) = *STrue*
by (*metis Compl-Un Compl-disjoint2 SStar25 SStar47 ST21 ST33 FusionSEmptyR*
inf.right-idem smore-def strue-def)

lemma *SStar50*:
assumes *SSkip*·((−*X*) ∪ ((*SStar SSkip*)·(*X* ∩ (*SSkip*·(−*X*))))) ∪ (−*X*)
⊆ (−*X*) ∪ (*SStar SSkip*)·(*X* ∩ (*SSkip*·(−*X*)))
shows ((*SStar SSkip*)·(−*X*)) ⊆ ((−*X*) ∪ ((*SStar SSkip*)·(*X* ∩ (*SSkip*·(−*X*)))))
using *SStarInductL assms* **by** *blast*

lemma *SStar51*:
SSkip·((−*X*) ∪ ((*SStar SSkip*)·(*X* ∩ (*SSkip*·(−*X*))))) ∪ (−*X*)

$\subseteq (-X) \cup (SStar\ SSkip) \cdot (X \cap (SSkip \cdot (-X)))$
by (*metis B20 FusionAssoc FusionUnionDistR Morgan ST47 UnfoldL UnionIdem inf.idem inf-commute le-sup-iff sup-ge1*)

lemma *SStar52*:
$(SStar\ X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$
by (*metis B04 SStar47 UnfoldL*)

lemma *SStar53*:
$SEmpty \cup (X \cap SMore) \cdot (SStar\ X) \subseteq (SStar\ X)$
by (*metis SStar12 SStar47*)

lemma *BD45*:
$(SBi\ ((-X) \cup X1)) \cap (X \cdot Y) \subseteq X1 \cdot Y$
proof −
have *1*: $(SBi\ ((-X) \cup X1)) = -(-((-X) \cup X1) \cdot (Y \cup (-Y)))$
by (*metis sbi-def sdi-def STrueTop*)
have *2*: $-(-((-X) \cup X1) \cdot (Y \cup (-Y))) \cap (X \cdot Y) \subseteq$
$-(-((-X) \cup X1) \cdot (Y)) \cap (X \cdot Y)$
using *FusionUnionDistR* **by** *fastforce*
have *3*: $-(-((-X) \cup X1) \cdot (Y)) \cap (X \cdot Y) \subseteq (((-X) \cup X1) \cap X) \cdot Y$
by (*metis* (*no-types*, *hide-lams*) *B20 FusionRuleL FusionUnionDistL Morgan UnionCommute double-compl order-refl*)
have *4*: $(((-X) \cup X1) \cap X) \cdot Y \subseteq X1 \cdot Y$
by (*metis B20 double-compl FusionRuleL inf.right-idem inf-le1*)
from *1 2 3 4* **show** *?thesis* **by** *blast*
qed

lemma *BD46*:
$(SAlways\ ((-Y) \cup Y1)) \cap (X1 \cdot Y) \subseteq (X1 \cdot Y1)$
proof −
have *1*: $(SAlways\ ((-Y) \cup Y1)) = -((X1 \cup (-X1)) \cdot (-((-Y) \cup Y1)))$
by (*metis salways-def ssometime-def STrueTop*)
have *2*: $-((X1 \cup (-X1)) \cdot (-((-Y) \cup Y1))) \cap (X1 \cdot Y) \subseteq$
$-(X1 \cdot (-((-Y) \cup Y1))) \cap (X1 \cdot Y)$
using *FusionUnionDistL* **by** *fastforce*
have *3*: $-(X1 \cdot (-((-Y) \cup Y1))) \cap (X1 \cdot Y) \subseteq X1 \cdot (((-Y) \cup Y1) \cap Y)$
by (*metis* (*no-types*, *lifting*) *B20 B04 compl-inf FusionUnionDistR Huntington Morgan Subsumption sup-ge1 UnionCommute*)
have *4*: $X1 \cdot (((-Y) \cup Y1) \cap Y) \subseteq (X1 \cdot Y1)$
by (*metis B20 double-compl FusionRuleR inf.right-idem inf-le1*)
from *1 2 3 4* **show** *?thesis* **by** *blast*
qed

23.4.2 ITL Axioms derived

lemma *SBoxGen*:
assumes $X = STrue$
shows $(SAlways\ X) = STrue$
using *assms*

by (*metis double-compl FusionSFalse salways-def sfalse-def ssometime-def strue-def*)

lemma *SBiGen*:
 assumes $X = STrue$
 shows $(SBi\ X) = STrue$
using *assms*
by (*metis double-compl sbi-def sdi-def sfalse-def SFalseFusion strue-def*)

lemma *SMP*:
 assumes $X \subseteq Y$
 assumes $X = STrue$
 shows $Y = STrue$
using *assms*(*1*) *assms*(*2*)
using *strue-def* **by** *blast*

lemma *SChopAssoc*:
 $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
by (*simp add*: *FusionAssoc*)

lemma *SOrChopImp*:
 $(X \cup Y) \cdot Z \subseteq (X \cdot Z) \cup (Y \cdot Z)$
by (*simp add*: *FusionUnionDistL*)

lemma *SChopOrImp*:
 $X \cdot (Y \cup Z) \subseteq (X \cdot Y) \cup (X \cdot Z)$
by (*simp add*: *FusionUnionDistR*)

lemma *SEmptyChop*:
 $SEmpty \cdot X = X$
by (*simp add*: *FusionSEmptyL*)

lemma *SChopEmpty*:
 $X \cdot SEmpty = X$
by (*simp add*: *FusionSEmptyR*)

lemma *SStateImpBi*:
 $(SInit\ X) \subseteq (SBi\ (SInit\ X))$
by (*simp add*: *ST28 ST38 sbi-def*)

lemma *SNextImpNotNextNot*:
 $(SNext\ X) \subseteq -(SNext\ (-X))$
proof −
 have *1*: $((SNext\ X) \subseteq -(SNext\ (-X))) = (((SNext\ X) \cap (SNext\ (-X))) \subseteq SFalse)$
  by (*simp add*: *disjoint-eq-subset-Compl sfalse-def*)
 have *2*: $((SNext\ X) \cap (SNext\ (-X))) = SSkip \cdot (X \cap (-X))$
  by (*metis CH07 SStar16 inf.orderE snext-def*)
 have *3*: $(SSkip) \cdot (X \cap (-X)) = SSkip \cdot SFalse$
  by (*simp add*: *sfalse-def*)
 have *4*: $SSkip \cdot SFalse = SFalse$ **by** (*simp add*: *FusionSFalse*)
 from *1 2 3 4* **show** *?thesis* **by** *auto*

qed

lemma *SBiBoxChopImpChop*:
  $(SBi\ ((-X) \cup X1)) \cap (SAlways\ ((-Y) \cup Y1)) \ \cap (X \cdot Y) \subseteq (X1 \cdot Y1)$
using *BD45 BD46* **by** *blast*

lemma *SBoxInduct*:
 $(SAlways\ (-X \cup (SWnext\ X))) \cap X \subseteq (SAlways\ X)$
proof −
 have *1*: $((SAlways\ (-X \cup (SWnext\ X))) \cap X \subseteq (SAlways\ X)) =$
  $((SSometime\ (-X)) \subseteq ((-X) \cup (SSometime\ (X \cap (SNext\ (-X)))\ )))$
  by (*simp add*: *salways-def snext-def swnext-def*)
   blast
 have *2*: $((SSometime\ (-X)) \subseteq ((-X) \cup (SSometime\ (X \cap (SNext\ (-X)))\ ))) =$
  $(\ ((SStar\ SSkip) \cdot (-X)) \subseteq ((-X) \cup ((SStar\ SSkip) \cdot (X \cap (SSkip \cdot (-X)))))\ \ )$
  by (*simp add*: *SStarSkip snext-def ssometime-def*)
 have *3*: $(\ ((SStar\ SSkip) \cdot (-X)) \subseteq ((-X) \cup ((SStar\ SSkip) \cdot (X \cap (SSkip \cdot (-X)))))\ \ )$
  using *SStar51 SStar50* **by** *blast*
 from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *SChopstarEqv*:
  $(SStar\ X) = SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$
using *SStar52 SStar53* **by** *blast*

