An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau Ben Moszkowski David Smallwood

October 26, 2022

Abstract

These Isabelle theories introduce the semantics and syntax of Finite and Infinite Interval Temporal Logic (ITL). The ITL proof system, as introduced in [6, 9], has been encoded and its soundness has been checked. The encoding is shallow using the Intensional Logic technique of [4]. An extensive library of Finite and Infinite ITL theorems, taken from [8], has been checked. Non-empty coinductive lists are used to denote intervals.

Contents

1 Extra operations on LLists

the operations kfilter, lleast, lbutlast, lidx, ridx, lfirst, lfuse, lsub, lsubc, llastlfirst, lltl, llbutlast, is_lfirst and lfusecat on coinductive lists are defined together with a library of lemmas.

theory *LList-Extras*

imports
Coinductive.Coinductive-List
begin

definition *kfilter* :: $('a \Rightarrow bool) \Rightarrow nat \Rightarrow 'a\ llist \Rightarrow nat\ llist$
where $kfilter\ P\ n\ xs = lmap\ snd\ (lfilter\ (P \circ fst)\ (lzip\ xs\ (iterates\ Suc\ n)))$

definition *lleast* :: $('a \Rightarrow bool) \Rightarrow 'a\ llist \Rightarrow nat$
where $lleast\ P\ xs = (LEAST\ na.\ na < llength\ xs \wedge P\ (lnth\ xs\ na))$

primcorec *lbutlast* :: $'a\ llist \Rightarrow 'a\ llist$
where $lbutlast\ xs =$
$(case\ xs\ of\ LNil \Rightarrow LNil\ |$
$(LCons\ x\ xs') \Rightarrow$
$(case\ xs'\ of\ LNil \Rightarrow LNil\ |$
$(LCons\ x1\ xs1) \Rightarrow (LCons\ x\ (lbutlast\ xs'))))$

simps-of-case *lbutlast-simps* [*simp*, *code*, *nitpick-simp*]: *lbutlast.code*

definition *ridx* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ llist \Rightarrow bool$
where $ridx\ R\ xs = (\forall\ i.\ (Suc\ i) < llength\ xs \longrightarrow R\ (lnth\ xs\ i)\ (lnth\ xs\ (Suc\ i)))$

definition *lidx* :: $nat\ llist \Rightarrow bool$
where $lidx\ xs \longleftrightarrow (\forall n.\ (Suc\ n) < llength\ xs \longrightarrow lnth\ xs\ n < lnth\ xs\ (Suc\ n))$

definition *lfirst* :: $'a\ llist \Rightarrow 'a$
where $lfirst\ xs = lhd\ xs$

definition *lfuse* :: $'a\ llist \Rightarrow 'a\ llist \Rightarrow 'a\ llist$
where $lfuse\ xs\ ys =$
$(if\ \neg\ lnull\ xs \wedge \neg\ lnull\ ys\ then\ lappend\ xs\ (ltl\ ys)\ else\ lappend\ xs\ ys)$

definition *lsub* :: $nat \Rightarrow nat \Rightarrow 'a\ llist \Rightarrow 'a\ llist$
where $lsub\ k\ n\ xs = (ltake\ (eSuc\ (n - k))\ (ldrop\ k\ xs))$

definition *lsubc* :: $nat \Rightarrow nat \Rightarrow 'a\ llist \Rightarrow 'a\ llist$
where $lsubc\ k\ n\ xs = (ltake\ (eSuc\ (n - k))\ (ldrop\ ((min\ k\ (epred\ (llength\ xs))))\ xs))$

definition *llastlfirst* :: $'a\ llist\ llist \Rightarrow bool$
where $llastlfirst\ xss = (\forall i.\ (Suc\ i) < llength\ xss \longrightarrow llast\ (lnth\ xss\ i) = lfirst(lnth\ xss\ (Suc\ i)))$

definition *lltl* :: *'a llist llist* ⇒ *'a llist llist*
where *lltl xss* = *lmap ltl xss*

definition *llbutlast* :: *'a llist llist* ⇒ *'a llist llist*
where *llbutlast xss* = *lmap lbutlast xss*

abbreviation *is-lfirst* ≡ (*λxs*. ∃ *b*. *xs* = (*LCons b LNil*))

1.1 Auxiliary lemmas

lemma *enat-min*:
  assumes $m \geq enat\ n'$
    and $enat\ n < m - enat\ n'$
  shows $enat\ n + enat\ n' < m$
using *assms*
by (*metis add.commute enat.simps*(*3*) *enat-add-mono enat-add-sub-same le-iff-add*)

lemma *enat-min-eq*:
  assumes $m \geq enat\ n'$
    and $enat\ n \leq m - enat\ n'$
  shows $enat\ n + enat\ n' \leq m$
using *assms*
by (*metis add.commute enat.simps*(*3*) *enat-add-sub-same enat-min le-iff-add le-less*)

lemma *llist-eq-lnth-eq*:
  (*xs* = *ys*) ⟷ (*llength xs* = *llength ys* ∧ (∀ *i* < *llength xs*. *lnth xs i* = *lnth ys i*))
proof *auto*
 show *llength xs* = *llength ys* ⟹ ∀ *i*. *enat i* < *llength ys* ⟶ *lnth xs i* = *lnth ys i* ⟹ *xs* = *ys*
 proof (*coinduction arbitrary*: *xs ys*)
 case (*Eq-llist xsa ysa*)
 then show *?case*
    proof −
     assume *a*: *llength xsa* = *llength ysa*
     assume *b*: ∀ *i*. *enat i* < *llength ysa* ⟶ *lnth xsa i* = *lnth ysa i*
     have *1*: *lnull xsa* = *lnull ysa*
       using *a* **by** *auto*
     have *2*: ¬ *lnull xsa* ⟶
            ¬ *lnull ysa* ⟶
            lhd xsa = *lhd ysa*
       using *b lhd-conv-lnth zero-enat-def* **by** *force*
     have *3*: ¬ *lnull xsa* ⟶
            ¬ *lnull ysa* ⟶
              (∃ *xs ys*. *ltl xsa* = *xs* ∧ *ltl ysa* = *ys* ∧ *llength xs* = *llength ys* ∧
                   (∀ *i*. *enat i* < *llength ys* ⟶ *lnth xs i* = *lnth ys i*))
       by (*metis Extended-Nat.eSuc-mono a b eSuc-enat eSuc-epred llength-eq-0 llength-ltl lnth-ltl*)
    show *?thesis* **using** *1 2 3* **by** *blast*
   qed
 qed
qed

lemma *exist-lset-lnth*:
  (∃ *x* ∈ *lset xs*. *P x*) ⟷ (∃ *i*<*llength xs*. *P* (*lnth xs i*))
by (*metis in-lset-conv-lnth*)

lemma *exist-llength-gr-zero*:
 assumes (∃ *x* ∈ *lset xs*. *P x*)
 shows *0*< *llength xs*
 by (*metis assms gr-implies-not-zero gr-zeroI in-lset-conv-lnth*)

1.2 lbutlast

lemma *lbutlast-snoc* [*simp*]:
  lbutlast (*lappend xs* (*LCons x LNil*)) = *xs*
proof (*cases lfinite xs*)
case *True*
then show *?thesis*
  proof (*induct rule*: *lfinite-induct*)
  case (*LNil xs*)
  then show *?case* **using** *llist.collapse*(*1*) **by** *fastforce*
  next
  case (*LCons xs*)
  then show *?case*
    proof (*cases xs*=*LNil*)
    case *True*
    then show *?thesis* **by** *simp*
    next
    case *False*
    then show *?thesis*
     proof −
      have *1*: ∃ *y ys*. *xs* =(*LCons y ys*)
        using *False llist.exhaust-sel* **by** *blast*
      obtain *y ys* **where** *2*: *xs* =(*LCons y ys*)
         using *1* **by** *blast*
      have *3*: *lbutlast* (*lappend xs* (*LCons x LNil*)) = *lbutlast* (*lappend* (*LCons y ys*) (*LCons x LNil*))
         by (*simp add*: *2*)
       have *4*: *lbutlast* (*lappend* (*LCons y ys*) (*LCons x LNil*)) = *lbutlast* (*LCons y* (*lappend ys* (*LCons x LNil*)))
        by *simp*
      have *5*: *lnull ys* ⟹ *?thesis*
        by (*simp add*: *2 lnull-def*)
      have *6*: ¬*lnull ys* ⟹ *?thesis*
         by (*metis 2 LCons.hyps*(*3*) *eq-LConsD lappend-code*(*2*) *lbutlast-simps*(*3*) *lhd-LCons-ltl*)
      show *?thesis*
      using *5 6* **by** *blast*
     qed
    qed
  qed
next
case *False*

```
then show ?thesis
 proof (coinduction arbitrary: xs)
 case (Eq-llist xsa)
 then show ?case
     proof −
      have 1: lnull (lbutlast (lappend xsa (LCons x LNil))) = lnull xsa
        by (metis Eq-llist lappend-inf lbutlast.disc(2) lfinite.simps lhd-LCons-ltl lnull-imp-lfinite)
      have 2: ¬ lnull (lbutlast (lappend xsa (LCons x LNil))) ⟶
               ¬ lnull xsa ⟶
               lhd (lbutlast (lappend xsa (LCons x LNil))) = lhd xsa
           by (metis Eq-llist eq-LConsD lappend-inf lbutlast.disc(1) lbutlast-simps(3) not-lnull-conv)
      have 3: ¬ lnull (lbutlast (lappend xsa (LCons x LNil))) ⟶
               ¬ lnull xsa ⟶
              (∃ xs. ltl (lbutlast (lappend xsa (LCons x LNil))) =
                          lbutlast (lappend xs (LCons x LNil)) ∧ ltl xsa = xs ∧ ¬ lfinite xs)
        by (metis Eq-llist eq-LConsD lappend-inf lbutlast-simps(3) lfinite-ltl lhd-LCons-ltl lnull-imp-lfinite)
      show ?thesis using 1 2 3 by auto
    qed
 qed
qed


lemma lbutlast-ltl:
  lbutlast (ltl xs) = ltl (lbutlast xs)
proof (cases lfinite xs)
case True
then show ?thesis
  proof  (induct rule: lfinite-induct)
  case (LNil xs)
  then show ?case by (simp add: lbutlast.ctr(1))
  next
  case (LCons xs)
  then show ?case
     by (simp add: lbutlast.code llist.case-eq-if)
  qed
next
case False
then show ?thesis
   proof  (coinduction arbitrary: xs)
   case (Eq-llist xsa)
   then show ?case
     proof −
      have 1: lnull (lbutlast (ltl xsa)) = lnull (ltl (lbutlast xsa))
        by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
      have 2: ¬ lnull (lbutlast (ltl xsa)) ⟶
               ¬ lnull (ltl (lbutlast xsa)) ⟶
               lhd (lbutlast (ltl xsa)) = lhd (ltl (lbutlast xsa))
        by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
      have 3: ¬ lnull (lbutlast (ltl xsa)) ⟶
```

¬ *lnull* (*ltl* (*lbutlast xsa*)) ⟶
(∃ *xs*. *ltl* (*lbutlast* (*ltl xsa*)) = *lbutlast* (*ltl xs*) ∧
ltl (*ltl* (*lbutlast xsa*)) = *ltl* (*lbutlast xs*) ∧ ¬ *lfinite xs*)
by (*metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl*)
show *?thesis* **using** *1 2 3* **by** *auto*
qed
qed
qed

lemma *lbutlast-not-lfinite*:
assumes ¬ *lfinite xs*
shows *lbutlast xs* = *xs*
using *assms*
by (*coinduction arbitrary*: *xs*)
(*metis lappend-inf lbutlast-snoc lfinite-ltl*)

lemma *lbutlast-lfinite*:
lfinite (*lbutlast xs*) ⟷ *lfinite xs*
proof
show *lfinite* (*lbutlast xs*) ⟹ *lfinite xs*
proof (*induct zs*≡*lbutlast xs rule*: *lfinite-induct*)
case *LNil*
then show *?case* **using** *lbutlast-not-lfinite* **by** *fastforce*
next
case *LCons*
then show *?case* **using** *lbutlast-not-lfinite* **by** *fastforce*
qed
show *lfinite xs* ⟹ *lfinite* (*lbutlast xs*)
proof (*induct rule*: *lfinite-induct*)
case (*LNil xs*)
then show *?case* **by** *simp*
next
case (*LCons xs*)
then show *?case* **by** (*simp add*: *lbutlast-ltl*)
qed
qed

lemma *llength-lbutlast* [*simp*]:
llength (*lbutlast xs*) = *epred* (*llength xs*)
by (*coinduction arbitrary*: *xs rule*: *enat-coinduct*)
(*simp*,
metis epred-enat-unfold lbutlast-ltl llength-def llength-eq-0 llength-ltl)

lemma *lbutlast-lappend*:
lbutlast (*lappend xs ys*) = (*if ys* = *LNil then lbutlast xs else lappend xs* (*lbutlast ys*))
proof (*cases lfinite xs*)
case *True*
then show *?thesis*
proof (*induct arbitrary*: *ys rule*: *lfinite-induct*)
case (*LNil xs*)

```
    then show ?case by (simp add: lnull-def)
    next
    case (LCons xs)
    then show ?case
      proof (cases lnull ys )
      case True
      then show ?thesis by (metis lappend-LNil2 lnull-def)
      next
      case False
      then show ?thesis
        proof (cases lnull xs)
        case True
        then show ?thesis
        using LCons.hyps(2) by auto
        next
        case False
        then show ?thesis using LCons by (auto simp add: llist.case-eq-if not-lnull-conv)
          (metis lappend.ctr(2) lbutlast-simps(3) llist.collapse(1) ltl-simps(2))
        qed
      qed
    qed
  next
  case False
  then show ?thesis by (metis lappend-inf lbutlast-snoc)
  qed

lemma lappend-lbutlast-llast-id-lfinite:
 assumes lfinite xs
         ¬ lnull xs
 shows   (lappend (lbutlast xs) (LCons (llast xs) LNil)) = xs
using assms
proof (induct rule: lfinite-induct)
case (LNil xs)
then show ?case by simp
next
case (LCons xs)
then show ?case
  proof (cases xs)
  case lfinite-LNil
  then show ?thesis using LCons by simp
  next
  case (lfinite-LConsI xs x)
  then show ?thesis using LCons
  by (auto simp add: llast-LCons)
    (metis lappend-code(1) lbutlast.ctr(1) llist.collapse(1) ltl-simps(2),
     metis lappend-code(2) lbutlast-simps(3) lhd-LCons-ltl)
qed
qed

lemma lappend-lbutlast-llast-id-not-lfinite:
```

assumes $\neg$*lfinite xs*
$\neg$ *lnull xs*
shows (*lappend* (*lbutlast xs*) (*LCons* (*llast xs*) *LNil*)) = *xs*
using *assms*
proof (*coinduction arbitrary*: *xs*)
case (*Eq-llist xsa*)
then show *?case*
by (*auto simp add*: *lbutlast-ltl llast-linfinite*)
(*metis lfinite-LNil lfinite-ltl llist.collapse*(*1*),
metis lbutlast.simps(*3*) *lbutlast-not-lfinite*)
qed

lemma *lappend-lbutlast-llast-id* [*simp*]:
shows $\neg$ *lnull xs* $\Longrightarrow$ (*lappend* (*lbutlast xs*) (*LCons* (*llast xs*) *LNil*)) = *xs*
using *lappend-lbutlast-llast-id-lfinite lappend-lbutlast-llast-id-not-lfinite* **by** *blast*

lemma *lbutlast-eq-LNil-conv*:
lbutlast xs = *LNil* $\longleftrightarrow$ *xs* =*LNil* $\vee$ ($\exists$ *x*. *xs* = (*LCons x LNil*))
by (*metis lbutlast.disc-iff*(*1*) *lbutlast-simps*(*2*) *lhd-LCons-ltl llist.collapse*(*1*) *llist.disc*(*1*))

lemma *lbutlast-eq-LCons-conv*:
lbutlast xs = (*LCons x ys*) $\longleftrightarrow$ *xs* = (*LCons x* (*lappend ys* (*LCons* (*llast xs*) *LNil*)))
by (*metis eq-LConsD lappend-code*(*2*) *lappend-lbutlast-llast-id lbutlast.ctr*(*1*) *lbutlast-snoc*)

lemma *lbutlast-conv-ltake*:
lbutlast xs = *ltake* (*epred* (*llength xs*)) *xs*
proof (*cases lfinite xs*)
case *True*
then show *?thesis*
proof (*induct rule*: *lfinite-induct*)
case (*LNil xs*)
then show *?case* **by** *simp*
next
case (*LCons xs*)
then show *?case*
by (*metis enat-le-plus-same*(*2*) *gen-llength-def lappend-lbutlast-llast-id-lfinite llength-code llength-lbutlast ltake-all ltake-lappend1*)
qed
next
case *False*
then show *?thesis*
proof (*coinduction arbitrary*: *xs*)
case (*Eq-llist xsa*)
then show *?case*
by (*auto simp add*: *not-lfinite-llength lbutlast-not-lfinite*)
(*metis epred-Infty epred-llength lbutlast-not-lfinite llength-eq-infty-conv-lfinite ltake-epred-ltl*)
qed
qed

lemma *lmap-lbutlast*:
lmap f (*lbutlast xs*) = *lbutlast* (*lmap f xs*)
by (*simp add*: *lbutlast-conv-ltake*)

lemma *snocs-eq-iff-lbutlast*:
lappend xs (*LCons x LNil*) = *ys* $\longleftrightarrow$
((*lfinite ys* $\wedge$ $\neg$ *lnull ys* $\wedge$ *lbutlast ys* = *xs* $\wedge$ *llast ys* = *x*)
$\vee$ ($\neg$ *lfinite ys* $\wedge$ *lbutlast xs* = *ys*))
by (*metis lappend.disc*(*2*) *lappend-inf lappend-lbutlast-llast-id lbutlast.ctr*(*1*) *lbutlast-snoc lfinite-LNil llast-lappend llast-singleton llist.discI*(*2*))

lemma *in-lset-lbutlastD*:
$x \in$ *lset*(*lbutlast xs*) $\Longrightarrow$ $x \in$ *lset xs*
by (*metis in-lset-lappend-iff lappend-ltake-ldrop lbutlast-conv-ltake*)

lemma *in-lset-lbutlast-lappendI*:
$x \in$ *lset* (*lbutlast xs*) $\vee$ (*lfinite xs* $\wedge$ $x \in$ *lset*(*lbutlast ys*)) $\Longrightarrow$
$x \in$ *lset* (*lbutlast* (*lappend xs ys*))
by (*metis empty-iff in-lset-lappend-iff in-lset-lbutlastD lbutlast-lappend lset-LNil*)

lemma *lnth-lbutlast*:
assumes *n*< *llength*(*lbutlast xs*)
shows *lnth* (*lbutlast xs*) *n* = *lnth xs n*
proof (*cases xs*)
case *LNil*
then show *?thesis* **by** *simp*
next
case (*LCons x21 x22*)
then show *?thesis* **by** (*metis assms lbutlast-conv-ltake llength-lbutlast lnth-ltake*)
qed