23.5 Extra Laws

23.5.1 Boolean Laws

lemma *B02*:
 assumes $-Y \subseteq -X$
 shows $(X :: 'a\ intervals) \subseteq Y$
using *assms* **by** *auto*

lemma *B03*:
 $((X :: 'a\ intervals) = Y\ ) \longleftrightarrow (-X = -Y)$
by *auto*

lemma *B05*:
 assumes $(X :: 'a\ intervals) \cup Y \subseteq Z$
 shows $X \subseteq Z \wedge Y \subseteq Z$
using *assms* **by** *auto*

lemma *B06*:
 assumes $X \subseteq Z \wedge Y \subseteq Z$
 shows $(X :: 'a\ intervals) \cup Y \subseteq Z$
using *assms* **by** *auto*

lemma *B07*:
 $(X :: 'a\ intervals) \cup Y \subseteq Z \longleftrightarrow$
  $X \subseteq Z \wedge Y \subseteq Z$

by *auto*

lemma *B08*:
 assumes $(X{::}\ 'a\ intervals) \subseteq Y$
 shows $-X \cup Y = STrue$
using *assms*
using *strue-def* **by** *auto*

lemma *B10*:
 $(X{::}\ 'a\ intervals) \subseteq Y \longleftrightarrow -X \cup Y = STrue$
using *strue-def* **by** *auto*

lemma *B11*:
 assumes $(X{::}\ 'a\ intervals) \subseteq Y$
 shows $X \cap -Y = SFalse$
using *assms sfalse-def* **by** *auto*

lemma *B12*:
 assumes $X \cap -Y = SFalse$
 shows $(X{::}\ 'a\ intervals) \subseteq Y$
using *assms sfalse-def* **by** *auto*

lemma *B13*:
 $(X{::}\ 'a\ intervals) \subseteq Y \longleftrightarrow X \cap -Y = SFalse$
using *sfalse-def* **by** *auto*

lemma *B14*:
 assumes $(X{::}\ 'a\ intervals) \subseteq Y$
 shows $X \cap Y = X$
using *assms* **by** *auto*

lemma *B15*:
 assumes $(X{::}\ 'a\ intervals) \subseteq Y \cap Z$
 shows $X \subseteq Y \wedge X \subseteq Z$
using *assms* **by** *auto*

lemma *B16*:
 assumes $X \subseteq Y \wedge X \subseteq Z$
 shows $(X{::}\ 'a\ intervals) \subseteq Y \cap Z$
using *assms* **by** *auto*

lemma *B17*:
 $(X{::}\ 'a\ intervals) \subseteq Y \cap Z \longleftrightarrow X \subseteq Y \wedge X \subseteq Z$
by *auto*

lemma *B18*:
 assumes $(X{::}\ 'a\ intervals) \subseteq Y \cup Z$
 shows $X \cap -Y \subseteq Z$
using *assms* **by** *auto*

lemma *B19*:
assumes $X \cap -Y \subseteq Z$
shows $(X{::}\ 'a\ intervals) \subseteq Y \cup Z$
using *assms* **by** *auto*

lemma *B21*:
$(X{::}\ 'a\ intervals) \cup (Y \cap Z) \subseteq (X \cup Y) \cap (X \cup Z) \longleftrightarrow$
$X \cup (Y \cap Z) \subseteq (X \cup Y) \wedge X \cup (Y \cap Z) \subseteq (X \cup Z)$
by *auto*

lemma *B22*:
$(X{::}\ 'a\ intervals) \cup (Y \cap Z) \subseteq X \cup Y$
by *auto*

lemma *B23*:
$(X{::}\ 'a\ intervals) \cup (Y \cap Z) \subseteq X \cup Z$
by *auto*

lemma *B24*:
$((X{::}\ 'a\ intervals) \cup Y) \cap (X \cup Z) \subseteq X \cup (Y \cap Z) \longleftrightarrow$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z$
by *auto*

lemma *B25*:
$(((X{::}\ 'a\ intervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z \longleftrightarrow$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \wedge$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$
by *auto*

lemma *B26*:
$(((X{::}\ 'a\ intervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y$
by *auto*

lemma *B27*:
$(((X{::}\ 'a\ intervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$
by *auto*

lemma *B29*:
$(X{::}\ 'a\ intervals) \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
by *auto*

23.5.2 Chop

lemma *CH02*:
$X \cdot Y \cap -(X \cdot Z) \subseteq X \cdot (Y \cap -Z)$
by (*metis B20 FusionRuleR FusionUnionDistR inf .right-idem inf-le1*)

lemma *CH03*:
$X \cdot Y \cap -(Z \cdot Y) \subseteq (X \cap -Z) \cdot Y$
by (*metis B20 FusionRuleL FusionUnionDistL inf .right-idem inf-le1*)

lemma *CH04*:
  $X \cdot Y \cap -(X \cdot -Z) \subseteq X \cdot (Y \cap Z)$
using *CH02* **by** *fastforce*

lemma *CH05*:
  $X \cdot Y \cap -(-Z \cdot Y) \subseteq (X \cap Z) \cdot Y$
using *CH03* **by** *fastforce*

lemma *CH06*:
 assumes $X \subseteq X1$
         $Y \subseteq Y1$
 shows $X \cdot Y \subseteq X1 \cdot Y1$
using *assms*
by (*metis FusionRuleL FusionRuleR order-trans*)

lemma *CH11*:
  $((X \cap (SPower\ SSkip\ n)) \cdot STrue) \cap ((SPower\ SSkip\ n) \cdot Y) = (X \cap (SPower\ SSkip\ n)) \cdot Y$
using *PwrFusionInterL*[*of n X STrue STrue Y*]
by (*simp add*: *inf-commute strue-def*)

lemma *CH12*:
  $(STrue \cdot (X \cap (SPower\ SSkip\ n))) \cap (Y \cdot (SPower\ SSkip\ n)) = (Y \cdot (X \cap (SPower\ SSkip\ n)))$
using *PwrFusionInterR*[*of STrue n X Y STrue*]
by (*metis STrueTop inf-commute inf-sup-absorb*)

lemma *CH13*:
  $(SPower\ SSkip\ n) \cdot (SPower\ SSkip\ m) = (SPower\ SSkip\ (n+m))$
proof
 (*induct n arbitrary*: *m*)
 case *0*
 then show *?case* **by** (*simp add*: *FusionSEmptyL*)
 next
 case (*Suc n*)
 then show *?case*
 by (*metis FusionAssoc add-Suc pwr-Suc*)
qed

23.5.3 Next

lemma *N01*:
  $(SNext\ SEmpty) = SSkip$
by (*simp add*: *FusionSEmptyR snext-def*)

lemma *N02*:
  $(SNext\ SFalse) = SFalse$
by (*simp add*: *FusionSFalse snext-def*)

lemma *N03*:
  $(SNext\ X) \cdot Y = (SNext\ (X \cdot Y))$

by (*simp add*: *snext-def FusionAssoc*)

lemma *N04*:
  (*SNext* $(X \cup Y)$) = (*SNext* X) $\cup$ (*SNext* Y)
by (*simp add*: *FusionUnionDistR snext-def*)

lemma *N05*:
  (*SNext* $(X \cap Y)$) = (*SNext* X) $\cap$ (*SNext* Y)
by (*metis CH07 SStar16 inf .orderE snext-def*)

lemma *N06*:
 assumes $X \subseteq Y$
 shows (*SNext* X) $\subseteq$ (*SNext* Y)
using *assms*
by (*metis FusionUnionDistR Subsumption snext-def*)

lemma *N07*:
  (*SNext* $((-X) \cup Y)$) = (*SNext* $(-X)$) $\cup$ (*SNext* Y)
by (*simp add*: *N04*)

lemma *N08*:
  SMore $\subseteq$ *SSkip*·*STrue*
by (*simp add*: *smore-def*)
  (*metis B10 SStarSkip UnfoldL double-complement*)

lemma *N23*:
 (*SWprev* X) $\subseteq$ (*SEmpty* $\cup$ (*SPrev* X))
proof −
have X · *SSkip* $\cup$ − X · *SSkip* = *SStar SSkip* · *SSkip*
by (*metis* (*no-types*) *Compl-empty-eq FusionUnionDistL SStarSkip strue-def sup-compl-top*)
then have − *SWprev* X $\cup$ (*SEmpty* $\cup$ *SPrev* X) = *STrue*
by (*metis* (*no-types*) *SStar19 SStarSkip UnfoldL UnionAssoc double-compl sprev-def sup-commute swprev-def*)
then show *?thesis*
by (*meson B09*)
qed

lemma *N24*:
  (*SEmpty*) $\subseteq$ (*SWprev* X)
by (*metis B10 B02 FusionRuleL SSkipFusionImpSMore SStar30 SStarSkip UnfoldL compl-bot-eq double-compl smore-def strue-def subset-antisym swprev-def top-greatest*)

lemma *N25*:
  (*SPrev* X) $\subseteq$ (*SWprev* X)
proof −
 have *1*: ((*SPrev* X) $\subseteq$ (*SWprev* X)) = (((*SPrev* X) $\cap$ (*SPrev* $(-X)$))$\subseteq$ *SFalse*)
   by (*simp add*: *B10 sfalse-def sprev-def swprev-def*)
 have *2*: ((*SPrev* X) $\cap$ (*SPrev* $(-X)$)) = $(X \cap (-X))$·*SSkip*
   by (*metis CH08 SStar16 inf .orderE sprev-def*)
 have *3*: $(X \cap (-X))$·*SSkip* = *SFalse*·*SSkip*

by (*simp add*: *sfalse-def*)
have *4*: *SFalse·SSkip* = *SFalse*
by (*simp add*: *SFalseFusion*)
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *N26*:
(*SWprev X*) = (*SEmpty* ∪ (*SPrev X*))
using *N23 N24 N25* **by** *blast*

lemma *N09*:
SSkip ∪ *SMore·SSkip* ⊆ *SMore*
proof −
have *1*: *SSkip* ⊆ *SMore* **by** (*simp add*: *smore-def sskip-def*)
have *2*: *SMore·SSkip* ⊆ *SMore*
by (*metis N26 UnionCommute compl-le-swap1 smore-def sup-ge2 swprev-def*)
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *N10*:
assumes *SSkip* ∪ *SMore·SSkip* ⊆ *SMore*
shows *SSkip·*(*SStar SSkip*) ⊆ *SMore*
using *assms*
using *SStarInductR N09* **by** *blast*

lemma *N11*:
SSkip · *STrue* ⊆ *SMore*
by (*metis SStarSkip N09 N10*)

lemma *N12*:
(*SNext X*) = −(*SWnext* (−*X*))
by (*simp add*: *snext-def swnext-def*)

lemma *N13*:
SMore·STrue = *SMore*
by (*metis FusionAssoc N11 N08 SStar48 SStarSkip ST47 UnfoldL subset-antisym sup.right-idem*)

lemma *N14*:
STrue·SSkip ⊆ *SMore*
by (*metis N11 SStar19 SStarSkip*)

lemma *N15*:
SMore ⊆ *STrue·SSkip*
by (*metis N08 SStar19 SStarSkip*)

lemma *N19*:
(*SWnext X*) ⊆ (*SEmpty* ∪ (*SNext X*))
proof −
have *SSkip·X* ∪ *SSkip·*(−*X*) = *SSkip·SStar SSkip*
using *FusionUnionDistR*[*of SSkip X* −*X*] *SStarSkip*

by (*metis STrueTop*)
then have $-$ *SWnext X* $\cup$ (*SEmpty* $\cup$ *SNext X*) $=$ *STrue*
using *B08 N08 SStarSkip smore-def snext-def swnext-def* **by** *fastforce*
then show *?thesis* **by** (*simp add*: *B09*)
qed

lemma *N20*:
(*SEmpty*) $\subseteq$ (*SWnext X*)
proof $-$
have *1*: ((*SEmpty*) $\subseteq$ (*SWnext X*)) $=$ ($-$(*SWnext X*) $\subseteq$ *SMore*)
by (*simp add*: *smore-def*)
have *2*: ($-$(*SWnext X*) $\subseteq$ *SMore*) $=$ ((*SNext* ($-X$)) $\subseteq$ *SMore*)
by (*simp add*: *snext-def swnext-def*)
have *3*: (*SNext* ($-X$)) $\subseteq$ *SSkip·STrue*
by (*metis FusionUnionDistR STrueTop snext-def sup.orderI sup.right-idem*)
hence *4*: (*SNext* ($-X$)) $\subseteq$ *SMore* **using** *SSkipFusionImpSMore* **by** *auto*
from *1 2 4* **show** *?thesis* **by** *auto*
qed

lemma *N21*:
(*SEmpty* $\cup$ (*SNext X*)) $\subseteq$ (*SWnext X*)
using *N20*
by (*metis B06 SNextImpNotNextNot snext-def swnext-def*)

lemma *N22*:
(*SWnext X*) $=$ (*SEmpty* $\cup$ (*SNext X*))
using *N21 N19* **by** *blast*

lemma *N16*:
(*SNext X*) $=$ *SMore* $\cap$ (*SWnext X*)
using *N12 N22 smore-def* **by** *blast*

lemma *N17*:
(*SWnext* ($X \cap Y$)) $=$ (*SWnext X*) $\cap$ (*SWnext Y*)
by (*simp add*: *N05 N22 Un-Int-distrib*)

lemma *N18*:
(*SWnext* ($X \cup Y$)) $=$ (*SWnext X*) $\cup$ (*SWnext Y*)
by (*simp add*: *swnext-def*)
(*metis* (*no-types*, *lifting*) *CH07 SStar16 compl-inf inf.orderE*)

lemma *N27*:
(*SNext* (($-X$) $\cup$ Y)) $\subseteq$ ($-$(*SNext X*) $\cup$ (*SNext Y*))
by (*metis N12 N16 N18 Un-Int-distrib double-compl sup-ge2 sup-left-idem*)

lemma *N28*:
(*SPrev* (($-X$) $\cup$ Y)) $\subseteq$ ($-$(*SPrev X*) $\cup$ (*SPrev Y*))
by (*metis B01 B05 B06 FusionUnionDistL Huntington N25 double-compl sprev-def sup-ge2 swprev-def*)

lemma *N29*:

$(SPrev\ X) = -(SWprev\ (-X)\ )$
by (*simp add*: *sprev-def swprev-def*)

23.5.4 SInit

lemma *ST01*:
$(X \cap SEmpty) \cdot Y \subseteq Y$
by (*metis Int-commute FusionUnionDistL FusionSEmptyL le-iff-sup sup-inf-absorb UnionCommute*)

lemma *ST02*:
$(X \cap SEmpty) \cdot Y \subseteq (X \cap SEmpty) \cdot STrue$
by (*simp add*: *FusionRuleR strue-def*)

lemma *ST03*:
$(X \cap SEmpty) \cdot (X \cap SEmpty) \subseteq (X \cap SEmpty)$
using *ST01* **by** *auto*

lemma *ST04*:
$(X \cap SEmpty) \subseteq (X \cap SEmpty) \cdot (X \cap SEmpty)$
by (*metis B04 Int-commute FusionSEmptyL FusionSEmptyR inf .right-idem inf-top.right-neutral CH10*)

lemma *ST05*:
$(X \cap SEmpty) \subseteq -((-X) \cap SEmpty)$
by *blast*

lemma *ST06*:
$((-X) \cap SEmpty) \subseteq -(X \cap SEmpty)$
by *auto*

lemma *ST07*:
$(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq (X \cap SEmpty) \cdot STrue$
using *ST02 FusionSEmptyR* **by** *blast*

lemma *ST08*:
$(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq (STrue \cap SEmpty) \cdot (Y \cap SEmpty)$
by (*metis FusionSEmptyL FusionSEmptyR ST33 inf .cobounded2*)