1.3 lfuse

lemma *lfuse-conv-lnull*:
lnull (*lfuse xs ys*) $\longleftrightarrow$ *lnull xs* $\wedge$ *lnull ys*
by (*simp add*: *lfuse-def*)

lemma *lfuse-LNil-1* [*simp*]:
lfuse LNil ys = *ys*
by (*simp add*: *lfuse-def*)

lemma *lfuse-LNil-2* [*simp*]:
lfuse xs LNil = *xs*
by (*metis lappend-lnull2 lfuse-def llist.discI*(*1*))

lemma *lfuse-One-a* [*simp*]:
assumes $\neg$ *lnull xs*
shows *lfuse* (*LCons* (*lhd xs*) *LNil*) *xs* = *xs*

using *assms* **by** (*simp add*: *lfuse-def*)

lemma *lfuse-One-b* [*simp*]:
 assumes ¬ *lnull xs*
 shows *lfuse xs* (*LCons y LNil*) = *xs*
using *assms*
by (*simp add*: *lfuse-def*)

lemma *lfuse-One-a-var*:
 shows *lfuse* (*LCons x LNil*) *ys* = (*if* ¬ *lnull ys then* (*LCons x* (*ltl ys*)) *else* (*LCons x LNil*))
unfolding *lfuse-def* **by** *simp*

lemma *lfuse-One-b-var*:
 shows *lfuse xs* (*LCons y LNil*) = (*if* ¬*lnull xs then xs else* (*LCons y LNil*))
unfolding *lfuse-def* **by** (*simp add*: *lappend-lnull1*)

lemma *lfuse-LCons-a* [*simp*]:
 lfuse (*LCons x xs*) *ys* =
 (*if lnull xs then* (*LCons x* (*ltl ys*)) *else* (*LCons x* (*lfuse xs ys*)))
by (*metis lappend-code*(*2*) *lappend-lnull1 lfuse-def llist.collapse*(*1*) *llist.disc*(*2*) *ltl-simps*(*1*))

lemma *lfuse-LCons-b* [*simp*]:
 lfuse xs (*LCons y ys*) =
 (*if* ¬ *lnull xs then lappend xs ys else* (*LCons y ys*))
unfolding *lfuse-def* **by** (*simp add*: *lappend-lnull1*)

lemma *lfuse-simps* [*simp*]:
 shows *lhd-lfuse*: *lhd* (*lfuse xs ys*) = (*if lnull xs then lhd ys else lhd xs*)
 and *ltl-lfuse*: *ltl* (*lfuse xs ys*) =
 (*if lnull xs*
 then ltl ys
 else (*if lnull* (*ltl ys*)
 then ltl xs
 else lappend (*ltl xs*) (*ltl ys*)))
by (*auto simp add*: *lfuse-def lappend-lnull2*)

lemma *lfuse-lbutlast*:
 assumes *llast xs* = *lfirst ys*
 shows *lfuse xs ys* = (*if lnull ys then xs else lappend* (*lbutlast xs*) *ys*)
using *assms*
by (*metis lappend-lbutlast-llast-id lappend-snocL1-conv-LCons2 lbutlast.ctr*(*1*) *lfirst-def lfuse-LNil-2 lfuse-def lhd-LCons-ltl llist.collapse*(*1*))

lemma *lfuse-llength*:
 shows *llength* (*lfuse xs ys*) = (*llength xs*) + (*if lnull xs then llength ys else epred*(*llength ys*))
unfolding *lfuse-def* **by** (*simp add*: *llist.case-eq-if epred-llength*)

lemma *not-lnull-llength*:
 ¬ *lnull xs* ⟷ *1* ≤ *llength xs*
by (*metis gr-zeroI ileI1 llength-eq-0 not-one-le-zero one-eSuc*)

lemma *lfuse-llength-atmost-one*:
 shows *llength* (*lfuse xs ys*) $\leq$ *1* $\longleftrightarrow$ *llength xs* $\leq$*1* $\wedge$ *llength ys* $\leq$ *1*
proof (*cases lnull xs*)
case *True*
then show *?thesis*
by (*metis lfuse-LNil-1 llength-LNil lnull-def zero-le*)
next
case *False*
then show *?thesis* **unfolding** *lfuse-llength*
by *auto*
 (*meson dual-order.trans enat-le-plus-same*(*1*),
 metis add.commute add-increasing2 co.enat.exhaust-sel not-lnull-llength order-antisym-conv
 order-refl plus-1-eSuc(*2*) *zero-le*,
 metis add-decreasing2 epred-1 epred-le-epredI)
qed

lemma *lfuse-llength-less-a*:
 assumes *1* < *llength xs*
 1 < *llength ys*
 llast xs = *lfirst ys*
 lfinite xs
 shows *llength xs* < *llength* (*lfuse xs ys*)
unfolding *lfuse-def*
using *assms*
by (*auto simp add*: *lfinite-llength-enat*)
 (*metis co.enat.exhaust-sel epred-llength linorder-neq-iff llength-eq-0 one-eSuc*)

lemma *lfuse-llength-less-b*:
 assumes *1* < *llength xs*
 1 < *llength ys*
 llast xs = *lfirst ys*
 lfinite ys
 shows *llength ys* < *llength* (*lfuse xs ys*)
proof −
 have *1*: *lfuse xs ys* = *lappend* (*lbutlast xs*) *ys*
 by (*metis assms*(*2*) *assms*(*3*) *lfuse-lbutlast llength-lnull not-less-zero*)
 have *2*: *llength* (*lappend* (*lbutlast xs*) *ys*) = *epred*(*llength xs*) + *llength ys*
 by *simp*
 have *3*: *0* < *epred*(*llength xs*)
 by (*metis assms*(*1*) *co.enat.exhaust-sel gr-zeroI less-numeral-extra*(*4*) *not-one-less-zero one-eSuc*)
 have *4*: *llength ys* < *epred*(*llength xs*) + *llength ys*
 using *3 assms*(*4*) *lfinite-llength-enat* **by** *auto*
 show *?thesis*
 using *1 2 4* **by** *presburger*
qed

lemma *lfuse-llength-le-a*:
 llength xs $\leq$ *llength* (*lfuse xs ys*)
by (*simp add*: *lfuse-llength*)

lemma *lfuse-llength-le-b*:
assumes *llast xs* = *lfirst ys*
shows *llength ys* ≤ *llength* (*lfuse xs ys*)
by (*simp add*: *assms lfuse-lbutlast*)

lemma *lfuse-lnth-a*:
assumes (*enat i*) < *llength xs*
shows *lnth* (*lfuse xs ys*) *i* = *lnth xs i*
using *assms*
by (*simp add*: *lfuse-def lnth-lappend1*)

lemma *lfuse-lnth-b*:
assumes *llength xs* ≤ (*enat i*)
(*enat i*) < *llength* (*lfuse xs ys*)
shows *lnth* (*lfuse xs ys*) *i* = (*lnth ys* (*i* − (*the-enat*(*epred*(*llength xs*)))))
proof (*cases* ¬ *lnull xs* ∧ ¬*lnull ys*)
case *True*
then show *?thesis*
proof −
have *1*: *lfinite xs*
using *assms*(*1*) *lfinite-ldropn lnull-imp-lfinite lnull-ldropn* **by** *blast*
obtain *n* **where** *15*: (*enat n*) = *llength xs* **using** *1* **by** (*metis assms*(*1*) *enat-ile*)
have *16*: (*the-enat*(*epred*(*llength xs*))) = *n*−*1*
by (*metis 15 epred-enat the-enat.simps*)
have *17*: *0*<*n*
by (*metis 15 True gr0I llength-eq-0 zero-enat-def*)
have *2*: *lfuse xs ys* = *lappend xs* (*ltl ys*)
unfolding *lfuse-def* **using** *assms True* **by** *presburger*
have *3*: *n*−*1* ≤ *i*
by (*metis 15 17 Suc-pred′ assms*(*1*) *enat-ord-simps*(*2*) *leD le-imp-less-Suc wlog-linorder-le*)
have *4*: (*Suc* (*i* − *n*)) = (*i* − (*n*−*1*))
by (*metis 15 17 Suc-diff-eq-diff-pred Suc-diff-le assms*(*1*) *enat-ord-simps*(*1*))
have *5*: *lnth* (*ltl ys*) (*i* − *the-enat* (*llength xs*)) = (*lnth ys* (*i* − (*the-enat*(*epred*(*llength xs*)))))
using *True 3 lnth-ltl*[*of ys* (*i* − *the-enat* (*llength xs*))] **by** (*metis 15 16 4 the-enat.simps*)
show *?thesis* **using** *lnth-lappend*[*of xs* (*ltl ys*)] **by** (*simp add*: *2 5 assms*(*1*) *leD*)
qed
next
case *False*
then show *?thesis* **using** *assms* **unfolding** *lfuse-def*
using *lappend-lnull1* **by** *fastforce*
qed

lemma *lfuse-lnth-c*:
assumes *epred*(*llength xs*) ≤ (*enat i*)
(*enat i*) < *llength* (*lfuse xs ys*)
llast xs = *lfirst ys*
shows *lnth* (*lfuse xs ys*) *i* =
(*if lnull ys then lnth xs* (*the-enat* (*epred*(*llength xs*)))
else *lnth ys* (*i* − (*the-enat* (*epred*(*llength xs*)))))

proof (*cases* ¬ *lnull xs* ∧ ¬*lnull ys*)
case *True*
then show *?thesis*
  using *assms*
  by (*simp add*: *leD lfuse-lbutlast lnth-lappend*)
next
case *False*
then show *?thesis*
   proof (*cases lnull xs*)
   case *True*
   then show *?thesis*
      using *assms*
     by (*metis epred-0 gr-implies-not-zero lfuse-conv-lnull lfuse-lnth-b llength-lnull*)
   next
   case *False*
   then show *?thesis*
      using *assms*
      by (*metis co.enat.exhaust-sel iless-Suc-eq leD lfuse-lbutlast llength-eq-0*
          llength-lbutlast lnth-lappend nle-le the-enat.simps)
   qed
qed

lemma *lfirst-lfuse-1*:
 shows *lfirst* (*lfuse xs ys*) = (*if* ¬ *lnull xs then lfirst xs else lfirst ys*)
by (*simp add*: *lfirst-def*)

lemma *llast-lfuse*:
 assumes ¬ *lnull xs*
   ¬ *lnull ys*
   lfinite xs
   lfinite ys
   llast xs = *lfirst ys*
 shows *llast*(*lfuse xs ys*) = *llast ys*
using *assms*
by (*metis lfirst-def lfuse-def lhd-LCons-ltl llast-LCons llast-lappend*)

lemma *lfuse-assoc*:
assumes ¬ *lnull xs*
   ¬ *lnull ys*
   ¬ *lnull zs*
shows (*lfuse xs* (*lfuse ys zs*)) = (*lfuse* (*lfuse xs ys*) *zs*)
using *assms*
by (*metis lappend-assoc lappend-ltl lfuse-def lfuse-conv-lnull*)

lemma *lfirst-llast*:
 assumes *i*< *llength xs*
 shows *llast* (*ltake* (*Suc i*) *xs*) = *lfirst* (*ldropn i xs*)
using *assms*
by (*simp add*: *lfirst-def lhd-ldropn ltake-Suc-conv-snoc-lnth*)

lemma *ltake-lfuse*:
shows *ltake* (*llength xs*) (*lfuse xs ys*) = *xs*
by (*metis dual-order.refl lappend-lnull2 lfuse-def ltake-all ltake-lappend1*)

lemma *llast-lfirst-LNil*:
llast LNil = *lfirst LNil*
by (*simp add*: *lfirst-def lhd-def llast-LNil*)

lemma *ldrop-lappend-either-LNil*:
assumes *lnull xs* ∨ *lnull ys*
shows *ldrop* (*llength xs*) (*lappend xs ys*) =
(*if lfinite xs then ys else LNil*)
proof –
have *1*: *lnull xs* ⟹ *?thesis*
by (*simp add*: *lappend-lnull1*)
have *2*: *lnull ys* ⟹ *?thesis*
by (*metis dual-order.refl lappend-LNil2 ldrop-eq-LNil lnull-def*)
show *?thesis*
using *1 2 assms* **by** *blast*
qed

lemma *ldrop-lfuse*:
assumes *llast xs* = *lfirst ys*
shows *ldrop* (*if* ¬ *lnull xs* ∧ ¬ *lnull ys then epred*(*llength xs*) *else* (*llength xs*)) (*lfuse xs ys*) =
(*if lfinite xs then ys else LNil*)
proof –
have *3*: *lfuse xs ys* = (*if* ¬ *lnull xs* ∧ ¬ *lnull ys*
then lappend (*lbutlast xs*) *ys*
else lappend xs ys)
by (*meson assms lfuse-def lfuse-lbutlast*)
have *4*: *ldrop* ((*if* ¬ *lnull xs* ∧ ¬ *lnull ys then epred*(*llength xs*) *else* (*llength xs*)))
(*if* ¬ *lnull xs* ∧ ¬ *lnull ys*
then lappend (*lbutlast xs*) *ys*
else lappend xs ys) = (*if lfinite xs then ys else LNil*)
proof (*cases lfinite xs*)
case *True*
then show *?thesis*
by (*simp add*: *ldrop-lappend lfinite-llength-enat*)
next
case *False*
then show *?thesis*
by *simp*
(*metis epred-Infty llength-eq-infty-conv-lfinite lnull-imp-lfinite*)
qed
show *?thesis*
by (*simp add*: *3 4*)
qed

lemma *ldrop-lfuse-a*:

assumes ¬ *lnull xs*
¬ *lnull ys*
llast xs = *lfirst ys*
shows *ldrop* (*epred*(*llength xs*)) (*lfuse xs ys*) =
(*if lfinite xs then ys else LNil*)
using *assms*
using *ldrop-lfuse* **by** *force*

lemma *lfuse-ltake-ldrop*:
assumes *i* < *llength xs*
shows *lfuse* (*ltake* (*eSuc i*) *xs*) (*ldrop i xs*) = *xs*
using *assms*
by (*metis lappend-ltake-ldrop ldrop-eSuc-conv-ltl ldrop-lnull leD lfuse-def lnull-ldrop ltake.disc*(*2*) *zero-ne-eSuc*)

lemma *lset-lfuse*:
shows *lset* (*lfuse xs ys*) =
(*if lfinite xs then*
(*if* ¬ *lnull xs* ∧ ¬ *lnull ys then lset xs* ∪ *lset* (*ltl ys*) *else lset xs* ∪ *lset ys*)
else lset xs)
by (*simp add*: *lappend-inf lfuse-def*)

lemma *mono-llist-lfuse2* [*partial-function-mono*]:
mono-llist A ⟹ *mono-llist* (λ*f*. *lfuse xs* (*A f*))
by (*auto intro*!: *monotoneI lprefix-lappend-sameI simp add*: *lfuse-def lnull-lprefix llist.case-eq-if fun-ord-def lprefix-ltlI monotone-def dest*: *monotoneD*)
(*metis llist.collapse*(*1*) *lprefix-ltlI ltl-simps*(*1*))

lemma *mono2mono-lfuse2* [*THEN llist.mono2mono*, *cont-intro*, *simp*]:
shows *monotone-lfuse2*: *monotone* (*lprefix*) (*lprefix*) (*lfuse xs*)
by (*rule monotoneI*)
(*metis Coinductive-List.finite-lprefix-def Coinductive-List.finite-lprefix-nitpick-simps*(*1*) *lfuse-def lprefix-LNil lprefix-lappend-sameI ltl-simps*(*1*) *monotoneD monotone-ltl*)

lemma *silys*:
assumes *f* = *g*
shows *mcont lSup* (*lprefix*) *lSup* (*lprefix*) *f* =
mcont lSup (*lprefix*) *lSup* (*lprefix*) *g*
using *assms* **by** *auto*

lemma *lfuse-LNil-eq-id*:
lfuse LNil = *id*
by (*simp add*: *fun-eq-iff llist.case-eq-if*)

lemma *mcont2mcont-lfuse2* [*THEN llist.mcont2mcont*, *cont-intro*, *simp*]:
  shows *mcont-lfuse2*: *mcont lSup* (*lprefix*) *lSup* (*lprefix*) (*lfuse xs*)
proof(*cases lfinite xs*)
  case *True*
  thus *?thesis*
 proof *induct*
  case *lfinite-LNil*
  then show *?case* **using** *lfuse-LNil-eq-id* **by** *simp*
  next
  case (*lfinite-LConsI xs x*)
  then show *?case* **by** (*simp add*:*monotone-lfuse2*)
 qed
next
  case *False*
  hence *lfuse xs* = (λ-. *xs*)
    by (*simp add*: *fun-eq-iff lappend-inf lfuse-def*)
  thus *?thesis* **by**(*simp add*: *ccpo.cont-const*[*OF llist-ccpo*])
qed

lemma *lfuse-inf*: ¬ *lfinite xs* ⟹ *lfuse xs ys* = *xs*
by (*simp add*: *lappend-inf lfuse-def*)

lemma *llist-all2-lfuseI*:
  assumes *1*: *llist-all2 P xs ys*
  and *2*: ⟦ *lfinite xs*; *lfinite ys* ⟧ ⟹ *llist-all2 P xs′ ys′*
  shows *llist-all2 P* (*lfuse xs xs′*) (*lfuse ys ys′*)
proof(*cases lfinite xs*)
  case *True*
  with *1* **have** *lfinite ys* **by**(*auto dest*: *llist-all2-lfiniteD*)
  from *1 2*[*OF True this*] **show** *?thesis*
   proof (*coinduction arbitrary*: *xs ys*)
   case *LNil*
   then show *?case* **by** (*simp add*: *lfuse-conv-lnull llist-all2-lnullD*)
   next
   case *LCons*
   then show *?case*
   by (*metis* (*no-types*, *lifting*) *lfuse-def llist.rel-sel llist-all2-lappendI*)
   qed
 next
  case *False*
  with *1* **have** ¬ *lfinite ys* **by**(*auto dest*: *llist-all2-lfiniteD*)
  with *False 1* **show** *?thesis*
  by (*simp add*: *lfuse-inf*)
qed

1.4 ridx and lidx

lemma *ridx-lidx*:
ridx (<) *xs* = *lidx xs*
unfolding *lidx-def ridx-def*
by *simp*

lemma *ridx-expand-1*:
$ridx\ R\ xs \longleftrightarrow lnull\ xs \lor llength\ xs = 1 \lor$
$(1 < llength\ xs \land (\forall n.\ (Suc\ n){<}llength\ xs \longrightarrow R\ (lnth\ xs\ n)\ \ (lnth\ xs\ (Suc\ n))))$
unfolding *ridx-def*
by (*metis Zero-not-Suc enat-0-iff*(*1*) *gr-implies-not-zero iless-eSuc0 linorder-cases llength-eq-0 one-eSuc*)

lemma *lidx-expand-1*:
$lidx\ xs \longleftrightarrow lnull\ xs \lor llength\ xs = 1 \lor$
$(1 < llength\ xs \land (\forall n.\ (Suc\ n){<}llength\ xs \longrightarrow lnth\ xs\ n < lnth\ xs\ (Suc\ n)))$
using *ridx-lidx ridx-expand-1* **by** *blast*

lemma *ridx-LCons*:
$ridx\ R\ (LCons\ x\ xs) \longleftrightarrow$
$lnull\ xs \lor$
$(0 < llength\ xs \land$
$(\forall n.\ (Suc\ n){<}\ eSuc\ (llength\ xs)\ \ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n))))$
unfolding *ridx-def*
using *enat-0-iff*(*1*) **by** *force*

lemma *lidx-LCons*:
$lidx\ (LCons\ x\ xs) \longleftrightarrow$
$lnull\ xs \lor$
$(0 < llength\ xs \land$
$(\forall n.\ (Suc\ n){<}\ eSuc\ (llength\ xs)\ \ \longrightarrow lnth\ (LCons\ x\ xs)\ n < lnth\ (LCons\ x\ xs)\ (Suc\ n)))$
using *ridx-lidx*[*of* (*LCons x xs*)] *ridx-LCons*[*of* (<) *x xs*] **by** *blast*

lemma *ridx-LCons-conv*:
$(0 < llength\ xs \land$
$(\forall n.\ (Suc\ n){<}\ eSuc\ (llength\ xs)\ \ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n))))$
$\longleftrightarrow (0 < llength\ xs \land$
$R\ x\ (lhd\ xs) \land$
$(\forall n.\ 0{\le}\ n \land (Suc\ n){<}\ \ (llength\ xs)\ \ \longrightarrow R\ (lnth\ xs\ n)\ \ (lnth\ xs\ (Suc\ n))))$
proof −
have *1*: $(0 < llength\ xs \land$
$(\forall n.\ (Suc\ n){<}\ eSuc\ (llength\ xs)\ \ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ \ (lnth\ (LCons\ x\ xs)\ (Suc\ n)))) \longleftrightarrow$
$(0 < llength\ xs \land$
$(\forall n.\ 0{\le}\ n \land (Suc\ n){<}\ eSuc\ (llength\ xs)\ \ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ \ (lnth\ (LCons\ x\ xs)\ (Suc$
$n))))$
by *blast*
have *2*: $(0 < llength\ xs \land$
$(\forall n.\ 0{\le}\ n \land (Suc\ n){<}\ eSuc\ (llength\ xs)\ \ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc$
$n)))) \longleftrightarrow$
$(0 < llength\ xs \land$

$R\ x\ \ (lhd\ xs) \wedge$
$(\forall n.\ 1\leq\ n \wedge (Suc\ n) < \ eSuc\ (llength\ xs)\ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n))))$
by (*metis Extended-Nat.eSuc-mono One-nat-def eSuc-enat gr-implies-not-zero ldropn-0 less-Suc-eq lhd-ldropn lnth-0 lnth-Suc-LCons not-le-imp-less zero-enat-def*)
have *3*: $(0 < llength\ xs \wedge$
$R\ x\ (lhd\ xs) \wedge$
$(\forall n.\ 1\leq\ n \wedge (Suc\ n) < \ eSuc\ (llength\ xs)\ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ \ (lnth\ (LCons\ x\ xs)\ (Suc\ n)))) \longleftrightarrow$
$(0 < llength\ xs \wedge$
$R\ x\ (lhd\ xs) \wedge$
$(\forall n.\ 1\leq\ n \wedge (n) < \ \ (llength\ xs)\ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ \ (lnth\ (LCons\ x\ xs)\ (Suc\ n))))$
using *Suc-ile-eq* **by** *auto*
have *4*: $(0 < llength\ xs \wedge$
$R\ x\ (lhd\ xs) \wedge$
$(\forall n.\ 1\leq\ n \wedge (n) < \ \ (llength\ xs)\ \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ \ (lnth\ (LCons\ x\ xs)\ (Suc\ n)))) \longleftrightarrow$
$(0 < llength\ xs \wedge$
$R\ x\ (lhd\ xs) \wedge$
$(\forall n.\ 1\leq\ n \wedge (n) < \ \ (llength\ xs)\ \longrightarrow R\ (lnth\ xs\ (n\ -1))\ (\ lnth\ xs\ (\ n))))$
by (*metis le-add-diff-inverse llist.disc*(*2*) *lnth-ltl ltl-simps*(*2*) *plus-1-eq-Suc*)
have *5*: $(0 < llength\ xs \wedge$
$R\ x\ (lhd\ xs) \wedge$
$(\forall n.\ 1\leq\ n \wedge (n) < \ \ (llength\ xs)\ \longrightarrow R\ (lnth\ xs\ (n\ -1))\ (\ lnth\ xs\ (\ n)))) \longleftrightarrow$
$(0 < llength\ xs \wedge$
$R\ x\ \ (lhd\ xs) \wedge$
$(\forall n.\ 0\leq\ (n-1) \wedge (Suc\ (n-1)) < \ \ (llength\ xs)\ \longrightarrow R\ (lnth\ xs\ (n\ -1))\ \ (lnth\ xs\ (Suc\ (n\ -1)))))$
by (*metis diff-Suc-1 le0 le-add1 le-add-diff-inverse plus-1-eq-Suc*)
have *6*: $(0 < llength\ xs \wedge$
$R\ x\ (lhd\ xs) \wedge$
$(\forall n.\ 0\leq\ (n-1) \wedge (Suc\ (n-1)) < \ \ (llength\ xs)\ \longrightarrow R\ (lnth\ xs\ (n\ -1))\ \ (lnth\ xs\ (Suc\ (n\ -1))))) \longleftrightarrow$
$(0 < llength\ xs \wedge$
$R\ x\ (lhd\ xs) \wedge$
$(\forall n.\ 0\leq\ n \wedge (Suc\ n) < \ \ (llength\ xs)\ \longrightarrow R\ (lnth\ xs\ n)\ \ (lnth\ xs\ (Suc\ n))))$
by (*metis diff-Suc-1*)
show *?thesis*
using *2 3 4 5 6* **by** *auto*
qed

lemma *lidx-LCons-conv*:
$(0 < llength\ xs \wedge (\forall n.\ (Suc\ n) < \ eSuc\ (llength\ xs)\ \longrightarrow lnth\ (LCons\ x\ xs)\ n < lnth\ (LCons\ x\ xs)\ (Suc\ n)))$
$\longleftrightarrow (0 < llength\ xs \wedge$
$x < \ lhd\ xs \wedge$
$(\forall n.\ 0\leq\ n \wedge (Suc\ n) < \ \ (llength\ xs)\ \longrightarrow lnth\ xs\ n < lnth\ xs\ (Suc\ n)))$
using *ridx-LCons-conv*
by *metis*

lemma *ridx-LCons-1* [*simp*]:

ridx R (*LCons x xs*) ⟷ *lnull xs* ∨ (*0* < *llength xs* ∧ *R x* (*lhd xs*) ∧ *ridx R xs*)
unfolding *ridx-def*
using *ridx-LCons-conv*[*of xs R x*]
by (*auto simp add*: *enat-0-iff*(*1*))

lemma *lidx-LCons-1* [*simp*]:
lidx (*LCons x xs*) ⟷ *lnull xs* ∨ (*0* < *llength xs* ∧ *x*< *lhd xs* ∧ *lidx xs*)
using *ridx-lidx*[*of LCons x xs*] *ridx-lidx*[*of xs*]*ridx-LCons-1*[*of* (<) *x xs*] **by** *blast*

lemma *ridx-less*:
assumes *ridx R xs*
Suc(*n*+*k*) < *llength xs*
transp R
shows *R* (*lnth xs n*) (*lnth xs* (*Suc*(*n*+*k*)))
using *assms* **unfolding** *ridx-def*
proof (*induct k*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
by (*metis Suc-ile-eq add-Suc-right order-less-imp-le transpE*)
qed

lemma *lidx-less*:
assumes *lidx xs*
Suc(*n*+*k*) < *llength xs*
shows *lnth xs n* < *lnth xs* (*Suc*(*n*+*k*))
using *assms* *ridx-lidx ridx-less*[*of* (<) *xs n k*] *transp-less* **by** *blast*

lemma *ridx-less-eq*:
assumes *ridx R xs*
k ≤ *j*
j < *llength xs*
transp R
reflp R
shows *R* (*lnth xs k*) (*lnth xs j*)
proof (*cases k*=*j*)
case *True*
then show *?thesis* **using** *assms* **by** (*simp add*: *reflp-def*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis less-iff-Suc-add order-neq-le-trans ridx-less*)
qed

lemma *lidx-less-eq*:
assumes *lidx xs*
k ≤ *j*

j < *llength xs*
shows *lnth xs k* ≤ *lnth xs j*
using *assms ridx-lidx ridx-less-eq*[*of* (<) *xs k j*]
by (*metis dual-order.order-iff-strict less-iff-Suc-add lidx-less*)

lemma *ridx-gr-first*:
assumes *ridx R xs*
0<i
i < *llength xs*
transp R
shows *R* (*lnth xs 0*) (*lnth xs i*)
using *assms ridx-less*[*of R xs 0 i*−*1*] **unfolding** *ridx-def* **by** *simp*

lemma *lidx-gr-first*:
assumes *lidx xs*
0<i
i < *llength xs*
shows (*lnth xs 0*) < *lnth xs i*
using *assms lidx-less*[*of xs 0 i*−*1*] **unfolding** *lidx-def* **by** *simp*

lemma *ridx-ltake-a*:
assumes *ridx R xs*
n ≤ *llength xs*
shows *ridx R* (*ltake n xs*)
using *assms*
unfolding *ridx-def*
by *simp*
(*metis Suc-ile-eq dual-order.strict-trans1 lnth-ltake order-less-imp-le*)

lemma *lidx-ltake-a*:
assumes *lidx xs*
n ≤ *llength xs*
shows *lidx* (*ltake n xs*)
using *assms*
using *ridx-lidx ridx-ltake-a* **by** *blast*

lemma *ridx-lappend-lfinite*:
assumes *lfinite xs*
shows *ridx R* (*lappend xs ys*) ⟷
ridx R xs ∧ ((*lnull xs* ∨ *lnull ys*) ∨ *R* (*llast xs*) (*lhd ys*)) ∧ *ridx R ys*
using *assms*
proof (*induction xs arbitrary*: *ys*)
case *lfinite-LNil*
then show *?case* **by** (*simp add*: *ridx-expand-1*)
next
case (*lfinite-LConsI xs x*)
then show *?case*
proof (*cases lnull xs*)
case *True*
then show *?thesis*

by (*metis lappend-code*(*1*) *lappend-code*(*2*) *llast-singleton llength-LCons llist.collapse*(*1*)
one-eSuc order-neq-le-trans ridx-LCons-1 ridx-expand-1 zero-le)
next
case *False*
then show *?thesis*
by (*simp add*: *lfinite-LConsI.IH llast-LCons*)
qed
qed

lemma *lidx-lappend-lfinite*:
assumes *lfinite xs*
shows *lidx* (*lappend xs ys*) ⟷
lidx xs ∧ ((*lnull xs* ∨ *lnull ys*) ∨ (*llast xs*) < (*lhd ys*)) ∧ *lidx ys*
using *assms* **by** (*metis ridx-lappend-lfinite ridx-lidx*)

lemma *ridx-ldrop*:
assumes *ridx R xs*
n ≤ *llength xs*
shows *ridx R* (*ldrop n xs*)
proof −
have *1*: *xs* = *lappend* (*ltake n xs*) (*ldrop n xs*)
by (*simp add*: *lappend-ltake-ldrop*)
have *2*: ¬ *lfinite* (*ltake n xs*) ⟹ *?thesis*
by (*simp add*: *ridx-expand-1*)
have *3*: *lfinite* (*ltake n xs*) ⟹ *ridx R* (*lappend* (*ltake n xs*) (*ldrop n xs*)) ⟷
ridx R (*ltake n xs*) ∧
((*lnull* (*ltake n xs*) ∨ *lnull* (*ldrop n xs*)) ∨ *R* (*llast* (*ltake n xs*)) (*lhd* (*ldrop n xs*))) ∧
ridx R (*ldrop n xs*)
using *ridx-lappend-lfinite*[*of* (*ltake n xs*) *R* (*ldrop n xs*)] **by** *blast*
have *4*: *lfinite* (*ltake n xs*) ⟹ *?thesis*
using *1 3 assms* **by** *metis*
show *?thesis* **using** *2 4* **by** *blast*
qed