lemma *ST09*:
$((X \cap SEmpty) \cdot STrue) \cap (STrue \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (X \cap SEmpty) \cdot (Y \cap SEmpty)$
by (*metis compl-bot-eq eq-refl FusionAssoc FusionSEmptyR inf .commute inf-top.left-neutral CH09 strue-def*)

lemma *ST10*:
$(X \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (X \cap SEmpty)$
by (*metis FusionRuleR FusionSEmptyR inf-le2*)

lemma *ST11*:
$(X \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (Y \cap SEmpty)$
using *ST01* **by** *blast*

lemma *ST12*:
$(X \cap SEmpty) \cap (Y \cap SEmpty) = (X \cap SEmpty)\cdot SEmpty \cap (Y \cap SEmpty)\cdot SEmpty$
by (*simp add*: *FusionSEmptyR*)

lemma *ST14*:
$((X \cap Y) \cap SEmpty)\cdot SEmpty = ((X \cap Y) \cap SEmpty)$
by (*simp add*: *FusionSEmptyR*)

lemma *ST16*:
$(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq SEmpty$
by (*simp add*: *le-infI2*)

lemma *ST17*:
$(X \cap SEmpty)\cdot(Y \cap SEmpty) \subseteq SEmpty$
using *ST10* **by** *auto*

lemma *ST18*:
$-((X \cap SEmpty) \cup (Y \cap SEmpty)) = -(X \cap SEmpty) \cap -(Y \cap SEmpty)$
by *auto*

lemma *ST19*:
$(X \cap SEmpty)\cdot((-X) \cap SEmpty) \subseteq (X \cap SEmpty)$
using *ST10* **by** *blast*

lemma *ST20*:
$(X \cap SEmpty)\cdot((-X) \cap SEmpty) \subseteq ((-X) \cap SEmpty)$
using *ST01* **by** *auto*

lemma *ST22*:
$((X \cap SEmpty)\cdot SSkip)\cdot(Y \cap SEmpty) \subseteq (X \cap SEmpty)\cdot SSkip$
using *FusionRuleR FusionSEmptyR* **by** *blast*

lemma *ST23*:
$((X \cap SEmpty)\cdot SSkip)\cdot(Y \cap SEmpty) \subseteq SSkip\cdot(Y \cap SEmpty)$
by (*simp add*: *ST01 FusionRuleL*)

lemma *ST27*:
$(SInit\ X) \cap (Y\cdot Z) \subseteq ((SInit\ X) \cap Y)\cdot Z$
by (*metis B04 compl-bot-eq FusionAssoc FusionSEmptyL inf-commute inf-top.left-neutral CH09 sinit-def strue-def*)

lemma *ST29*:
$(SInit\ X)\cdot Y \subseteq (SInit\ X)$
using *ST02 FusionAssoc sinit-def* **by** *fastforce*

lemma *ST30*:
$(SInit\ X) \cap (SDi\ Y) = (SDi\ (\ (SInit\ X) \cap Y))$
by (*metis FusionAssoc FusionSEmptyL CH09 compl-bot-eq inf-top.left-neutral sdi-def sinit-def strue-def*)

lemma *ST31*:
$(X \cdot (STrue \cap SEmpty)) \cap (STrue \cdot (Y \cap SEmpty)) = X \cdot (Y \cap SEmpty)$
by (*metis Int-commute compl-bot-eq inf-top.right-neutral CH10 strue-def*)

lemma *ST32*:
$(STrue \cap SEmpty) \cdot SEmpty \cap (SInit\ X) = (X \cap SEmpty)$
by (*metis Compl-empty-eq Int-commute CH09 ST14 inf-top.right-neutral sinit-def strue-def*)

lemma *ST34*:
$((X \cap SEmpty) \cdot Y) = (SInit\ X) \cap Y$
by (*metis FusionSEmptyL Int-commute CH09 compl-bot-eq inf-top-right sinit-def strue-def*)

lemma *ST35*:
$((SInit\ X) \cap Y) \cdot Z \subseteq (SInit\ X) \cap (Y \cdot Z)$
by (*metis B04 ST34 FusionAssoc*)

lemma *ST39*:
$SEmpty \cap (SInit\ X) \subseteq (X \cap SEmpty)$
using *ST32* **by** *blast*

lemma *ST40*:
$(X \cap SEmpty) \subseteq SEmpty \cap (SInit\ X)$
using *ST32* **by** *auto*

lemma *ST41*:
$SEmpty \cap (SInit\ X) = (X \cap SEmpty)$
using *ST40 ST39* **by** *auto*

lemma *ST42*:
$(X \cap SEmpty) \subseteq ((X \cup Y) \cap SEmpty)$
by *blast*

lemma *ST43*:
$(Y \cap SEmpty) \subseteq ((X \cup Y) \cap SEmpty)$
by *blast*

lemma *ST44*:
$(X \cap SEmpty) \cap ((-X) \cap SEmpty) = SFalse$
by (*simp add*: *sfalse-def*)

lemma *ST45*:
$((X \cup Y) \cap SEmpty) \subseteq (X \cap SEmpty) \cup (Y \cap SEmpty)$
by *auto*

lemma *ST46*:
$(SInit\ X) \cup (SInit\ Y) = (SInit\ (X \cup Y))$
by (*simp add*: *Int-Un-distrib2 FusionUnionDistL sinit-def*)

lemma *ST48*:
$-(STrue \cdot (X \cap SEmpty)) \subseteq STrue \cdot ((-X) \cap SEmpty)$
by (*metis B09 FusionSEmptyR FusionUnionDistR ST21 double-compl*)

lemma *ST49*:
$STrue \cdot ((-X) \cap SEmpty) \subseteq -(STrue \cdot (X \cap SEmpty))$
by (*metis CH10 Compl-disjoint2 FusionSEmptyR FusionSFalse ST33 disjoint-eq-subset-Compl inf-compl-bot-left2 sfalse-def strue-def*)

lemma *ST50*:
$-(STrue \cdot (X \cap SEmpty)) = STrue \cdot ((-X) \cap SEmpty)$
using *ST48 ST49* **by** *blast*

23.5.5 SStar

lemma *SStar02*:
assumes $X \subseteq Y$
shows $X \cdot (SStar\ Y) \cup SEmpty \subseteq (SStar\ Y)$
using *assms*
by (*metis FusionUnionDistL Int-lower1 SStar15 Un-commute Un-mono UnfoldL inf .orderE sup.orderE sup.orderI*)

lemma *SStar04*:
$(SStar\ X) \subseteq (SStar\ X) \cdot (SStar\ X)$
by (*metis Un-absorb UnfoldL UnionAssoc ST47 sup.absorb-iff2*)

lemma *SStar09*:
assumes $(X \cdot (SEmpty \cup (X \cdot (SStar\ X)))) \cup SEmpty \subseteq (SEmpty \cup (X \cdot (SStar\ X)))$
shows $(SStar\ X) \subseteq SEmpty \cup (X \cdot (SStar\ X))$
using *assms*
by (*simp add*: *UnfoldL*)

lemma *SStar10*:
$(X \cdot (SEmpty \cup (X \cdot (SStar\ X)))) \subseteq (SEmpty \cup (X \cdot (SStar\ X)))$
by (*metis UnfoldL sup-ge2*)

lemma *SStar11*:
$SEmpty \subseteq (SEmpty \cup (X \cdot (SStar\ X)))$
by *auto*

lemma *SStar13*:
$(SStar\ SSkip) = STrue$
by (*simp add*: *SStarSkip*)

lemma *SStar14*:
$(SSometime\ X) = (SStar\ SSkip) \cdot X$
by (*simp add*: *SStarSkip ssometime-def*)

lemma *SStar20*:

$(SStar\ SEmpty) = SEmpty$
by (*metis FusionSEmptyR ST15 ST33*)

lemma *SStar21*:
$(SStar\ (SEmpty \cap X)) \cdot (SEmpty \cap X) = (SEmpty \cap X)$
by (*metis ST15 FusionSEmptyL inf-commute*)

lemma *SStar24*:
$(SStar\ SFalse) = SEmpty$
by (*metis SStar20 SStar47 inf-compl-bot sfalse-def smore-def*)

lemma *SStar26*:
$X \subseteq (SStar\ X)$
by (*metis FusionSEmptyR FusionUnionDistR SStar08 UnCI UnfoldL subsetI subset-iff*)

lemma *SStar27*:
$SEmpty \subseteq (SStar\ X)$
using *UnfoldL* **by** *blast*

lemma *SStar31*:
assumes $X \cup (X \cdot Y) \cdot (X \cdot (SStar\ (Y \cdot X))) \subseteq X \cdot (SStar\ (Y \cdot X))$
shows $(SStar\ (X \cdot Y)) \cdot X \subseteq X \cdot (SStar\ (Y \cdot X))$
using *assms SStarInductL* **by** *blast*

lemma *SStar32*:
$X \cup (X \cdot Y) \cdot (X \cdot (SStar\ (Y \cdot X))) \subseteq X \cdot (SStar\ (Y \cdot X))$
by (*metis B06 SStar10 SStar11 FusionAssoc FusionRuleR FusionSEmptyR UnfoldL*)

lemma *SStar33*:
$(SStar\ (X \cdot Y)) \cdot X \subseteq X \cdot (SStar\ (Y \cdot X))$
using *SStar31 SStar32* **by** *blast*

lemma *SStar37*:
assumes $X \cdot Z \subseteq Z \cdot Y$
shows $(SStar\ X) \cdot Z \subseteq Z \cdot (SStar\ Y)$
proof −
have $Z \cdot SStar\ Y = Z \cdot SEmpty \cup Z \cdot (Y \cdot SStar\ Y)$
by (*metis FusionUnionDistR UnfoldL*)
then have $Z \cdot SStar\ Y \cup (Z \cup X \cdot (Z \cdot SStar\ Y)) = Z \cup Z \cdot Y \cdot SStar\ Y \cup X \cdot Z \cdot SStar\ Y$
using *FusionAssoc FusionSEmptyR* **by** *blast*
then have $Z \cdot SStar\ Y \cup (Z \cup X \cdot (Z \cdot SStar\ Y)) = Z \cdot SStar\ Y$
by (*metis* (*no-types*) *FusionAssoc FusionSEmptyR FusionUnionDistL FusionUnionDistR UnfoldL UnionAssoc assms sup.absorb-iff1*)
then show *?thesis*
by (*meson SStarInductL sup.absorb-iff1*)
qed

lemma *SStar38*:
assumes $Z \cdot X \subseteq Y \cdot Z$
shows $Z \cdot (SStar\ X) \subseteq (SStar\ Y) \cdot Z$

using *assms*
proof −
have *f1*: $Z \cup SStar\ Y \cdot Y \cdot Z = SStar\ Y \cdot Z$
by (*metis* (*no-types*) *SStar30 ST47 UnfoldL*)
have $SStar\ Y \cdot Y \cdot Z = SStar\ Y \cdot Z \cdot X \cup SStar\ Y \cdot Y \cdot Z$
by (*metis FusionAssoc FusionUnionDistR assms subset-Un-eq*)
then have $Z \cup SStar\ Y \cdot Z \cdot X \subseteq SStar\ Y \cdot Z$
using *f1* **by** *blast*
then show *?thesis*
by (*simp add*: *SStarInductR*)
qed

lemma *SStar39*:
$Y \cdot (SStar\ ((SStar\ X) \cdot Y)) \subseteq (SStar\ (Y \cdot (SStar\ X))) \cdot Y$
by (*simp add*: *SStar38 FusionAssoc*)

lemma *SStar40*:
$(SStar\ (Y \cdot (SStar\ X))) \cdot Y \subseteq Y \cdot (SStar\ ((SStar\ X) \cdot Y))$
by (*simp add*: *SStar33*)

lemma *SStar41*:
$Y \cdot (SStar\ ((SStar\ X) \cdot Y)) = (SStar\ (Y \cdot (SStar\ X))) \cdot Y$
using *SStar39 SStar40* **by** *blast*

lemma *SStar42*:
$Z \cdot (SStar\ (Y \cdot Z)) \subseteq (SStar\ (Z \cdot Y)) \cdot Z$
by (*simp add*: *SStar38 FusionAssoc*)

lemma *SStar43*:
$(SStar\ (Z \cdot Y)) \cdot Z \subseteq Z \cdot (SStar\ (Y \cdot Z))$
by (*simp add*: *SStar33*)

lemma *SStar44*:
$Z \cdot (SStar\ (Y \cdot Z)) = (SStar\ (Z \cdot Y)) \cdot Z$
using *SStar42 SStar43* **by** *blast*

lemma *SStar49*:
$(SStar\ X) = SEmpty \cup (SStar\ X) \cdot X$
using *SStar30 UnfoldL* **by** *blast*

23.5.6 Box and Diamond

lemma *BD01*:
$(SSometime\ SEmpty) = STrue$
by (*simp add*: *ssometime-def FusionSEmptyR*)

lemma *BD02*:
$X \subseteq (SSometime\ X)$
by (*metis FusionUnionDistL SEmptyChop STrueTop Subsumption Un-absorb semigroup.assoc*

ssometime-def sup.semigroup-axioms)

lemma *BD03*:
$(SNext\ (SSometime\ X)) \subseteq (SSometime\ X)$
by (*metis FusionUnionDistL N03 SStar16 SStar03 SStar04 SStarSkip snext-def ssometime-def sup.absorb-iff2 sup.orderE*)

lemma *BD04*:
$(SSometime\ (SNext\ X)) \subseteq (SSometime\ X)$
by (*metis CH01 FusionAssoc FusionUnionDistL FusionUnionDistR SStar16 SStarSkip snext-def ssometime-def sup.absorb-iff2*)

lemma *BD05*:
$(SSometime\ X) \cup (SSometime\ Y) = (SSometime\ (X \cup Y))$
by (*simp add*: *FusionUnionDistR ssometime-def*)

lemma *BD06*:
$(SSometime\ STrue) = STrue$
by (*simp add*: *CH01 ssometime-def*)

lemma *BD07*:
$(SSometime\ (X \cap Y)) \subseteq (SSometime\ X) \cap (SSometime\ Y)$
by (*simp add*: *FusionRuleR ssometime-def*)

lemma *BD08*:
$(SAlways\ STrue) = STrue$
by (*simp add*: *SBoxGen*)

lemma *BD09*:
$-(SAlways\ X) = (SSometime\ (-X))$
by (*simp add*: *salways-def*)

lemma *BD10*:
$(SAlways\ X) \subseteq (SSometime\ X)$
by (*metis B02 BD02 BD09 set-rev-mp subsetI*)

lemma *BD11*:
$(SSometime\ (SSometime\ X)) = (SSometime\ X)$
by (*simp add*: *CH01 ssometime-def FusionAssoc*)

lemma *BD12*:
$(SAlways\ X) \subseteq X$
by (*simp add*: *B02 BD02 BD09*)

lemma *BD13*:
$(SDi\ STrue) = STrue$
by (*simp add*: *CH01 sdi-def*)

lemma *BD14*:
$(SDi\ SEmpty) = STrue$

by (*simp add*: *sdi-def FusionSEmptyL*)

lemma *BD15*:
 (*SBi STrue*) = *STrue*
by (*simp add*: *SBiGen*)

lemma *BD16*:
 (*SDi* ($X \cup Y$)) = (*SDi* X) $\cup$ (*SDi* Y)
by (*simp add*: *FusionUnionDistL sdi-def*)

lemma *BD17*:
 assumes $X \subseteq Y$
 shows (*SDi* X) $\subseteq$ (*SDi* Y)
using *assms*
by (*metis FusionUnionDistL Subsumption sdi-def*)

lemma *BD18*:
 (*SDi* (*SDi* X)) = (*SDi* X)
by (*metis CH01 FusionAssoc sdi-def*)

lemma *BD19*:
 (*SDa SEmpty*) = *STrue*
by (*simp add*: *CH01 sda-def FusionSEmptyR*)

lemma *BD20*:
 (*SDa STrue*) = *STrue*
by (*simp add*: *CH01 sda-def*)

lemma *BD21*:
 (*SBa STrue*) = *STrue*
by (*metis BD15 BD08 BD09 sba-def sbi-def sda-def sdi-def ssometime-def*)