lemma *lidx-ldrop*:
assumes *lidx xs*
n ≤ *llength xs*
shows *lidx* (*ldrop n xs*)
using *assms ridx-ldrop ridx-lidx* **by** *blast*

lemma *ridx-ltake-all*:
assumes ⋀*n*. *n*≤ *llength xs* ⟹ *ridx R* (*ltake* (*enat n*) *xs*)
shows *ridx R xs*
using *assms*
unfolding *ridx-def*
by *auto*
(*metis Suc-ile-eq dual-order.refl eSuc-enat ile-eSuc lessI lnth-ltake*)

lemma *lidx-ltake-all*:
assumes ⋀*n*. *n*≤ *llength xs* ⟹ *lidx* (*ltake* (*enat n*) *xs*)

shows *lidx xs*
using *assms ridx-ltake-all ridx-lidx* **by** *blast*

lemma *ridx-ltake*:
assumes *ridx R* (*ltake n xs*)
$n \leq$ *llength xs*
$k \leq n$
shows *ridx R* (*ltake* (*enat k*) *xs*)
using *assms*
using *ridx-ltake-a* **by** *fastforce*

lemma *lidx-ltake*:
assumes *lidx* (*ltake n xs*)
$n \leq$ *llength xs*
$k \leq n$
shows *lidx* (*ltake* (*enat k*) *xs*)
using *assms ridx-ltake ridx-lidx* **by** *blast*

lemma *lidx-imp-lsorted*:
assumes *lidx xs*
shows *lsorted xs*
using *assms*
by (*metis* (*no-types*, *lifting*) *less-imp-le lhd-LCons-ltl lidx-LCons-1 lsorted-coinduct′*)

lemma *lidx-imp-ldistinct*:
assumes *lidx xs*
shows *ldistinct xs*
using *assms*
proof (*coinduction arbitrary*: *xs*)
case (*ldistinct xsa*)
then show *?case*
proof −
have *1*: *lhd xsa* $\notin$ *lset* (*ltl xsa*)
by (*metis empty-iff lappend-code*(*1*) *lappend-lnull2 ldistinct*(*1*) *ldistinct*(*2*) *leD lhd-LCons-ltl lidx-LCons-1 lidx-imp-lsorted lmember-code*(*2*) *lset-LNil lset-lmember lsortedD*)
have *2*: (($\exists$ *xs*. *ltl xsa* = *xs* $\wedge$ *lidx xs*) $\vee$ *ldistinct* (*ltl xsa*))
unfolding *lidx-def*
by (*metis Extended-Nat.eSuc-mono eSuc-enat ldistinct*(*1*) *ldistinct*(*2*) *lhd-LCons-ltl lidx-def llength-LCons lnth-ltl*)
show *?thesis*
using *1 2* **by** *auto*
qed
qed

lemma *ldistinct-Ex1*:
assumes *ldistinct xs*
$x \in$ *lset xs*
shows $\exists!i$. $i<$ *llength xs* $\wedge$ (*lnth xs i*) = *x*
using *assms*
by (*metis in-lset-conv-lnth ldistinct-conv-lnth*)

lemma *lidx-lset-eq*:
 assumes *lidx xs*
 lidx ys
 lset xs = *lset ys*
 shows *xs* = *ys*
using *assms*
 by (*simp add*: *lidx-imp-ldistinct lidx-imp-lsorted lsorted-ldistinct-lset-unique*)

lemma *ridx-lfuse-lfirst-llast*:
 assumes *ridx R ys*
 (*lnth ys 0*) = (*0*::*nat*)
 ridx R zs
 (*lnth zs 0*) = *0*
 lfinite ys
 lfinite xs
 ¬ *lnull xs*
 ¬ *lnull zs*
 llast ys = *cp*
 shows *llast ys* = *lfirst*(*lmap* (*λx*. *x*+*cp*) *zs*)
using *assms* **unfolding** *lfirst-def*
by (*simp add*: *lnth-0-conv-lhd*)

lemma *lidx-lfuse-lfirst-llast*:
 assumes *lidx ys*
 (*lnth ys 0*) = *0*
 lidx zs
 (*lnth zs 0*) = *0*
 lfinite ys
 lfinite xs
 ¬ *lnull xs*
 ¬ *lnull zs*
 llast ys = *cp*
 shows *llast ys* = *lfirst*(*lmap* (*λx*. *x*+*cp*) *zs*)
using *assms*
by (*simp add*: *lfirst-def lnth-0-conv-lhd*)

lemma *ridx-lfuse-lnth-cp*:
 assumes *ridx R ys*
 (*lnth ys 0*) = *0*
 ridx R zs
 (*lnth zs 0*) = *0*
 lfinite ys
 lfinite zs
 lfinite xs
 ¬ *lnull xs*
 ¬ *lnull zs*
 ¬ *lnull ys*
 llast ys = *cp*
 llast zs = *the-enat*(*epred* (*llength xs*)) − *cp*

i < (*llength zs*)
cp < *llength xs*
shows *lnth* (*lfuse ys* (*lmap* (*λx. x+cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) = *cp* + (*lnth zs i*)
proof −
have *1*: *llast ys* = *lfirst*(*lmap* (*λx. x+cp*) *zs*)
by (*metis assms*(*11*) *assms*(*4*) *assms*(*9*) *lfirst-def llist.map-sel*(*1*) *lnth-0-conv-lhd plus-nat.add-0*)
have *3*: *lfirst*(*lmap* (*λx. x+cp*) *zs*) = *cp* + (*lnth zs 0*)
using *1 assms*(*11*) *assms*(*4*) **by** *force*
have *8*: *i*≤ *epred*(*llength zs*)
using *assms*
by (*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0*)
have *81*: *enat* (*the-enat* (*epred* (*llength ys*)) + *i*) ≤
enat (*the-enat* (*epred* (*llength ys*)) + (*the-enat* (*epred*(*llength zs*))))
by (*metis 8 add-mono-thms-linordered-semiring*(*2*) *assms*(*6*) *assms*(*9*) *co.enat.exhaust-sel enat-ord-simps*(*1*) *enat-ord-simps*(*4*) *enat-the-enat ile-eSuc iless-Suc-eq lfinite-llength-enat llength-eq-0 order-le-less-trans*)
have *9*: *epred*(*llength zs*) < *llength zs*
by (*metis assms*(*6*) *assms*(*9*) *co.enat.exhaust-sel ile-eSuc iless-Suc-eq lfinite-llength-enat llength-eq-0 order-neq-le-trans*)
have *10*: *epred* (*llength ys*) ≤ *enat* (*the-enat* (*epred* (*llength ys*)) + (*i*::*nat*))
by (*metis assms*(*10*) *assms*(*5*) *co.enat.exhaust-sel enat-ord-simps*(*1*) *enat-the-enat ile-eSuc infinity-ileE le-add1 lfinite-llength-enat llength-eq-0*)
have *83*: *enat* (*the-enat* (*epred* (*llength ys*)) + (*the-enat* (*epred*(*llength zs*)))) =
enat (*the-enat* (*epred* (*llength ys*) + *epred*(*llength zs*)))
by (*metis 10 9 enat-ord-code*(*4*) *enat-the-enat leD order-less-imp-le plus-enat-simps*(*1*))
have *84*: *enat* (*the-enat* (*epred* (*llength ys*) + *epred*(*llength zs*))) =
((*epred* (*llength ys*) + *epred*(*llength zs*)))
by (*metis 10 9 enat-ord-code*(*4*) *enat-the-enat leD order-less-imp-not-less plus-enat-simps*(*1*))
have *85*: *epred*(*llength ys*) < *llength ys*
by (*metis 10 assms*(*10*) *co.enat.exhaust-sel enat-ord-code*(*4*) *enat-the-enat ile-eSuc iless-Suc-eq leD llength-eq-0 order-neq-le-trans*)
have *86*: *llength* (*lfuse ys* (*lmap* (*λx*::*nat. x* + *cp*) *zs*)) = *llength ys* + *epred*(*llength zs*)
by (*simp add*: *assms*(*10*) *lfuse-llength*)
have *87*: ((*epred* (*llength ys*) + *epred*(*llength zs*))) < *llength ys* + *epred*(*llength zs*)
by (*metis 83 84 85 add.commute enat-le-plus-same*(*2*) *enat-less-enat-plusI2 enat-the-enat infinity-ileE*)
have *82*: *enat* (*the-enat* (*epred* (*llength ys*)) + (*the-enat* (*epred*(*llength zs*)))) <
llength (*lfuse ys* (*lmap* (*λx*::*nat. x* + *cp*) *zs*))
using *83 84 86 87* **by** *presburger*
have *11*: *enat* (*the-enat* (*epred* (*llength ys*)) + *i*) < *llength* (*lfuse ys* (*lmap* (*λx*::*nat. x* + *cp*) *zs*))
using *81 82 order-le-less-trans* **by** *blast*
have *12*: (*the-enat* (*epred* (*llength ys*)) + *i* − *the-enat* (*epred* (*llength ys*))) = *i*
by *auto*
have *4*: *lnth* (*lfuse ys* (*lmap* (*λx. x+cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) =
(*lnth* (*lmap* (*λx. x+cp*) *zs*) *i*)
by (*simp add*: *1 10 11 assms*(*13*) *assms*(*9*) *lfuse-lnth-c*)
have *5*: (*lnth* (*lmap* (*λx. x+cp*) *zs*) *i*) = *cp* + (*lnth zs i*)
by (*simp add*: *assms*)
show *?thesis*
using *4 5* **by** *presburger*

qed

lemma *lidx-lfuse-lnth-cp*:
assumes *lidx ys*
(*lnth ys 0*) = *0*
lidx zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite zs
lfinite xs
¬ *lnull xs*
¬ *lnull zs*
¬ *lnull ys*
llast ys = *cp*
llast zs = *the-enat*(*epred* (*llength xs*)) − *cp*
i < (*llength zs*)
cp < *llength xs*
shows *lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) = *cp* + (*lnth zs i*)
using *assms ridx-lidx ridx-lfuse-lnth-cp* **by** *blast*

lemma *ridx-lfuse-lnth-cp-a*:
assumes *ridx R ys*
(*lnth ys 0*) = *0*
ridx R zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite zs
lfinite xs
¬ *lnull xs*
¬ *lnull zs*
¬ *lnull ys*
llast ys = *cp*
llast zs = *the-enat*(*epred* (*llength xs*)) − *cp*
i < (*epred*(*llength ys*)) + (*llength zs*)
the-enat(*epred*(*llength ys*)) ≤ *i*
cp < *llength xs*
shows *lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) *i* = *cp* + (*lnth zs* (*i* −*the-enat*(*epred*(*llength ys*))))
proof −
have *1*: *i* = (*the-enat* (*epred* (*llength ys*)) + (*i* − *the-enat* (*epred* (*llength ys*))))
using *assms* **by** *fastforce*
have *2*: *enat* (*i* − *the-enat* (*epred* (*llength ys*))) < *llength zs*
using *assms*
by (*metis 1 enat-add-mono epred-enat lfinite-llength-enat plus-enat-simps*(*1*) *the-enat.simps*)
show *?thesis*
using *1 2 assms ridx-lfuse-lnth-cp*[*of R ys zs xs cp* (*i* −*the-enat*(*epred*(*llength ys*)))]
by *presburger*
qed

lemma *lidx-lfuse-lnth-cp-a*:
assumes *lidx ys*

(*lnth ys 0*) = *0*
lidx zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite zs
lfinite xs
¬ *lnull xs*
¬ *lnull zs*
¬ *lnull ys*
llast ys = *cp*
llast zs = *the-enat*(*epred* (*llength xs*)) − *cp*
i < (*epred*(*llength ys*)) + (*llength zs*)
the-enat(*epred*(*llength ys*)) ≤ *i*
cp < *llength xs*
shows *lnth* (*lfuse ys* (*lmap* (*λx. x*+*cp*) *zs*)) (*i*) = *cp* + (*lnth zs* (*i* −*the-enat*(*epred*(*llength ys*))))
using *assms ridx-lidx ridx-lfuse-lnth-cp-a* **by** *blast*

lemma *ridx-lfuse-lnth-cp-llast*:
assumes *ridx R ys*
(*lnth ys 0*) = *0*
ridx R zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite zs
lfinite xs
¬ *lnull xs*
¬ *lnull zs*
¬ *lnull ys*
llast ys = *cp*
llast zs = *the-enat*(*epred* (*llength xs*)) − *cp*
i < (*llength zs*)
cp < *llength xs*
shows *llast* (*lfuse ys* (*lmap* (*λx. x*+*cp*) *zs*)) = (*the-enat* (*epred*(*llength xs*)))
proof −
have *1*: *llast* (*lfuse ys* (*lmap* (*λx. x*+*cp*) *zs*)) = *llast* (*lmap* (*λx. x*+*cp*) *zs*)
using *assms*
by (*metis add-cancel-left-left lfinite-lmap lfirst-def llast-lfuse llist.map-disc-iff llist.map-sel*(*1*) *lnth-0-conv-lhd*)
have *2*: *llast* (*lmap* (*λx. x*+*cp*) *zs*) = *cp* + (*llast zs*)
by (*simp add*: *assms*(*6*) *assms*(*9*) *llast-lmap*)
have *3*: *cp* + (*llast zs*) = (*the-enat* (*epred*(*llength xs*)))
using *assms*
by (*metis add-diff-inverse-nat co.enat.exhaust-sel enat-ord-simps*(*2*) *enat-the-enat ileI1 ile-eSuc infinity-ileE leD lfinite-conv-llength-enat llength-eq-0*)
show *?thesis* **using** *1 2 3* **by** *auto*
qed

lemma *lidx-lfuse-lnth-cp-llast*:
assumes *lidx ys*
(*lnth ys 0*) = *0*

lidx zs
(*lnth zs 0*) = *0*
lfinite ys
lfinite zs
lfinite xs
¬ *lnull xs*
¬ *lnull zs*
¬ *lnull ys*
llast ys = *cp*
llast zs = *the-enat*(*epred* (*llength xs*)) − *cp*
i < (*llength zs*)
cp < *llength xs*
shows *llast* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) = (*the-enat* (*epred*(*llength xs*)))
using *assms ridx-lidx ridx-lfuse-lnth-cp-llast* **by** *blast*

lemma *ridx-lfuse-lnth-cp-infinite*:
assumes *ridx R ys*
(*lnth ys 0*) = (*0*::*nat*)
ridx R zs
(*lnth zs 0*) = *0*
lfinite ys
¬*lfinite zs*
¬*lfinite xs*
¬ *lnull ys*
llast ys = *cp*
shows *lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) = *cp* + (*lnth zs i*)
proof −
have *1*: *llast ys* = *lfirst*(*lmap* (λ*x*. *x*+*cp*) *zs*)
by (*metis assms*(*4*) *assms*(*6*) *assms*(*9*) *lfirst-def llist.map-sel*(*1*) *lnth-0-conv-lhd lnull-imp-lfinite plus-nat.add-0*)
have *3*: *lfirst*(*lmap* (λ*x*. *x*+*cp*) *zs*) = *cp* + (*lnth zs 0*)
using *1 assms* **by** *auto*
have *8*: *i*≤ *epred*(*llength zs*)
by (*metis assms*(*6*) *enat.simps*(*3*) *ldrop-eq-LNil lfinite-ldrop lfinite-ltl linorder-le-cases llength-ltl*)
have *10*: *epred* (*llength ys*) ≤ *enat* (*the-enat* (*epred* (*llength ys*)) + (*i*::*nat*))
by (*metis add.right-neutral add-le-same-cancel1 assms*(*5*) *assms*(*8*) *co.enat.exhaust-sel enat-ord-simps*(*1*) *enat-the-enat ile-eSuc infinity-ileE le-zero-eq lfinite-llength-enat linorder-le-cases llength-eq-0*)
have *11*: *enat* (*the-enat* (*epred* (*llength ys*)) + *i*) < *llength* (*lfuse ys* (*lmap* (λ*x*::*nat*. *x* + *cp*) *zs*))
using *assms*
by (*metis 1 enat-ile ldrop-lfuse lfinite-conv-llength-enat lfinite-ldrop lfinite-lmap not-le-imp-less*)
have *12*: (*the-enat* (*epred* (*llength ys*)) + *i* − *the-enat* (*epred* (*llength ys*))) = *i*
by *auto*
have *4*: *lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+*cp*) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) =
(*lnth* (*lmap* (λ*x*. *x*+*cp*) *zs*) *i*)
using *assms*
by (*metis 1 10 11 12 lfuse-lnth-c llist.map-disc-iff lnull-imp-lfinite*)
have *5*: (*lnth* (*lmap* (λ*x*. *x*+*cp*) *zs*) *i*) = *cp* + (*lnth zs i*)
using *assms*

by (*metis 8 add.commute co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-lmap lnull-imp-lfinite*)
show *?thesis*
using *4 5* **by** *presburger*
qed

lemma *lidx-lfuse-lnth-cp-infinite*:
assumes *lidx ys*
$(lnth\ ys\ 0\ ) = 0$
lidx zs
$(lnth\ zs\ 0\ ) = 0$
lfinite ys
$\neg$*lfinite zs*
$\neg$*lfinite xs*
$\neg$ *lnull ys*
llast ys = *cp*
shows *lnth* (*lfuse ys* (*lmap* ($\lambda x.\ x{+}cp$) *zs*)) (*the-enat*(*epred*(*llength ys*)) + *i*) = *cp* + (*lnth zs i*)
using *assms ridx-lidx ridx-lfuse-lnth-cp-infinite* **by** *blast*

lemma *lidx-lfuse-lidx*:
assumes *lidx ys*
lnth ys 0 = *0*
lidx zs
lnth zs 0 = *0*
lfinite ys
$\neg$ *lnull ys*
llast ys = *cp*
lfinite zs
$\neg$ *lnull zs*
lfinite xs
llast zs = *the-enat*(*epred*(*llength xs*)) $-$*cp*
cp < *llength xs*
i < *llength zs*
cp < *llength xs*
shows *lidx* (*lfuse ys* (*lmap* ($\lambda x.\ x{+}\ cp$) *zs*)) $\wedge$ (*lnth* (*lfuse ys* (*lmap* ($\lambda x.\ x{+}\ cp$) *zs*)) *0*) = *0*
proof $-$
have *1*: *llast ys* = *lfirst*(*lmap* ($\lambda x.\ x{+}\ cp$) *zs*)
by (*metis assms*(*4*) *assms*(*7*) *assms*(*9*) *cancel-comm-monoid-add-class.diff-cancel diff-add dual-order.refl lfirst-def llist.map-sel*(*1*) *lnth-0-conv-lhd*)
have *2*: *lfirst* (*lfuse ys* (*lmap* ($\lambda x.\ x{+}\ cp$) *zs*)) = *lfirst ys*
by (*simp add*: *assms*(*6*) *assms*(*9*) *lfirst-lfuse-1*)
have *3*: *llength* (*lfuse ys* (*lmap* ($\lambda x.\ x{+}\ cp$) *zs*)) = *llength ys* + *epred*(*llength zs*)
by (*simp add*: *assms*(*13*) *assms*(*6*) *lfuse-llength*)
have *4*: $\bigwedge j.\ j{<}llength\ ys \Longrightarrow$ *lnth* (*lfuse ys* (*lmap* ($\lambda x.\ x{+}\ cp$) *zs*)) *j*= *lnth ys j*
using *lfuse-lnth-a* **by** *blast*
have *40*: $\exists$ *k1*. *llength ys* = (*enat k1*)
by (*simp add*: *assms*(*5*) *lfinite-llength-enat*)
obtain *k1* **where** *41*: *llength ys* = (*enat k1*)
using *40* **by** *blast*
have *42*: (*enat* (*k1* $-$*1*)) = *epred*(*llength ys*)

using *41 epred-enat* **by** *presburger*
have *43*: *0<k1*
using *assms 41*
by (*metis gr0I llength-eq-0 zero-enat-def*)
have *44*: *the-enat*(*epred*(*llength ys*)) = (*k1* −*1*)
by (*metis 42 the-enat.simps*)
have *5*: $\bigwedge$*j*. *epred*(*llength ys*) ≤*j* ∧ *j* < *epred*(*llength ys*) + *llength zs* ⟹
lnth (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j* =
cp + (*lnth zs* (*j*− *the-enat*(*epred*(*llength ys*))))
using *assms lidx-lfuse-lnth-cp-a*[*of ys zs xs cp*]
by (*metis enat-ord-simps*(*1*) *enat-the-enat gr-implies-not-zero*
infinity-ileE llength-eq-0)
have *45*: $\bigwedge$*j*. *k1*−*1* ≤*j* ∧ *j* < (*enat* (*k1*−*1*)) + *llength zs* ⟹
lnth (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j* =
cp + (*lnth zs* (*j*− (*k1*−*1*)))
using *5 44* **by** (*metis 42 enat-ord-simps*(*1*))
have *50*: *llength*(*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) = *epred*(*llength ys*) + *llength zs*
using *assms*
by (*metis 3 add.commute epred-iadd1 llength-eq-0*)
have *51*: $\bigwedge$*j*. *enat* (*Suc j*) < *llength ys* ⟹
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j*) <
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) (*Suc j*))
by (*metis assms*(*1*) *assms*(*5*) *eSuc-enat iless-Suc-eq lfinite-conv-llength-enat lfuse-lnth-a*
lidx-def not-le-imp-less not-less-iff-gr-or-eq)
have *52*: $\bigwedge$*j*. *k1*−*1* ≤*j* ∧ (*Suc j*) < *enat* (*k1*−*1*) + *llength zs* ⟹
cp + (*lnth zs* (*j*− (*k1*−*1*))) <
cp + (*lnth zs* ((*Suc j*)− (*k1*−*1*)))
using *assms*(*3*) **unfolding** *lidx-def*
by *simp*
(*metis Suc-diff-le add-Suc-right assms*(*8*) *enat-ord-simps*(*2*) *leD lfinite-llength-enat*
nat-add-left-cancel-le not-le-imp-less ordered-cancel-comm-monoid-diff-class.add-diff-inverse
plus-enat-simps(*1*))
have *6*: $\bigwedge$*j*. *enat* (*Suc j*) < *llength*(*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) ⟹
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j*) <
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) (*Suc j*))
proof −
fix *j*
assume *a*: *enat* (*Suc j*) < *llength*(*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*))
show (*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j*) < (*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) (*Suc j*))
proof −
have *61*: *enat* (*Suc j*) < *llength ys* ⟹
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j*) < (*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) (*Suc j*))
using *51* **by** *blast*
have *62*: *k1*−*1* ≤*j* ∧ (*Suc j*) < *llength*(*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) ⟹
(*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) *j*) < (*lnth* (*lfuse ys* (*lmap* (λ*x*. *x*+ *cp*) *zs*)) (*Suc j*))
by (*metis 42 45 50 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le*)
show *?thesis*
by (*metis 41 43 61 62 Suc-diff-1 Suc-mono a enat-ord-simps*(*2*) *leI*)
qed
qed

show *?thesis* **unfolding** *lidx-def*
by (*simp add*: *41 43 6 assms*(*13*) *assms*(*2*) *lfuse-lnth-a*)
qed

lemma *lidx-lfuse-lidx-infinite*:
assumes *lidx ys*
lnth ys 0 = 0
lidx zs
lnth zs 0 = 0
lfinite ys
¬ lnull ys
llast ys = cp
¬lfinite zs
¬lfinite xs
shows *lidx* (*lfuse ys* (*lmap* (*λx. x+ cp*) *zs*)) ∧ (*lnth* (*lfuse ys* (*lmap* (*λx. x+ cp*) *zs*)) *0*) = *0*
proof −
have *1*: *llast ys = lfirst*(*lmap* (*λx. x+ cp*) *zs*)
by (*metis assms*(*4*) *assms*(*7*) *assms*(*8*) *lfirst-def llist.map-sel*(*1*) *lnth-0-conv-lhd lnull-imp-lfinite plus-nat.add-0*)
have *2*: *lfirst* (*lfuse ys* (*lmap* (*λx. x+ cp*) *zs*)) = *lfirst ys*
by (*simp add*: *assms*(*6*) *lfirst-lfuse-1*)
have *4*: ⋀*j. j<llength ys* ⟹ *lnth* (*lfuse ys* (*lmap* (*λx. x+ cp*) *zs*)) *j= lnth ys j*
using *lfuse-lnth-a* **by** *blast*
have *40*: ∃ *k1. llength ys* = (*enat k1*)
by (*simp add*: *assms*(*5*) *lfinite-llength-enat*)
obtain *k1* **where** *41*: *llength ys* = (*enat k1*)
using *40* **by** *blast*
have *42*: (*enat* (*k1 −1*)) = *epred*(*llength ys*)
using *41 epred-enat* **by** *presburger*
have *43*: *0<k1*
using *assms 41*
by (*metis gr0I llength-eq-0 zero-enat-def*)
have *44*: *the-enat*(*epred*(*llength ys*)) = (*k1 −1*)
by (*metis 42 the-enat.simps*)
have *5*: ⋀*j. epred*(*llength ys*) ≤*j* ∧ *j* < *epred*(*llength ys*) + *llength zs* ⟹
lnth (*lfuse ys* (*lmap* (*λx. x+ cp*) *zs*)) *j* =
cp + (*lnth zs* (*j*− *the-enat*(*epred*(*llength ys*))))
using *assms lidx-lfuse-lnth-cp-infinite*[*of ys zs xs cp*]
by (*metis 42 44 enat-ord-simps*(*1*) *ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)
have *45*: ⋀*j. k1−1* ≤*j* ∧ *j* < (*enat* (*k1−1*)) + *llength zs* ⟹
lnth (*lfuse ys* (*lmap* (*λx. x+ cp*) *zs*)) *j* =
cp + (*lnth zs* (*j*− (*k1−1*)))
using *5 44* **by** (*metis 42 enat-ord-simps*(*1*))
have *46*: *llength ys* + *epred* (*llength zs*) = *epred* (*llength ys*) + *llength zs*
by (*metis add.commute assms*(*6*) *assms*(*8*) *epred-iadd1 llength-eq-0 lnull-imp-lfinite*)
have *50*: *llength*(*lfuse ys* (*lmap* (*λx. x+ cp*) *zs*)) = *epred*(*llength ys*) + *llength zs*
using *lfuse-llength*[*of ys* (*lmap* (*λx::nat. x* + *cp*) *zs*)]
llength-lmap[*of* (*λx::nat. x* + *cp*) *zs*]
using *46 assms*(*6*) **by** *presburger*
have *51*: ⋀*j. enat* (*Suc j*) < *llength ys* ⟹

($lnth$ ($lfuse$ ys ($lmap$ ($\lambda x.$ $x+$ cp) zs)) j) $<$
($lnth$ ($lfuse$ ys ($lmap$ ($\lambda x.$ $x+$ cp) zs)) (Suc j))
by (*metis assms*(*1*) *assms*(*5*) *eSuc-enat iless-Suc-eq lfinite-conv-llength-enat lfuse-lnth-a lidx-def not-le-imp-less not-less-iff-gr-or-eq*)
have *52*: $\bigwedge j.$ $k1-1 \leq j \wedge (Suc\ j) < enat\ (k1-1) + llength\ zs \Longrightarrow$
$cp + (lnth\ zs\ (j-\ (k1-1))) <$
$cp + (lnth\ zs\ ((Suc\ j)-\ (k1-1)))$
using *assms*(*3*) **unfolding** *lidx-def* **by** *simp*
(*metis Nat.add-diff-assoc assms*(*8*) *enat-ile lfinite-conv-llength-enat not-le-imp-less plus-1-eq-Suc*)
have *53*: $\bigwedge j.$ $k1-1 \leq j \wedge (Suc\ j) < enat\ (k1-1) + llength\ zs \Longrightarrow$
$lnth\ (lfuse\ ys\ (lmap\ (\lambda x.\ x+\ cp)\ zs))\ j <$
$lnth\ (lfuse\ ys\ (lmap\ (\lambda x.\ x+\ cp)\ zs))\ (Suc\ j)$
by (*metis 45 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le*)
have *6*: $\bigwedge j.$ $enat\ (Suc\ j) < llength(lfuse\ ys\ (lmap\ (\lambda x.\ x+\ cp)\ zs)) \Longrightarrow$
$(lnth\ (lfuse\ ys\ (lmap\ (\lambda x.\ x+\ cp)\ zs))\ j) <$
$(lnth\ (lfuse\ ys\ (lmap\ (\lambda x.\ x+\ cp)\ zs))\ (Suc\ j))$
by (*metis 41 42 43 50 51 53 Suc-diff-1 Suc-eq-plus1 add-less-cancel-right enat-ord-simps*(*2*) *leI*)
show *?thesis* **unfolding** *lidx-def*
by (*simp add*: *41 43 6 assms lfuse-lnth-a*)
qed

1.5 lsub

lemma *lsub-eq-lsubc*:
assumes $k \leq n$
$n < llength\ xs$
shows $lsub\ k\ n\ xs = lsubc\ k\ n\ xs$
using *assms*
unfolding *lsub-def lsubc-def*
by (*auto simp add*: *min-def*)
(*metis co.enat.exhaust-sel enat-ord-simps*(*1*) *iless-Suc-eq ldrop-lnull llength-eq-0 order-le-less-trans*)

lemma *lsub-same*:
assumes $(enat\ k) < llength\ xs$
shows $lsub\ k\ k\ xs = (LCons\ (lnth\ xs\ k\ )\ LNil)$
using *assms*
unfolding *lsub-def*
by (*metis LNil-eq-ltake-iff diff-is-0-eq' enat-0-iff*(*1*) *ldrop-enat ldropn-Suc-conv-ldropn ltake-eSuc-LCons order.order-iff-strict*)

lemma *lsubc-same*:
assumes $(enat\ k) < llength\ xs$
shows $lsubc\ k\ k\ xs = (LCons\ (lnth\ xs\ k\ )\ LNil)$
using *assms*
lsub-eq-lsubc[*of k k xs*] *lsub-same*[*of k xs*]
by *fastforce*

lemma *llength-lsub*:
 assumes $k \leq n$
 $n < llength\ xs$
 shows $llength\ (lsub\ k\ n\ xs) = (eSuc\ (n-k))$
proof –
 have *1*: $llength\ (ldrop\ (enat\ k)\ xs) = (llength\ xs - enat\ k)$
 by (*simp add*: *ldrop-enat*)
 have *2*: $min\ (eSuc\ (enat\ (n - k)))\ (llength\ xs - enat\ k) = (eSuc\ (n-k))$
 by (*metis 1 Suc-diff-le assms*(*1*) *assms*(*2*) *eSuc-enat enat-llength-ldropn ileI1 ldrop-enat min.absorb1*)
 have *3*: $llength\ (ltake\ (eSuc\ (enat\ (n - k)))\ (ldrop\ (enat\ k)\ xs)) = (eSuc\ (n-k))$
 by (*simp add*: *1 2*)
 show *?thesis*
 by (*metis 3 idiff-enat-enat lsub-def*)
qed

lemma *llength-lsubc*:
 assumes $k \leq n$
 $n < llength\ xs$
 shows $llength\ (lsubc\ k\ n\ xs) = (eSuc\ (n-k))$
using *assms lsub-eq-lsubc*[*of k n xs*] *llength-lsub*[*of k n xs*] **by** *presburger*

lemma *llength-lsub-a*:
 shows $llength\ (lsub\ k\ n\ xs) =$
 $min\ (eSuc\ (n - k))$
 $(if\ k = \infty\ then\ 0{::}enat\ else\ llength\ xs - k)$
unfolding *lsub-def*
using *llength-ltake*[*of eSuc* (*n−k*) (*ldrop k xs*)] *llength-ldrop*[*of k xs*]
using *idiff-enat-enat* **by** *presburger*

lemma *llength-lsubc-a*:
 shows $llength\ (lsubc\ k\ n\ xs) =$
 $min\ (eSuc\ (n - k))$
 $(if\ min\ k\ (epred\ (llength\ xs)) = \infty$
 $then\ 0{::}enat$
 $else\ llength\ xs - min\ k\ (epred\ (llength\ xs)))$
unfolding *lsubc-def*
using *llength-ldrop*[*of* (*min k* (*epred* (*llength xs*))) *xs*]
using *llength-ltake*[*of* (*eSuc* (*n* − *k*)) (*ldrop* (*min k* (*epred* (*llength xs*))) *xs*)]
using *idiff-enat-enat* **by** *presburger*

lemma *lsub-not-lnull*:
 assumes $k \leq n$
 $n < llength\ xs$
 shows ¬*lnull* (*lsub k n xs*)
using *assms*
by (*metis eSuc-enat idiff-enat-enat llength-LNil llength-lsub llist.collapse*(*1*) *zero-ne-eSuc*)

lemma *lsub-llength-gr-one*:
 assumes $k<n$
 $n < llength\ xs$

shows *1< llength (lsub k n xs)*
using *llength-lsub-a[of k n xs] assms*
by (*auto simp add*: *min-def one-eSuc zero-enat-def llength-lsub*)

lemma *lsub-lfinite*:
assumes $k \leq n$
$n < llength\ xs$
shows *lfinite (lsub k n xs)*
using *assms*
by (*simp add*: *eSuc-enat llength-eq-enat-lfiniteD llength-lsub*)