lemma *BD22*:
 (*SDa* ($X \cup Y$)) = (*SDa* X) $\cup$ (*SDa* Y)
by (*simp add*: *FusionUnionDistL FusionUnionDistR sda-def*)

lemma *BD23*:
 assumes $X \subseteq Y$
 shows (*SDa* X) $\subseteq$ (*SDa* Y)
using *assms*
by (*metis BD22 Subsumption*)

lemma *BD24*:
 assumes $X \subseteq Y$
 shows (*SDa* ($-Y$)) $\subseteq$ (*SDa* ($-X$))
using *assms*
by (*simp add*: *BD23*)

lemma *BD25*:
 (*SDi* X) $\subseteq$ (*SDa* X)

by (*metis BD02 FusionAssoc sda-def sdi-def ssometime-def*)

lemma *BD26*:
$(SSometime\ X) \subseteq (SDa\ X)$
by (*metis BD01 BD02 FusionSEmptyR FusionUnionDistR SStar14 le-iff-sup sda-def*)

lemma *BD27*:
$(SBa\ X) \subseteq (SBi\ X)$
by (*simp add*: *BD25 sba-def sbi-def*)

lemma *BD28*:
$(SBa\ X) \subseteq (SAlways\ X)$
by (*simp add*: *B02 BD26 BD09 sba-def*)

lemma *BD29*:
$(SAlways\ X) \cap (SAlways\ Y) = (SAlways\ (X \cap Y))$
by (*metis BD05 BD09 Morgan compl-inf salways-def*)

lemma *BD30*:
$(SAlways\ X) \cup (SAlways\ Y) \subseteq (SAlways\ (X \cup Y))$
using *BD07*
by (*metis B02 BD09 compl-sup*)

lemma *BD31*:
$(SDi\ (X \cap Y)) \subseteq (SDi\ X) \cap (SDi\ Y)$
by (*simp add*: *BD17*)

lemma *BD32*:
$(SBi\ X) \cup (SBi\ Y) \subseteq (SBi\ (X \cup Y))$
using *BD31*
by (*metis* (*mono-tags*, *lifting*) *B02 compl-sup double-compl sbi-def*)

lemma *BD33*:
$(SDa\ (X \cap Y)) \subseteq (SDa\ X) \cap (SDa\ Y)$
by (*simp add*: *BD23*)

lemma *BD34*:
$(SBa\ X) \cup (SBa\ Y) \subseteq (SBa\ (X \cup Y))$
using *BD33*
by (*metis* (*mono-tags*, *lifting*) *B02 compl-sup double-compl sba-def*)

lemma *BD35*:
$(SAlways\ SEmpty) = SEmpty$
by (*metis N13 SStar14 SStar30 SStar48 SStarSkip double-complement salways-def smore-def*)

lemma *BD36*:
$(SBi\ SEmpty) = SEmpty$
using *N13 sbi-def sdi-def smore-def* **by** *fastforce*

lemma *BD37*:

$(SBa\ SEmpty) = SEmpty$
by (*metis N13 SStar30 SStar48 double-complement sba-def sda-def smore-def*)

lemma *BD38*:
assumes $X \subseteq Y$
shows $(SAlways\ X) \subseteq (SAlways\ Y)$
using *assms*
by (*simp add*: *BD29 inf.absorb-iff2*)

lemma *BD39*:
assumes $X \subseteq Y$
shows $(SBi\ X) \subseteq (SBi\ Y)$
using *assms*
by (*simp add*: *BD17 sbi-def*)

lemma *BD40*:
assumes $X \subseteq Y$
shows $(SBa\ X) \subseteq (SBa\ Y)$
using *assms*
by (*simp add*: *BD24 sba-def*)

lemma *BD41*:
$(SBi\ (SBi\ X)) = (SBi\ X)$
by (*simp add*: *BD18 sbi-def*)

lemma *BD42*:
$(SAlways\ (SAlways\ X)) = (SAlways\ X)$
by (*simp add*: *BD11 salways-def*)

lemma *BD43*:
$(SDa\ (SDa\ X)) = (SDa\ X)$
by (*metis CH01 FusionAssoc sda-def*)

lemma *BD44*:
$(SBa\ (SBa\ X)) = (SBa\ X)$
by (*simp add*: *BD43 sba-def*)

lemma *BD47*:
$(SAlways\ (\ (-X) \cup Y)) \subseteq (\ -(SAlways\ X) \cup (SAlways\ Y))$
by (*metis B20 BD12 BD29 BD38 BD42 double-compl*)

lemma *BD48*:
$(SAlways\ X) \subseteq X \cap (SWnext\ (SAlways\ X))$
by (*metis B02 B16 BD03 BD09 BD12 N12 salways-def*)

lemma *BD49*:
$(SBi\ (\ (-X) \cup Y)) \subseteq (-(SBi\ X) \cup (SBi\ Y)\ )$
by (*metis B20 BD45 Un-commute double-complement sbi-def sdi-def*)

lemma *BD50*:

$(SPrev\ (SDi\ X)) \subseteq (SDi\ X)$
by (*metis B04 FusionAssoc FusionUnionDistR N08 SSkipFusionImpSMore SStar19 SStarSkip STrueTop sdi-def smore-def sprev-def sup-ge2*)

lemma *BD51*:
$-(SBi\ X) = (SDi\ (-X))$
by (*simp add*: *sbi-def*)

lemma *BD52*:
$X \subseteq (SDi\ X)$
by (*metis FusionSEmptyR FusionUnionDistR ST33 Subsumption UnionCommute sdi-def sup-inf-absorb*)

lemma *BD53*:
$(SBi\ X) \subseteq X$
by (*simp add*: *B02 BD51 BD52*)

lemma *BD54*:
$(SBi\ X) \subseteq X \cap (SWprev\ (SBi\ X))$
by (*metis B02 B16 BD50 BD51 BD53 N29 sbi-def*)

lemma *BD55*:
$(SBi\ (SMore \cup X)) = (SInit\ X)$
by (*metis* (*no-types*, *lifting*) *ST38 compl-sup double-complement inf-commute sbi-def sdi-def sinit-def smore-def*)

lemma *BD56*:
$(SAlways\ (SMore \cup X)) = STrue \cdot (X \cap SEmpty)$
by (*simp add*: *SStar14 SStarSkip ST50 UnionCommute salways-def smore-def*)

23.6 Time Reversal

23.6.1 Time Reversal Axioms

lemma *SRevSEmpty*:
$(SRev\ SEmpty) = SEmpty$
using *set-eqI*[*of* (*SRev SEmpty*) *SEmpty*]
by (*simp add*: *sempty-elim srev-elim*)

lemma *SRevSNot*:
$(SRev\ (-\ X)) = (-\ (SRev\ X))$
using *set-eqI*[*of* $(SRev\ (-\ X))\ (-\ (SRev\ X))$]
by (*simp add*: *srev-elim*)

lemma *SRevFusion*:
$(SRev\ (X \cdot Y)) = (SRev\ Y) \cdot (SRev\ X)$
using *set-eqI*[*of* $(SRev\ (X \cdot Y))\ \ (SRev\ Y) \cdot (SRev\ X)$]
by (*simp add*: *fusion-iff-1 srev-elim*)
(*metis diff-diff-cancel interval-intrev-prefix interval-intrev-suffix interval-suffix-intlen-bound interval-suffix-length*)

lemma *SRevUnion*:
(*SRev* (*X* ∪ *Y*)) = (*SRev X*) ∪ (*SRev Y*)
using *set-eqI*[*of* (*SRev* (*X* ∪ *Y*)) (*SRev X*) ∪ (*SRev Y*)]
using *srev-elim* **by** *auto*

lemma *SRevSPower*:
(*SRev* (*SPower X n*)) = (*SPower* (*SRev X*) *n*)
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *SRevSEmpty*)
next
case (*Suc n*)
then show *?case*
proof −
have *SRev X* ∩ *SMore* = *SRev* (*X* ∩ *SMore*)
by (*metis* (*no-types*) *Morgan SRevSEmpty SRevSNot SRevUnion smore-def*)
then show *?thesis*
by (*simp add*: *SRevFusion Suc.hyps spower-commutes*)
qed
qed

lemma *SRevSStar*:
(*SRev* (*SStar X*)) = (*SStar* (*SRev X*))
proof −
have *1*: (*SRev* (*SStar X*)) = (*SRev* (⋃ *n*. *SPower X n*)) **by** (*simp add*: *sstar-def*)
have *2*: (*SRev* (⋃ *n*. *SPower X n*)) = (⋃ *n*. *SPower* (*SRev X*) *n*)
using *set-eqI*[*of* (*SRev* (⋃ *n*. *SPower X n*)) (⋃ *n*. *SPower* (*SRev X*) *n*)]
by (*metis* (*mono-tags*, *lifting*) *SRevSPower UN-iff srev-elim*)
have *3*: (⋃ *n*. *SPower* (*SRev X*) *n*) = (*SStar* (*SRev X*)) **by** (*simp add*: *sstar-def*)
from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *SRevSRev*:
(*SRev* (*SRev X*)) = *X*
using *set-eqI*[*of* (*SRev* (*SRev X*)) *X*]
by (*simp add*: *srev-elim*)