lemma *lnth-lsub*:
assumes $n < llength\ xs$
$k+j \leq n$
shows *lnth (lsub k n xs) j* = (*lnth xs (k+j)*)
proof −
have *1*: *enat (j::nat) < eSuc (enat ((n::nat) − (k::nat)))*
using *assms(2)* **by** *auto*
have *2*: *lnth (ltake (eSuc (enat (n − k))) (ldrop (enat k) (xs::'a llist))) j* =
lnth (ldrop (enat k) xs) j
using *1 lnth-ltake* **by** *blast*
have *3*: *lnth (ldrop (enat k) xs) j* = (*lnth xs (k+j)*)
by (*metis add.commute assms(1) assms(2) enat-ord-simps(1) ldrop-enat lnth-ldropn order-le-less-trans*)
show *?thesis* **unfolding** *lsub-def*
using *2 3* **by** *presburger*
qed

lemma *lnth-lsub-a*:
assumes $n < llength\ xs$
$m \leq n$
$k \leq m$
shows *lnth (lsub k n xs) (m−k)* = (*lnth xs m*)
using *assms* **by** (*simp add*: *lnth-lsub*)

lemma *ltake-lsub*:
assumes $n < llength\ xs$
$m + k \leq n$
shows *ltake (eSuc m) (lsub k n xs)* = *lsub k (m+k) xs*
proof −
have *1*: *ltake (eSuc m) (ltake (eSuc (n − k)) (ldrop k xs))* =
ltake (eSuc m) (ldrop k xs)
using *ltake-ltake[of (eSuc m) (eSuc (n − k)) (ldrop k xs)]*
using *Nat.le-diff-conv2 assms(2)* **by** *auto*
have *2*: *ltake (eSuc (m + k − k)) (ldrop k xs)* = *ltake (eSuc m) (ldrop k xs)*
by *simp*
show *?thesis*
unfolding *lsub-def* **using** *1 2* **by** *presburger*
qed

lemma *ldrop-lsub*:
assumes $n < llength\ xs$
$m + k \leq\ n$
shows $ldrop\ m\ (lsub\ k\ n\ xs) = lsub\ (m{+}k)\ n\ xs$
proof –
have *1*: $(ltake\ (eSuc\ (n - k))\ (ldrop\ k\ xs)) =$
$ldrop\ k\ (ltake\ (eSuc\ n)\ xs)$
by (*metis Suc-diff-le add-leD2 assms*(*2*) *eSuc-enat idiff-enat-enat ldrop-ltake*)
have *2*: $ldrop\ m\ (ldrop\ k\ (ltake\ (eSuc\ n)\ xs)) =$
$ldrop\ (m\ {+}k)\ (ltake\ (eSuc\ n)\ xs)$
by *simp*
show *?thesis* **unfolding** *lsub-def* **using** *1 2*
by (*simp add*: *Suc-diff-le assms*(*2*) *eSuc-enat ldrop-ltake*)
qed

lemma *ltl-lsub*:
assumes $n < llength\ xs$
$k \leq n$
shows $ltl(lsub\ k\ n\ xs) = (if\ k{=}n\ then\ LNil\ else\ lsub\ k\ (n{-}1)\ (ltl\ xs))$
proof –
have *1*: $(lsub\ k\ n\ xs) = (ltake\ (eSuc\ (n - k))\ (ldrop\ k\ xs))$
unfolding *lsub-def* **by** *simp*
have *2*: $k{=}n \Longrightarrow ?thesis$
by (*simp add*: *assms*(*1*) *lsub-same*)
have *25*: $k \neq n \Longrightarrow (epred\ (eSuc\ (n - k))) = (eSuc\ (enat\ (n - (1{::}nat) - k)))$
by (*metis Suc-diff-1 assms*(*2*) *co.enat.sel*(*2*) *diff-commute eSuc-enat order-neq-le-trans zero-less-diff*)
have *3*: $k \neq n \Longrightarrow ?thesis$
using *assms ltl-ltake*[*of* $(eSuc\ (n - k))\ (ldrop\ k\ xs)$]
ltl-ldrop[*of k xs*]
unfolding *lsub-def*
using *25* **by** *presburger*
show *?thesis*
using *2 3* **by** *blast*
qed

lemma *ltl-ldrop-one*:
$ltl\ xs = ldrop\ 1\ xs$
by (*metis ldrop-0 ldrop-ltl one-eSuc*)

lemma *lappend-lsub-ltl-lsub*:
assumes $k \leq n$
$n \leq m$
$m < llength\ xs$
shows $lappend\ (lsub\ k\ n\ xs)\ (ltl\ (lsub\ n\ m\ xs)) = lsub\ k\ m\ xs$
proof –
have *0*: $(ltl\ (lsub\ n\ m\ xs)) = ldrop\ 1\ (lsub\ n\ m\ xs)$
by (*metis ldrop-0 ldrop-ltl one-eSuc*)
have *1*: $(lsub\ k\ n\ xs) = (ltake\ (eSuc\ (n - k))\ (ldrop\ k\ xs))$
unfolding *lsub-def* **by** *simp*
have *2*: $(lsub\ n\ m\ xs) = (ltake\ (eSuc\ (m - n))\ (ldrop\ n\ xs))$

unfolding *lsub-def* **by** *simp*
have *3*: *lsub k m xs* = (*ltake* (*eSuc* (*m* − *k*)) (*ldrop k xs*))
unfolding *lsub-def* **by** *simp*
have *8*: *ltake* (*eSuc* (*enat* ((*n*::*nat*) − (*k*::*nat*))) + *enat* ((*m*::*nat*) − *n*)) (*ldrop* (*enat k*) (*xs*::′*a llist*)) =
(*ltake* (*eSuc* (*m* − *k*)) (*ldrop k xs*))
by (*simp add*: *assms*(*1*) *assms*(*2*) *eSuc-enat*)
have *9*: (*ltake* (*enat* (*m* − *n*)) (*ldrop* (*eSuc* (*enat* (*n* − *k*))) (*ldrop* (*enat k*) *xs*))) =
ltl(*ltake* (*eSuc* (*m* − *n*)) (*ldrop n xs*))
by (*metis 0 2 add.commute assms*(*1*) *eSuc-minus-1 ldrop-ldrop ldrop-ltake le-add-diff-inverse*
plus-1-eSuc(*1*) *plus-enat-simps*(*1*))
have *10*: *ltake* (*eSuc* (*enat* ((*n*::*nat*) − (*k*::*nat*))) + *enat* ((*m*::*nat*) − *n*)) (*ldrop* (*enat k*) (*xs*::′*a llist*)) =
lappend (*ltake* (*eSuc* (*enat* (*n* − *k*))) (*ldrop* (*enat k*) *xs*))
(*ltake* (*enat* (*m* − *n*)) (*ldrop* (*eSuc* (*enat* (*n* − *k*))) (*ldrop* (*enat k*) *xs*)))
using *ltake-plus-conv-lappend*[*of* (*eSuc* (*n* − *k*)) *m*−*n* (*ldrop k xs*)] **by** *blast*
show *?thesis*
using *1 10 2 3 8 9* **by** *fastforce*
qed

lemma *lsub-lfuse*:
assumes
$k \leq n$
$n \leq m$
$m <$ *llength xs*
shows *lfuse* (*lsub k n xs*) (*lsub n m xs*) = *lsub k m xs*
using *assms*
by (*simp add*: *lappend-lsub-ltl-lsub lfuse-def lsub-not-lnull order-le-less-subst2*)

lemma *llast-lsub*:
assumes *lfinite xs*
¬ *lnull xs*
$k \leq n$
$n <$ *llength xs*
shows *llast* (*lsub k n xs*) = (*lnth xs n*)
using *assms*
using *lnth-lsub*[*of n xs k*]
by (*metis enat-ord-simps*(*1*) *idiff-enat-enat llast-conv-lnth llength-lsub*
not-le-imp-less order-less-irrefl ordered-cancel-comm-monoid-diff-class.add-diff-inverse)

lemma *lfirst-lsub*:
assumes ¬ *lnull xs*
$k \leq n$
$n <$ *llength xs*
shows *lfirst* (*lsub k n xs*) = (*lnth xs k*)
using *assms*
by (*simp add*: *ldrop-enat lfirst-def lhd-ldropn lsub-def order-le-less-subst2*)

lemma *lsub-lfuse-lidx*:
assumes *lidx ls*
lfinite ls

lfinite xs
¬ *lnull xs*
llast ls = *epred*(*llength xs*)
(*Suc i*) < (*llength ls*)
shows *lfuse* (*lsub* (*lnth ls i*) (*lnth ls* (*Suc i*)) *xs*) (*lsub* (*lnth ls* (*Suc i*)) (*llast ls*) *xs*) =
(*lsub* (*lnth ls i*) (*llast ls*) *xs*)
proof −
have *1*: (*lnth ls i*) ≤ (*lnth ls* (*Suc i*))
by (*simp add*: *assms*(*1*) *assms*(*6*) *lidx-less-eq*)
have *2*: *llast ls* = (*lnth ls* (*the-enat*(*epred* (*llength ls*))))
using *llast-conv-lnth*
by (*metis assms*(*2*) *assms*(*6*) *co.enat.collapse enat.simps*(*3*) *enat-ord-simps*(*4*) *enat-the-enat gr-implies-not-zero ile-eSuc lfinite-llength-enat not-less-iff-gr-or-eq order-neq-le-trans*)
have *3*: *1* < *llength ls*
by (*metis assms*(*1*) *assms*(*6*) *enat-ord-simps*(*1*) *gr-implies-not-zero leD le-add1 lidx-expand-1 llength-eq-0 one-enat-def plus-1-eq-Suc*)
have *4*: (*lnth ls* (*Suc i*)) ≤ (*llast ls*)
using *2 assms*(*1*) *assms*(*2*) *assms*(*6*) *lfinite-llength-enat lidx-less-eq* **by** *fastforce*
have *5*: *enat* (*lnth ls* (*Suc i*)) < *llength xs*
by (*metis 4 assms*(*4*) *assms*(*5*) *co.enat.exhaust-sel eSuc-enat enat-ord-simps*(*2*) *le-imp-less-Suc llength-eq-0*)
have *6*: *llast* (*lsub* (*lnth ls i*) (*lnth ls* (*Suc i*)) *xs*) = (*lnth xs* (*lnth ls* (*Suc i*)))
using *llast-lsub*[*of xs* (*lnth ls i*) (*lnth ls* (*Suc i*))]
using *1 5 assms*(*3*) *assms*(*4*) *enat-ord-simps*(*1*) **by** *blast*
have *7*: *lfirst* (*lsub* (*lnth ls* (*Suc i*)) (*llast ls*) *xs*) = (*lnth xs* (*lnth ls* (*Suc i*)))
by (*metis 4 assms*(*4*) *assms*(*5*) *co.enat.exhaust-sel enat-ord-simps*(*1*) *ile-eSuc iless-Suc-eq lfirst-lsub llength-eq-0 order-less-le*)
show *?thesis*
by (*metis 1 4 assms*(*4*) *assms*(*5*) *co.enat.exhaust-sel enat-ord-simps*(*1*) *ile-eSuc iless-Suc-eq llength-eq-0 lsub-lfuse order-neq-le-trans*)
qed

1.6 llastlfirst

lemma *llastlfirst-ridx*:
llastlfirst xss = *ridx* (λ *a b*. *llast a* = *lfirst b*) *xss*
by (*simp add*: *llastlfirst-def ridx-def*)

lemma *llastlfirst-LNil*[*simp*]:
llastlfirst LNil
unfolding *llastlfirst-def* **by** *simp*

lemma *llastlfirst-LOne*[*simp*]:
llastlfirst (*LCons xs LNil*)
unfolding *llastlfirst-def* **by** (*simp add*: *zero-enat-def*)

lemma *llastlfirst-LCons*[*simp*]:
assumes ¬*lnull xss*
shows *llastlfirst* (*LCons xs xss*) ⟷ *llast xs* = *lfirst* (*lfirst xss*) ∧ *llastlfirst xss*
using *assms* **unfolding** *llastlfirst-def*

by *auto*
  (*metis One-nat-def lfirst-def lhd-LCons-ltl lnth-LCons′ not-lnull-llength one-enat-def*,
   metis Suc-ile-eq lnth-Suc-LCons,
   metis One-nat-def Suc-diff-le Suc-ile-eq add-diff-cancel-left′ le-Suc-eq le-add1 lfirst-def llist.disc(*2*) *lnth-0 lnth-0-conv-lhd lnth-ltl ltl-simps*(*2*) *plus-1-eq-Suc*)

lemma *llastlfirst-lappend-lfinite*:
assumes *lfinite xss*
shows *llastlfirst* (*lappend xss yss*) ⟷
  (*llastlfirst xss* ∧ *llastlfirst yss* ∧
  (*if lnull xss* ∨ *lnull yss then True else llast*(*llast xss*) = *lfirst*(*lfirst yss*)))
using *assms*
by (*metis* (*mono-tags*, *lifting*) *lfirst-def llastlfirst-ridx ridx-lappend-lfinite*)

1.7 lfusecat

context notes [[*function-internals*]]
begin

partial-function (*llist*) *lfusecat* :: ′*a llist llist* ⇒ ′*a llist*
where *lfusecat xss* = (*case xss of LNil* ⇒ *LNil* | *LCons xs xss*′ ⇒ *lfuse xs* (*lfusecat xss*′))

end

lemma *lfusecat-simps* [*simp*, *code*]:
 shows *lfusecat-LNil*: *lfusecat LNil* = *LNil*
 and *lfusecat-LCons*: *lfusecat* (*LCons xs xss*) = *lfuse xs* (*lfusecat xss*)
by (*simp-all add*: *lfusecat.simps*)

declare *lfusecat.mono*[*cont-intro*]

lemma *mono2mono-lfusecat*[*THEN llist.mono2mono*, *cont-intro*, *simp*]:
 shows *monotone-lfusecat*: *monotone* (*lprefix*) (*lprefix*) *lfusecat*
by(*rule llist.fixp-preserves-mono1*[*OF lfusecat.mono lfusecat-def*]) *simp*

lemma *mcont2mcont-lfusecat*[*THEN llist.mcont2mcont*, *cont-intro*, *simp*]:
 shows *mcont-lfusecat*: *mcont lSup* (*lprefix*) *lSup* (*lprefix*) *lfusecat*
by (*rule llist.fixp-preserves-mcont1*[*OF lfusecat.mono lfusecat-def*])
  simp

lemmas *lfusecat-fixp-parallel-induct* =
 parallel-fixp-induct-1-1[*OF llist-partial-function-definitions llist-partial-function-definitions lfusecat.mono lfusecat.mono lfusecat-def lfusecat-def*, *case-names adm LNil step*]

lemma *llist-all2-lfusecatI*:
 llist-all2 (*llist-all2 A*) *xss yss*
 ⟹ *llist-all2 A* (*lfusecat xss*) (*lfusecat yss*)
proof(*induct arbitrary*: *xss yss rule*: *lfusecat-fixp-parallel-induct*)

case *adm*
then show *?case* **by** (*auto split*: *llist.split intro*: *llist-all2-lappendI*)
next
case *LNil*
then show *?case* **by** (*auto split*: *llist.split intro*: *llist-all2-lappendI*)
next
case (*step f g*)
then show *?case*
using *llist-all2-lfuseI* **by** (*auto split*: *llist.split intro*: *llist-all2-lappendI*) *blast*
qed

lemma *not-lnull-lset-conv-a*:
  $\neg$ *lnull xss* $\wedge$ ($\forall$ *xs* $\in$ *lset xss*. $\neg$ *lnull xs*) $\longleftrightarrow$ *lset xss* $\neq$ {} $\wedge$ *LNil* $\notin$ *lset xss*
using *llist.collapse*(*1*) *llist.disc*(*1*) *lset-eq-empty* **by** *blast*

lemma *not-lnull-lset-conv-b*:
  $\neg$ *lnull xss* $\wedge$ ($\forall$ *xs* $\in$ *lset xss*. $\neg$ *lnull xs*) $\longleftrightarrow$ *lset xss* $\neq$ {} $\wedge$ *lset xss* $\cap$ {*LNil*} = {}
using *llist.collapse*(*1*) **by** *force*

lemma *lfusecat-eq-LNil*:
lfusecat xss = *LNil* $\longleftrightarrow$ *lset xss* $\subseteq$ {*LNil*}
proof (*induction xss*)
case *adm*
then show *?case*
  by *simp*
next
case *LNil*
then show *?case*
by *simp*
next
case (*LCons xs xss*)
then show *?case*
by *simp*
   (*metis* (*no-types*, *lifting*) *lfuse-conv-lnull* *llist.disc*(*1*) *llist.expand*)
qed

lemma *lfusecat-lfilter-neq-LNil*:
  lfusecat (*lfilter* (λ*xs*. $\neg$ *lnull xs*) *xss*) = *lfusecat xss*
proof (*induct xss*)
case *adm*
then show *?case* **by** *simp*
next
case *LNil*
then show *?case* **by** *simp*
next
case (*LCons xs xss*)
then show *?case* **by** (*simp add*: *lappend-lnull1 lfuse-def*)
qed

lemma *lprefix-lfusecatI*:
lprefix xss yss $\Longrightarrow$ *lprefix* (*lfusecat xss*) (*lfusecat yss*)
by (*meson mcont-lfusecat mcont-monoD*)

lemma *lset-lltl-llength*:
($\forall$ *xs* $\in$ *lset xss*. *llength xs* $\leq$ *1*) $\longleftrightarrow$ ($\forall$ *xs* $\in$ *lset* (*lltl xss*). *lnull xs*)
unfolding *lltl-def*
by (*auto simp add*: *ltl-ldrop-one*)

lemma *lset-lltl-llength-var*:
($\forall$ *xs* $\in$ *lset xss*. *llength xs* $\leq$ *1*) $\longleftrightarrow$ *lset*(*lltl xss*) $\subseteq$ {*LNil*}
using *lset-lltl-llength*
by (*metis all-not-in-conv insert-iff lnull-def subsetI subset-singletonD*)

lemma *lset-lltl-llength-var2*:
($\forall$ *xs* $\in$ *lset xss*. *llength xs* $>$ *1*) $\Longrightarrow$ *lset* (*lltl xss*) $\cap$ {*LNil*} = {}
unfolding *lltl-def*
by *auto*
(*metis co.enat.exhaust-sel epred-llength less-numeral-extra*(*4*) *llength-LNil not-one-less-zero one-eSuc*)

lemma *lfusecat-all-empty-or-LNil-a*:
assumes *lset*(*lltl xss*) $\subseteq$ {*LNil*}
shows *llength* (*lfusecat xss*) $\leq$ *1*
using *assms*
proof (*induction xss*)
case *adm*
then show *?case* **unfolding** *lltl-def* **by** *simp*
next
case *LNil*
then show *?case* **by** *simp*
next
case (*LCons xs xss*)
then show *?case*
unfolding *lltl-def*
by *simp*
(*metis LCons.prems lfuse-llength-atmost-one llist.set-intros*(*1*) *lset-lltl-llength-var*)
qed

lemma *lfusecat-all-empty-or-LNil-b*:
assumes *llength* (*lfusecat xss*) $\leq$ *1*
shows *lset*(*lltl xss*) $\subseteq$ {*LNil*}
using *assms*
proof (*induction xss*)
case *adm*
then show *?case* **unfolding** *lltl-def* **by** *simp*
next

case *LNil*
then show *?case* **unfolding** *lltl-def* **by** *simp*
next
case (*LCons xs xss*)
then show *?case*
unfolding *lltl-def*
by (*simp add*: *lfuse-llength-atmost-one ltl-ldrop-one*)
qed

lemma *lfusecat-all-empty-or-LNil*:
shows $llength\ (lfusecat\ xss) \leq 1 \longleftrightarrow lset(lltl\ xss) \subseteq \{LNil\}$
using *lfusecat-all-empty-or-LNil-a lfusecat-all-empty-or-LNil-b* **by** *blast*

lemma *lfusecat-not-lnull*:
$\neg lnull\ (lfusecat\ xss) \longleftrightarrow \neg lnull\ xss \wedge (\exists\, xs \in lset\ (\ xss).\ \neg lnull(\ xs))$
by (*metis lconcat-eq-LNil lfusecat-LNil lfusecat-eq-LNil lnull-def lnull-lconcat mem-Collect-eq subsetD subsetI*)

lemma *lset-eq-forall-lnull*:
$lnull\ xss \vee (\forall\, xs \in lset\ (\ xss).\ lnull(\ xs)) \longleftrightarrow lset\ xss \subseteq \{LNil\}$
by (*auto simp add*: *lset-lnull*)

lemma *lfusecat-not-lnull-var*:
assumes $\neg lnull\ xss$
$\forall\, xs \in lset\ xss.\ \neg lnull\ xs$
shows $\neg lnull(lfusecat\ xss)$
using *assms*
using *lfusecat-not-lnull llist.set-sel*(*1*) **by** *blast*

lemma *lfirst-lfusecat-lfirst*:
assumes $\neg lnull\ xss$
$\neg lnull\ (lfirst\ xss)$
shows $lfirst(lfusecat\ xss) = lfirst(lfirst\ xss)$
proof (*cases xss*)
case *LNil*
then show *?thesis*
using *assms*(*1*) *llist.disc*(*1*) **by** *blast*
next
case (*LCons x21 x22*)
then show *?thesis*
by (*metis assms*(*2*) *eq-LConsD lfirst-def lfirst-lfuse-1 lfusecat-LCons*)
qed

lemma *lfirst-lfusecat*:
assumes $\neg\ lnull\ xs$
shows $lfirst(lfusecat\ (LCons\ xs\ xss)) = lfirst\ xs$
using *assms* **by** (*simp add*: *lfirst-def*)

lemma *ltl-lfusecat* :
assumes $\neg$*lnull xss*
$\neg$*lnull* (*lfirst xss*)
shows *ltl*(*lfusecat xss*) = *lappend* (*ltl* (*lhd xss*)) (*ltl* (*lfusecat* (*ltl xss*)))
using *assms*
unfolding *lfirst-def*
by (*metis lappend-lnull2 lfusecat-LCons lhd-LCons-ltl ltl-lfuse*)

lemma *lfusecat-lappend*:
assumes *lfinite xss*
shows *lfusecat* (*lappend xss yss*) = *lfuse* (*lfusecat xss*) (*lfusecat yss*)
using *assms*
proof (*induct xss arbitrary*: *yss*)
case *lfinite-LNil*
then show *?case* **by** *auto*
next
case (*lfinite-LConsI xss xs*)
then show *?case*
proof −
have *1*: *lfusecat* (*lappend* (*LCons xs xss*) *yss*) = *lfuse xs* (*lfusecat* (*lappend xss yss*))
by *simp*
have *2*: *lfinite xss*
by (*simp add*: *lfinite-LConsI.hyps*(*1*))
have *3*: (*lfusecat* (*lappend xss yss*)) = *lfuse* (*lfusecat xss*) (*lfusecat yss*)
by (*simp add*: *lfinite-LConsI.hyps*(*2*))
have *4*: *lfuse xs* (*lfuse* (*lfusecat xss*) (*lfusecat yss*)) =
lfuse (*lfusecat* (*LCons xs xss*)) (*lfusecat yss*)
by (*metis lappend-lnull1 lfuse-LNil-2 lfuse-assoc lfuse-def lfusecat-LCons*)
show *?thesis*
by (*simp add*: *3 4*)
qed
qed

lemma *lfusecat-split*:
shows *lfusecat xss* = *lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))
proof −
have *1*: *xss* = *lappend* (*ltake n xss*) (*ldrop n xss*)
by (*simp add*: *lappend-ltake-ldrop*)
show *?thesis* **using** *1 lfusecat-lappend*[*of* (*ltake n xss*) (*ldrop n xss*)] **by** *force*
qed

lemma *lfuse-lfinite*:
shows *lfinite* (*lfuse xs ys*) $\longleftrightarrow$ *lfinite xs* $\wedge$ *lfinite ys*
by (*metis lfinite-lappend lfinite-ltl lnull-imp-lfinite ltl-lfuse*)

lemma *lfusecat-lfinite-a*:
assumes *lfinite xss*
$\forall$ *xs* $\in$ *lset xss*. *lfinite xs*
shows *lfinite* (*lfusecat xss*)

using *assms*
proof (*induct xss*)
case *lfinite-LNil*
then show *?case* **by** *auto*
next
case (*lfinite-LConsI xss xs*)
then show *?case*
  proof (*cases xss=LNil*)
  case *True*
  then show *?thesis* **by** (*simp add*: *lfinite-LConsI.prems*)
  next
  case *False*
  then show *?thesis*
      proof −
    have *1*: *lfinite xs*
      by (*simp add*: *lfinite-LConsI.prems*)
    have *5*: ∀ *xs* ∈ *lset xss*. *lfinite xs*
     by (*simp add*: *lfinite-LConsI.prems*)
    have *6*: *lfinite* (*lfusecat xss*)
      using *5 lfinite-LConsI.hyps*(*2*) **by** *blast*
    have *7*: *lfinite* (*lfuse xs* (*lfusecat xss*))
      by (*simp add*: *1 6 lfuse-lfinite*)
    show *?thesis* **by** (*simp add*: *7*)
  qed
 qed
qed

lemma *lfusecat-repeat-LNil* [*simp*]:
  lfusecat (*repeat LNil*) = *LNil*
by (*simp add*: *lfusecat-eq-LNil*)

lemma *llastfirst-lfusecat-llast*:
 assumes *lfinite xss*
         ¬*lnull xss*
         ∀ *xs* ∈ *lset xss*. ¬*lnull xs*
         llastlfirst xss
         ∀ *xs* ∈ *lset xss*. *lfinite xs*
 shows   *llast*(*lfusecat xss*) = *llast*(*llast xss*)
using *assms*
proof (*induction xss*)
case *lfinite-LNil*
then show *?case*
using *llist.disc*(*1*) **by** *blast*
next
case (*lfinite-LConsI xss xs*)
then show *?case*
  proof (*cases xss* = *LNil*)
  case *True*
  then show *?thesis*
  by *simp*

next
case *False*
then show *?thesis*
proof −
have *1*: *llastlfirst xss*
using *False lfinite-LConsI.prems*(*3*) *llastlfirst-LCons llist.collapse*(*1*) **by** *blast*
have *2*: ∀ *xs* ∈ *lset xss. lfinite xs*
by (*simp add*: *lfinite-LConsI.prems*(*4*))
have *3*: ∀ *xs* ∈ *lset xss.* ¬*lnull xs*
using *lfinite-LConsI.prems*(*2*) **by** *auto*
have *4*: *lfinite xss*
by (*simp add*: *lfinite-LConsI.hyps*)
have *5*: *llast* (*lfusecat xss*) = *llast* (*llast xss*)
using *1 2 3 False lfinite-LConsI.IH llist.collapse*(*1*) **by** *blast*
have *6*: *llast* (*llast* (*LCons xs xss*)) = *llast* (*llast xss*)
by (*metis False llast-LCons llist.collapse*(*1*))
have *7*: (*lfusecat* (*LCons xs xss*)) = *lfuse xs* (*lfusecat xss*)
by *simp*
have *8*: ¬ *lnull xs*
by (*simp add*: *lfinite-LConsI.prems*(*2*))
have *9*: *lfinite xs*
by (*simp add*: *lfinite-LConsI.prems*(*4*))
have *10*: ¬ *lnull* (*lfusecat xss*)
using *3 False lfusecat-not-lnull-var llist.collapse*(*1*) **by** *blast*
have *11*: *lfinite* (*lfusecat xss*)
using *2 4 lfusecat-lfinite-a* **by** *blast*
have *111*: *lfirst* (*lfusecat xss*) = *lfirst* (*lfirst xss*)
by (*metis 3 False lfirst-def lfirst-lfusecat-lfirst llist.collapse*(*1*) *llist.set-sel*(*1*))
have *12*: *llast xs* = *lfirst* (*lfusecat xss*)
using *10 111 lfinite-LConsI.prems*(*3*) *lfusecat-not-lnull llastlfirst-LCons* **by** *auto*
have *13*: *llast* (*lfuse xs* (*lfusecat xss*)) = *llast* (*lfusecat xss*)
using *llast-lfuse*[*of xs* (*lfusecat xss*)] *8 9 10 11 12* **by** *blast*
have *14*: *llast* (*lfusecat* (*LCons xs xss*)) = *llast* (*lfusecat xss*)
by (*simp add*: *13*)
show *?thesis*
using *13 5 6* **by** *auto*
qed
qed
qed

lemma *lfusecat-llength-lNil*:
llength (*lfusecat LNil*) = *0*
by *simp*

lemma *lfusecat-llength-lfinite*:
assumes *lfinite xss*
¬*lnull xss*
∀ *xs* ∈ *lset xss.* ¬ *lnull xs*
∀ *xs* ∈ *lset xss. lfinite xs*

llastlfirst xss
shows *llength*(*lfusecat xss*) =
eSuc($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*)))
using *assms*
proof (*induction xss*)
case *lfinite-LNil*
then show *?case* **by** *simp*
next
case (*lfinite-LConsI xss xs*)
then show *?case*
proof (*cases xss* = *LNil*)
case *True*
then show *?thesis*
proof −
have *1*: *lfusecat* (*LCons xs xss*) = *xs*
by (*simp add*: *True*)
have *2*: *llength* (*lfusecat* (*LCons xs xss*)) = *llength xs*
using *1* **by** *auto*
have *4*: ($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength*(*LCons xs xss*)))).
epred (*llength* (*lnth* (*LCons xs xss*) *i*))) =
epred(*llength* (*lnth* (*LCons xs xss*) *0*))
using *True* **by** *simp*
have *5*: *eSuc*(*epred*(*llength* (*lnth* (*LCons xs xss*) *0*))) = *llength xs*
by (*simp add*: *lfinite-LConsI.prems*(*2*))
show *?thesis*
using *2 4 5* **by** *presburger*
qed
next
case *False*
then show *?thesis*
proof −
have *6*: (*lfusecat* (*LCons xs xss*)) = *lfuse xs* (*lfusecat xss*)
by *simp*
have *7*: *llastlfirst* (*LCons xs xss*)
by (*simp add*: *lfinite-LConsI.prems*(*4*))
have *71*: *lfirst* (*lfusecat xss*) = *lfirst*(*lfirst xss*)
by (*metis False insert-iff lfinite-LConsI.prems*(*2*) *lfirst-def lfirst-lfusecat-lfirst*
llist.collapse(*1*) *llist.set-sel*(*1*) *llist.simps*(*19*))
have *8*: *llast xs* = *lfirst* (*lfusecat xss*)
using *7 71 False llastlfirst-LCons llist.collapse*(*1*) **by** *auto*
have *9*: *llength* (*lfuse xs* (*lfusecat xss*)) = *llength xs* + *epred*(*llength*(*lfusecat xss*))
by (*simp add*: *lfinite-LConsI.prems*(*2*) *lfuse-llength*)
have *10*: (*llength*(*lfusecat xss*)) =
eSuc($\sum$ *i* = *0* .. (*the-enat*(*epred* (*llength xss*))). *epred*(*llength* (*lnth xss i*)))
using *7 False lfinite-LConsI.IH lfinite-LConsI.prems*(*2*) *lfinite-LConsI.prems*(*3*) *llastlfirst-LCons*
llist.collapse(*1*) **by** *auto*
have *11*: (*the-enat*(*epred*(*llength* (*LCons xs xss*)))) = (*1*+*the-enat*(*epred*(*llength xss*)))
using *False*
by *simp*
(*metis False co.enat.sel*(*2*) *eSuc-enat lfinite.cases lfinite-LConsI.hyps lfinite-llength-enat*

llength-LCons the-enat.simps)
have *12*: ($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength* (*LCons xs xss*)))). *epred* (*llength* (*lnth* (*LCons xs xss*) *i*))) =
epred(*llength*(*lnth* (*LCons xs xss*) *0*)) +
($\sum$ *i* = *1* ..(*1*+*the-enat*(*epred*(*llength xss*))). *epred* (*llength* (*lnth* (*LCons xs xss*) *i*)))
using *11*
by (*simp add*: *sum.atLeast-Suc-atMost*)
have *13*: *epred*(*llength*(*lnth* (*LCons xs xss*) *0*)) = *epred* (*llength xs*)
by *simp*
have *14*: ($\sum$ *i* = *1* ..(*1*+*the-enat*(*epred*(*llength xss*))). *epred* (*llength* (*lnth* (*LCons xs xss*) *i*))) =
($\sum$ *i* = *0* ..(*the-enat*(*epred*(*llength xss*))). *epred* (*llength* (*lnth* (*LCons xs xss*) (*Suc i*))))
using *sum.shift-bounds-cl-nat-ivl*[*of* λ*k*. *epred* (*llength* (*lnth* (*LCons xs xss*) *k*)) *0 1*
(*the-enat*(*epred*(*llength xss*)))]
by *simp*
have *15*: ($\sum$ *i* = *0* ..(*the-enat*(*epred*(*llength xss*))). *epred* (*llength* (*lnth* (*LCons xs xss*) (*Suc i*)))) =
($\sum$ *i* = *0* ..(*the-enat*(*epred*(*llength xss*))). *epred* (*llength* (*lnth* (*xss*) (*i*))))
by *auto*
have *16*: *epred* (*llength xs*) + ($\sum$ *i* = *0* ..(*the-enat*(*epred*(*llength xss*))). *epred* (*llength* (*lnth* (*xss*) (*i*)))) =
($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength* (*LCons xs xss*)))). *epred* (*llength* (*lnth* (*LCons xs xss*) *i*)))
using *12 13 14 15* **by** *presburger*
have *17*: *llength xs* + *epred*(*llength*(*lfusecat xss*)) =
eSuc(*epred* (*llength xs*) + ($\sum$ *i* = *0* ..(*the-enat*(*epred*(*llength xss*))). *epred* (*llength* (*lnth* (*xss*) (*i*)))))
by (*metis 10 eSuc-epred eSuc-plus epred-eSuc lfinite-LConsI.prems*(*2*) *llength-eq-0 lset-intros*(*1*))
show *?thesis*
using *16 17 6 9* **by** *presburger*
qed
qed
qed