23.6.2 Time Reversal Laws

lemma *TR01*:
(*SRev SMore*) = *SMore*
by (*simp add*: *SRevSEmpty SRevSNot smore-def*)

lemma *TR02*:
(*SRev SSkip*) = *SSkip*
by (*metis SRevFusion SRevSEmpty SRevSNot SRevUnion TR01 sskip-def*)

lemma *TR03*:
(*SRev STrue*) = *STrue*
by (*metis SRevSStar SStarSkip TR02*)

lemma *TR04*:
(*SRev SFalse*) = *SFalse*
by (*metis Compl-eq-Compl-iff SRevSNot TR03 sfalse-def strue-def*)

lemma *TR05*:
(*SRev* (*SSometime X*)) = (*SDi* (*SRev X*))
by (*simp add*: *SRevFusion TR03 sdi-def ssometime-def*)

lemma *TR06*:
(*SRev* (*SAlways X*)) = (*SBi* (*SRev X*))
by (*simp add*: *SRevSNot TR05 salways-def sbi-def*)

lemma *TR07*:
(*SRev* (*SDi X*)) = (*SSometime* (*SRev X*))
by (*simp add*: *SRevFusion TR03 sdi-def ssometime-def*)

lemma *TR08*:
(*SRev* (*SBi X*)) = (*SAlways* (*SRev X*))
by (*metis SRevSRev TR06*)

lemma *TR09*:
(*SRev* (*SNext X*)) = (*SPrev* (*SRev X*))
by (*simp add*: *SRevFusion TR02 snext-def sprev-def*)

lemma *TR10*:
(*SRev* (*SWnext X*)) = (*SWprev* (*SRev X*))
by (*simp add*: *SRevFusion SRevSNot TR02 swnext-def swprev-def*)

lemma *TR11*:
(*SRev* (*SPrev X*)) = (*SNext* (*SRev X*))
by (*simp add*: *SRevFusion TR02 snext-def sprev-def*)

lemma *TR12*:
(*SRev* (*SWprev X*)) = (*SWnext* (*SRev X*))
by (*metis SRevSRev TR10*)

lemma *TR13*:
(*SRev* (*SDa X*)) = (*SDa* (*SRev X*))
by (*simp add*: *SRevFusion TR03 sda-def FusionAssoc*)

lemma *TR14*:
(*SRev* (*SBa X*)) = (*SBa* (*SRev X*))
by (*simp add*: *SRevSNot TR13 sba-def*)

lemma *TR15*:
(*SRev* (*SPower SSkip n*)) = (*SPower SSkip n*)
by (*simp add*: *SRevSPower TR02*)

lemma *TR16*:

assumes $X \subseteq Y$
shows $(SRev\ X) \subseteq (SRev\ Y)$
using *assms* **by** (*metis SRevUnion le-iff-sup*)

lemma *TR17*:
assumes $X = Y$
shows $(SRev\ X) = (SRev\ Y)$
using *assms TR16* **by** *auto*

23.7 Link between Set of Intervals and ITL

lemma *interval-lan* [*simp*]:
$\sigma \in (lan\ f) \longleftrightarrow (\sigma \models f)$
by (*simp add*: *lan-def*)

lemma *valid-lan-eqv* :
$(\ (lan\ f) = (lan\ g)\ ) \longleftrightarrow (\ \vdash f = g\ )$
using *interval-lan lan-def Valid-def* **by** *fastforce*

lemma *valid-lan-imp* :
$(\ (lan\ f) \subseteq (lan\ g)\ ) \longleftrightarrow (\vdash f \longrightarrow g)$
using *interval-lan lan-def Valid-def*
by (*simp add*: *Valid-def lan-def Collect-mono-iff*)

lemma *valid-strue* :
$(\ (lan\ f) = STrue) \longleftrightarrow (\vdash f)$
using *strue-def* **by** *fastforce*

lemma *strue-true*:
$\sigma \in STrue \longleftrightarrow (\sigma \models \#True)$
by (*simp add*: *strue-elim*)

lemma *strue-true-1*:
$STrue = (lan\ (LIFT\ \#True))$
using *lan-def strue-true* **by** *fastforce*

lemma *sfalse-false*:
$\sigma \in SFalse \longleftrightarrow (\sigma \models \#False)$
by (*simp add*: *sfalse-def*)

lemma *sfalse-false-1*:
$SFalse = (lan\ (LIFT(\#False)))$
using *sfalse-false* **using** *lan-def* **by** *fastforce*

lemma *not-negation*:
$\sigma \in (-(lan\ f)) \longleftrightarrow (\sigma \models \neg\ f)$
by *simp*

lemma *not-negation-1*:
$-(lan\ f\ ) = (lan\ (LIFT(\neg\ f)))$

using *interval-lan lan-def* **by** *fastforce*

lemma *inter-and*:
$(\sigma \in ((lan\ f) \cap (lan\ g))) \longleftrightarrow (\sigma \models f \wedge g)$
by (*simp add*: *lan-def*)

lemma *inter-and-1*:
$((lan\ f) \cap (lan\ g)) = (lan\ (LIFT(f \wedge g)))$
using *inter-and lan-def* **by** *fastforce*

lemma *union-or*:
$(\sigma \in ((lan\ f) \cup (lan\ g))) \longleftrightarrow (\sigma \models f \vee g)$
by (*simp add*: *lan-def*)

lemma *union-or-1*:
$((lan\ f) \cup (lan\ g)) = (lan\ (LIFT(f \vee g)))$
using *union-or lan-def* **by** *fastforce*

lemma *subset-impl*:
$(\sigma \in (-(lan\ f) \cup (lan\ g))) \longleftrightarrow (\sigma \models f \longrightarrow g)$
by *simp*

lemma *subset-impl-1*:
$(-(lan\ f) \cup (lan\ g)) = (lan\ (LIFT(f \longrightarrow g)))$
using *subset-impl lan-def* **by** *fastforce*

lemma *fusion-chop*:
$(\sigma \in ((lan\ f)\cdot(lan\ g))) \longleftrightarrow (\sigma \models f;g)$
by (*metis fusion-iff interval-chop-fuse interval-lan*)

lemma *fusion-chop-1*:
$((lan\ f)\cdot(lan\ g)) = (lan\ (LIFT(f;g)))$
using *fusion-chop lan-def* **by** *blast*

lemma *sempty-empty*:
$\sigma \in SEmpty \longleftrightarrow (\sigma \models empty)$
by (*simp add*: *empty-defs sempty-elim*)

lemma *sempty-empty-1*:
$SEmpty = (lan\ (LIFT\ empty))$
using *sempty-empty lan-def* **by** *fastforce*

lemma *smore-more*:
$\sigma \in SMore \longleftrightarrow (\sigma \models more)$
by (*simp add*: *more-defs smore-elim*)

lemma *smore-more-1*:
$SMore = (lan\ (LIFT\ more))$
using *smore-more lan-def* **by** *fastforce*

lemma *sskip-skip*:
$\sigma \in SSkip = (\sigma \models skip)$
by (*simp add*: *skip-defs sskip-elim*)

lemma *sskip-skip-1*:
$SSkip = (lan\ (LIFT\ skip))$
using *sskip-skip lan-def* **by** *fastforce*

lemma *snext-next*:
$\sigma \in (SNext\ (lan\ f)) \longleftrightarrow (\sigma \models \bigcirc\ f)$
by (*metis snext-def fusion-chop next-d-def sskip-skip-1*)

lemma *snext-next-1*:
$(SNext\ (lan\ f)) = (lan\ (LIFT(\bigcirc\ f)))$
using *snext-next lan-def* **by** *fastforce*

lemma *swnext-wnext*:
$\sigma \in (SWnext\ (lan\ f)) \longleftrightarrow (\sigma \models wnext\ f)$
by (*simp add*: *swnext-def fusion-chop-1 next-d-def not-negation-1 sskip-skip-1 wnext-d-def*)

lemma *swnext-wnext-1*:
$(SWnext\ (lan\ f)) = (lan\ (LIFT(wnext\ f)))$
using *swnext-wnext lan-def* **by** *fastforce*

lemma *sprev-prev*:
$\sigma \in (SPrev\ (lan\ f)) \longleftrightarrow (\sigma \models prev\ f)$
by (*metis fusion-chop prev-d-def sprev-def sskip-skip-1*)

lemma *sprev-prev-1*:
$(SPrev\ (lan\ f)) = (lan\ (LIFT(prev\ f)))$
using *sprev-prev lan-def* **by** *fastforce*

lemma *swprev-wprev*:
$\sigma \in (SWprev\ (lan\ f)) \longleftrightarrow (\sigma \models wprev\ f)$
by (*simp add*: *fusion-chop-1 not-negation-1 prev-d-def sskip-skip-1 swprev-def wprev-d-def*)

lemma *swprev-wprev-1*:
$(SWprev\ (lan\ f)) = (lan\ (LIFT(wprev\ f)))$
using *swprev-wprev lan-def* **by** *fastforce*

lemma *sinit-init*:
$\sigma \in SInit\ (lan\ f) \longleftrightarrow (\sigma \models init\ f)$
by (*simp add*: *Int-commute fusion-chop-1 init-d-def inter-and-1 sempty-empty-1 sinit-def strue-true-1*)

lemma *sinit-init-1*:
$SInit\ (lan\ f) = (lan\ (LIFT(init\ f)))$
using *sinit-init lan-def* **by** *fastforce*

lemma *and-inter-more*:
$\sigma \in (((lan\ f) \cap SMore)) \longleftrightarrow (\sigma \models (f\ \wedge\ more)\ )$

using *smore-more inter-and* **by** *auto*

lemma *and-inter-more-1*:
$\sigma \in (((lan\ f) \cap SMore)) \longleftrightarrow (\sigma \in (lan\ (LIFT(f \wedge more))\ ))$
using *and-inter-more lan-def* **by** (*simp add*: *smore-more-1*)

lemma *and-inter-more-2*:
$((lan\ f) \cap SMore) = (lan\ (LIFT(f \wedge more))\ )$
using *and-inter-more-1* **by** *blast*

lemma *and-chop*:
$\sigma \in (((lan\ f) \cap SMore)\cdot\ (lan\ g)) \longleftrightarrow (\sigma \models (f \wedge more);g)$
by (*metis fusion-chop inter-and-1 smore-more-1*)

lemma *and-chop-1*:
$(((lan\ f) \cap SMore)\cdot\ (lan\ g)) = (lan\ (LIFT((f \wedge more);g)))$
using *and-chop lan-def* **by** *blast*

lemma *spower-chop-power*:
$(SPower\ (lan\ f)\ n) = (lan\ (LIFT(power\ (f \wedge more)\ n)))$
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *sempty-empty-1*)
next
case (*Suc n*)
then show *?case* **by** (*metis and-chop-1 pow-Suc pwr-Suc*)
qed

lemma *sstar-spower*:
$\sigma \in SStar\ (lan\ f) \longleftrightarrow (\exists\ n.\ \sigma \in SPower\ (lan\ f)\ n)$
by (*simp add*: *sstar-def*)

lemma *sstar-chopstar*:
$\sigma \in (SStar\ (lan\ f)) \longleftrightarrow \sigma \in (lan\ (LIFT(f^\star)))$
proof −
have *1*: $\sigma \in (SStar\ (lan\ f)) = (\exists\, n.\ \sigma \in SPower\ (lan\ f)\ n)$
using *sstar-spower* **by** *blast*
have *2*: $(\exists\, n.\ \sigma \in SPower\ (lan\ f)\ n) =$
$(\exists\, n.\ \sigma \in lan\ (LIFT(power\ (f \wedge more)\ n)))$
using *spower-chop-power* **by** *blast*
have *3*: $(\exists\, n.\ \sigma \in lan\ (LIFT(power\ (f \wedge more)\ n))) =$
$(\exists\, n.\ \ (LIFT(power\ (f \wedge more)\ n))\ \sigma\ \ )$
using *interval-lan* **by** *simp*
have *4*: $(\exists\, n.\ \ (LIFT(power\ (f \wedge more)\ n))\ \sigma\ \ ) =$
$(\sigma \in (lan\ (LIFT(f^\star))))$
by (*simp add*: *chopstar-d-def powerstar-d-def*)
show *?thesis* **by** (*simp add*: *1 2 4*)
qed

lemma *chopstar-sstar-1*:

$(SStar\ (lan\ f)) = (lan\ (LIFT(f^{\star})))$
using *sstar-chopstar lan-def* **by** *blast*

lemma *chopstar-seqv*:
$\sigma \in (lan\ (LIFT(f^{\star}))) \longleftrightarrow$
$\sigma \in (lan\ (LIFT(empty \vee\ (f \wedge more);\ f^{\star})))$
by (*metis Un-iff and-chop-1 chopstar-sstar-1 sempty-empty-1 sstar-eqv union-or-1*)

lemma *chopstar-seqv-1*:
$(lan\ (LIFT(f^{\star}))) = (lan\ (LIFT(empty \vee\ (f \wedge more);\ f^{\star})))$
using *chopstar-seqv lan-def* **by** *blast*

lemma *srev-rev*:
$\sigma \in (SRev\ (lan\ f)) \longleftrightarrow \sigma \in (lan\ (LIFT(f^{r})))$
by (*simp add*: *reverse-d-def srev-elim*)

lemma *srev-rev-1*:
$(SRev\ (lan\ f)) = (lan\ (LIFT(f^{r})))$
using *srev-rev lan-def* **by** *blast*

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. Compositional reasoning about projected and infinite time. In *Proceedings of the First IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS'95)*, pages 238–245. IEEE Computer Society Press, 1995. Download pdf.

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

[6] B. Moszkowski. Compositional reasoning using intervals and time reversal. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):175–250, 2014.

[7] B. Moszkowski and D. Guelev. An application of temporal projection to interleaving concurrency. *Formal Aspects of Computing*, 29(4):705–750, July 2017.

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

[9] B. C. Moszkowski. A Complete Axiom System for Propositional Interval Temporal Logic with Infinite Time. *Logical Methods in Computer Science Journal*, 8(3), 2012.

[10] B. C. Moszkowski and D. P. Guelev. An Application of Temporal Projection to Interleaving Concurrency. In *Dependable Software Engineering: Theories, Tools, and Applications - First International Symposium, SETTA 2015, Nanjing, China, November 4-6, 2015, Proceedings*, pages 153–167, 2015.

[11] J. S. Warford, D. Vega, and S. M. Staley. A Calculational Deductive System for Linear Temporal Logic. https://www.cslab.pepperdine.edu/warford/Papers/Vega-Paper.pdf, 2019.

[12] M. Wildmoser and T. Nipkow. Certifying Machine Code Safety: Shallow versus Deep Embedding. In K. Slind, A. Bunker, and G. Gopalakrishnan, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2004)*, volume 3223 of *LNCS*, pages 305–320. Springer, 2004.