lemma *llastlfirst-ltake*:
assumes *n*≤*llength xss*
llastlfirst xss
shows *llastlfirst* (*ltake* (*enat n*) *xss*)
using *assms*
proof (*induction n arbitrary*: *xss*)
case *0*
then show *?case*
by (*simp add*: *llastlfirst-def*)
next
case (*Suc n*)
then show *?case*
unfolding *llastlfirst-def*
by *auto*
(*metis Suc-ile-eq enat-ord-simps*(*2*) *less-Suc-eq lnth-ltake order-le-less-trans*)
qed

lemma *lfusecat-ltake*:
assumes¬ *lnull xss*

n ≤ *llength xss*
∀ *xs* ∈ *lset xss*. ¬ *lnull xs*
∀ *xs* ∈ *lset xss*. *lfinite xs*
llastlfirst xss
shows *lfusecat* (*ltake* (*enat n*) *xss*) =
ltake (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)
proof −
have *1*: *lfinite* (*ltake* (*enat n*) *xss*)
using *enat-ord-code*(*4*) *lfinite-ltake* **by** *blast*
have *2*: *llength* (*ltake* (*enat n*) *xss*) = *n*
by (*simp add*: *assms*(*2*))
have *3*: (*the-enat*(*epred*(*llength* (*ltake* (*enat n*) *xss*)))) = *n*−*1*
using *2 epred-enat the-enat.simps* **by** *presburger*
have *4*: ∀ *xs* ∈ *lset* (*ltake* (*enat n*) *xss*). ¬ *lnull xs*
by (*meson assms*(*3*) *lset-ltake subsetD*)
have *5*: ∀ *xs* ∈ *lset* (*ltake* (*enat n*) *xss*). *lfinite xs*
by (*meson assms*(*4*) *lset-ltake subsetD*)
have *6*: *llastlfirst* (*ltake* (*enat n*) *xss*)
by (*simp add*: *assms*(*2*) *assms*(*5*) *llastlfirst-ltake*)
have *7*: ⋀ *j*. *0*< *n* ∧ *j* < *n*−*1* ⟶
epred(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *j*)) =
epred(*llength* (*lnth xss j*))
by (*metis diff-le-self dual-order.strict-trans1 enat-ord-simps*(*2*) *lnth-ltake*)
have *71*: *llength* (*lfusecat* (*ltake* (*enat n*) *xss*)) =
(*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *i*))))
using *lfusecat-llength-lfinite*[*of* (*ltake* (*enat n*) *xss*)]
by (*metis 1 2 3 4 5 6 lfusecat-LNil llength-LNil llist.collapse*(*1*) *ltake-0*
the-enat.simps zero-enat-def)
have *8*: (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *i*)))) =
(*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
using *7 atLeastAtMost-iff*[*of* - *0 n*−*1*] *sum.cong*[*of* {*0*..*n*−*1*}
{*0*..*n*−*1*} λ*i*. *epred*(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *i*))
λ*i*. *epred*(*llength* (*lnth xss i*))]
by (*simp add*: *Suc-ile-eq dual-order.refl lnth-ltake*)
have *9*: *llength* (*lfusecat* (*ltake* (*enat n*) *xss*)) ≤ *llength* (*lfusecat xss*)
by (*metis lfuse-llength-le-a lfusecat-split*)
have *10*: *llength* (*ltake* (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) =
(*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
using *71 8 9* **by** *auto*
have *11*: *ltake* (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))) =
(*lfusecat* (*ltake n xss*))
using *71 8 ltake-lfuse*[*of* (*lfusecat* (*ltake* (*enat n*) *xss*)) (*lfusecat* (*ldrop n xss*))]
by *presburger*
have *12*: (*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))) = *lfusecat xss*
by (*metis lfusecat-split*)
show *?thesis*
using *11 12* **by** *auto*

qed

lemma *lfusecat-ldrop*:
assumes $\neg$ *lnull xss*
$n <$ *llength xss*
$\forall$ *xs* $\in$ *lset xss*. $\neg$ *lnull xs*
$\forall$ *xs* $\in$ *lset xss*. *lfinite xs*
llastlfirst xss
shows *lfusecat* (*ldrop* (*enat n*) *xss*) =
ldrop (*if n* = *0 then 0 else* ($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)
proof −
have *1*: *lfinite* (*ltake* (*enat n*) *xss*)
using *enat-ord-code*(*4*) *lfinite-ltake* **by** *blast*
have *2*: *llength* (*ltake* (*enat n*) *xss*) = *n*
by (*simp add*: *assms*(*2*))
have *3*: (*the-enat*(*epred*(*llength* (*ltake* (*enat n*) *xss*)))) = *n*−*1*
using *2 epred-enat the-enat.simps* **by** *presburger*
have *4*: $\forall$ *xs* $\in$ *lset* (*ltake* (*enat n*) *xss*). $\neg$ *lnull xs*
by (*meson assms*(*3*) *lset-ltake subsetD*)
have *5*: $\forall$ *xs* $\in$ *lset* (*ltake* (*enat n*) *xss*). *lfinite xs*
by (*meson assms*(*4*) *lset-ltake subsetD*)
have *6*: *llastlfirst* (*ltake* (*enat n*) *xss*)
by (*simp add*: *assms*(*2*) *assms*(*5*) *llastlfirst-ltake order.strict-implies-order*)
have *7*: $\bigwedge$ *j*. *0*< *n* $\wedge$ *j* < *n*−*1* $\longrightarrow$
epred(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *j*)) =
epred(*llength* (*lnth xss j*))
by (*metis diff-le-self dual-order.strict-trans1 enat-ord-simps*(*2*) *lnth-ltake*)
have *71*: *llength* (*lfusecat* (*ltake* (*enat n*) *xss*)) =
(*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *i*))))
using *lfusecat-llength-lfinite*[*of* (*ltake* (*enat n*) *xss*)]
by (*metis 1 2 3 4 5 6 lfusecat-LNil llength-LNil llist.collapse*(*1*) *ltake-0*
the-enat.simps zero-enat-def)
have *8*: (*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *i*)))) =
(*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
using *7 atLeastAtMost-iff*[*of - 0 n*−*1*] *sum.cong*[*of* {*0*..*n*−*1*} {*0*..*n*−*1*}
λ*i*. *epred*(*llength* (*lnth* (*ltake* (*enat n*) *xss*) *i*))
λ*i*. *epred*(*llength* (*lnth xss i*))]
by (*simp add*: *Suc-ile-eq dual-order.refl lnth-ltake*)
have *9*: *llength* (*lfusecat* (*ltake* (*enat n*) *xss*)) $\leq$ *llength* (*lfusecat xss*)
by (*metis lfuse-llength-le-a lfusecat-split*)
have *10*: *llength* (*ltake* (*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) =
(*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
using *71 8 9* **by** *auto*
have *11*: *ltake* (*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))) =
(*lfusecat* (*ltake n xss*))
using *71 8 ltake-lfuse*[*of* (*lfusecat* (*ltake* (*enat n*) *xss*)) (*lfusecat* (*ldrop n xss*))]
by *presburger*

```
 have 12: ldrop 0 (lfuse (lfusecat (ltake 0 xss)) (lfusecat (ldrop 0 xss))) =
          (lfusecat (ldrop 0 xss))
    by simp
 have 13: 0< n ⟹¬ lnull (lfusecat (ltake (enat (n::nat)) (xss::'a llist llist)))
   using 71 8 by force
 have 14: 0< n ⟹ ¬ lnull (lfusecat (ldrop (enat n) xss))
  by (meson assms(2) assms(3) in-lset-ldropD leD lfusecat-not-lnull-var lnull-ldrop)
 have 15: 0<n ⟹ llast (lfusecat (ltake (enat n) xss)) = lfirst (lfusecat (ldrop (enat n) xss))
  proof −
   assume a: 0<n
   have 151: llast (lfusecat (ltake (enat n) xss)) = llast(llast  (ltake (enat n) xss))
     using 1 13 4 5 6 a lfusecat-not-lnull llastfirst-lfusecat-llast by blast
   have 152: lfirst (lfusecat (ldrop (enat n) xss)) = lfirst (lfirst (ldrop (enat n) xss))
      by (metis assms(2) assms(3) in-lset-conv-lnth ldrop-enat leD lfirst-def lfirst-lfusecat-lfirst
         lhd-ldropn lnull-ldrop)
   have 153: llast(llast  (ltake (enat n) xss)) = lfirst (lfirst (ldrop (enat n) xss))
     using assms a
     by (metis 1 13 lappend-ltake-ldrop leD lfusecat-not-lnull llastlfirst-lappend-lfinite
         lnull-ldrop)
   show ?thesis
   by (simp add: 151 152 153)
  qed
 have 16: 0<n ⟹ ldrop (epred (eSuc(∑ i = 0 .. (n−1) . epred(llength (lnth xss i)))))
              (lfuse (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))) =
             (lfusecat (ldrop n xss))
  using 71 8 13 14 15 ldrop-lfuse-a[of (lfusecat (ltake (enat n) xss)) (lfusecat (ldrop n xss))]
  by (metis 1 5 less-numeral-extra(3) lfusecat-lfinite-a)
 have 17: ldrop (if n =0 then 0 else (∑ i = 0 .. (n−1) . epred(llength (lnth xss i))))
              (lfuse (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))) =
             (lfusecat (ldrop n xss))
  using 11 16 by auto
 show ?thesis
 by (metis 17  lfusecat-split)
qed


lemma lfusecat-eq-LCons-conv:
shows  lfusecat xss = LCons x xs ⟷
  (∃ xs' xss' xss''. xss = lappend (llist-of xss') (LCons (LCons x xs') xss'') ∧
                  xs = lappend xs' (ltl (lfusecat xss'')) ∧ set xss' ⊆ {xs. lnull xs})
  (is ?lhs ⟷ ?rhs)
proof
  assume ?rhs
  then obtain xs' xss' xss''
    where xss = lappend (llist-of xss') (LCons (LCons x xs') xss'')
    and xs = ( (lappend xs' (ltl (lfusecat xss''))) )
    and set xss' ⊆ {xs. lnull xs} by blast
  moreover from ‹set xss' ⊆ {xs. lnull xs}›
  have lnull (lfusecat (llist-of xss'))
```

by (*metis lfusecat-not-lnull lset-llist-of mem-Collect-eq subset-eq*)
have *1*: *lfusecat xss = lfuse (lfusecat (llist-of xss')) (lfusecat (LCons (LCons x xs') xss''))*
by (*simp add*: *calculation*(*1*) *lfusecat-lappend*)
have *2*: *lfuse (lfusecat (llist-of xss')) (lfusecat (LCons (LCons x xs') xss'')) =*
(lfusecat (LCons (LCons x xs') xss''))
by (*simp add*: ‹*lnull (lfusecat (llist-of (xss'::'a llist list)))*› *lfuse-conv-lnull llist.expand*)
have *3*: *(lfusecat (LCons (LCons x xs') xss'')) = lfuse (LCons x xs') (lfusecat xss'')*
by *simp*
have *4*: *lfuse (LCons x xs') (lfusecat xss'') =*
(LCons x (lappend xs' (ltl (lfusecat xss''))))
unfolding *lfuse-def*
using *llist.collapse*(*1*) **by** *force*
ultimately show *?lhs*
using *1 2* **by** *auto*
next
assume *?lhs*
hence ¬ *lnull (lfusecat xss)* **by** *simp*
hence ¬ *lset xss* ⊆ {*xs. lnull xs*} **using** *lfusecat-eq-LNil lnull-def* **by** *blast*
hence ¬ *lnull (lfilter (λxs.* ¬ *lnull xs) xss)* **by**(*auto*)
then obtain *y ys* **where** *yss*: *lfilter (λxs.* ¬ *lnull xs) xss = LCons y ys*
unfolding *not-lnull-conv* **by** *auto*
from *lfilter-eq-LConsD*[*OF this*]
obtain *us vs* **where** *xss*: *xss = lappend us (LCons y vs)*
and *lfinite us*
and *lset us* ⊆ {*xs. lnull xs*} ¬ *lnull y*
and *ys*: *ys = lfilter (λxs.* ¬ *lnull xs) vs* **by** *blast*
from ‹*lfinite us*› **obtain** *us'* **where** [*simp*]: *us = llist-of us'*
unfolding *lfinite-eq-range-llist-of* **by** *blast*
from ‹*lset us* ⊆ {*xs. lnull xs*}› **have** *us*: *lnull (lfusecat us)*
using *lfusecat-eq-LNil lnull-def* **by** *blast*
from ‹¬ *lnull y*› **obtain** *y' ys'* **where** *y*: *y = LCons y' ys'*
unfolding *not-lnull-conv* **by** *blast*
from ‹*?lhs*› *us* **have** [*simp*]: *y' = x*
xs = (lappend ys' (ltl (lfusecat vs)))
unfolding *xss y*
by (*metis* ‹¬ *lnull (y::'a llist)*› ‹*lfinite (us::'a llist llist)*› *eq-LConsD lfusecat-LCons*
lfusecat-lappend lhd-lfuse y)
(*metis* ‹¬ *lnull (y::'a llist)*› ‹*lfusecat (xss::'a llist llist) = LCons (x::'a) (xs::'a llist)*›
eq-LConsD lappend-lnull2 lfusecat-LCons lfusecat-lfilter-neq-LNil ltl-lfuse y ys yss)
from ‹*lset us* ⊆ {*xs. lnull xs*}› *ys* **show** *?rhs* **unfolding** *xss y* **by** *simp blast*
qed

lemma *not-lnull-expand*:
¬ *lnull xs* ⟷ (∃ *b. xs = (LCons b LNil)*) ∨ (∃ *b xs'. xs = (LCons b xs')* ∧ ¬ *lnull xs'*)
by (*metis lhd-LCons-ltl llist.disc*(*1*) *llist.disc*(*2*) *llist.expand*)

lemma *is-LMore-llength*:
(∃ *b xs'. xs = (LCons b xs')* ∧ ¬ *lnull xs'*) ⟷ *1 < llength xs*

by (*metis dual-order.strict-implies-not-eq gr-implies-not-zero lhd-LCons-ltl llength-LCons llength-eq-0 lstrict-prefix-code*(*2*) *lstrict-prefix-code*(*4*) *lstrict-prefix-llength-less one-eSuc*)

lemma *is-LEmpty-llength*:
(∃ *b*. *xs* = (*LCons b LNil*)) ⟷ *llength xs* =*1*
by (*metis epred-llength llength-LCons llength-eq-0 llist.disc*(*1*) *ltl-simps*(*2*) *not-lnull-expand one-eSuc*)

lemma *lfusecat-all-llength-one-lfuse*:
assumes ∀ *ys* ∈ *lset* (*llist-of xss'*). *llength ys* = *1*
shows *lfuse* (*lfusecat* (*llist-of xss'*)) *ys* =
lfuse (*if* ¬ *lnull* (*llist-of xss'*) *then* (*LCons* (*lfirst* (*lfirst* (*llist-of xss'*))) *LNil*) *else LNil*) *ys*

proof −
have *1*: (*lfusecat* (*llist-of xss'*)) =
(*if* ¬ *lnull* (*llist-of xss'*) *then* (*LCons* (*lfirst* (*lfirst* (*llist-of xss'*))) *LNil*) *else LNil*)
using *assms*
proof (*induction xss'*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons as xss'*)
then show *?case*
proof (*cases xss'=Nil*)
case *True*
then show *?thesis* **using** *Cons*
by *simp* (*metis is-LEmpty-llength lfirst-def lhd-LCons*)
next
case *False*
then show *?thesis* **using** *Cons*
by *simp*
(*metis is-LEmpty-llength lfirst-def lhd-LCons llength-lnull zero-one-enat-neq*(*1*))
qed
qed
show *?thesis*
using *1* **by** *presburger*
qed

lemma *lfusecat-eq-LCons-conv-all-lset-singleton*:
assumes ∀ *ys* ∈ *lset* (*llist-of xss*). *llength ys* = *1*
¬ *lnull* (*llist-of xss*)
llastlfirst (*llist-of xss*)
shows *lset* (*llist-of xss*) = {*lfirst* (*llist-of xss*) }
using *assms*
proof (*induction xss*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons as xss*)

then show *?case*
  proof −
   have *1*: $xss = [] \lor as \in lset\ (llist\text{-}of\ xss)$
    by (*metis Cons.prems*(*1*) *Cons.prems*(*3*) *List.set-insert insert-iff is-LEmpty-llength le-numeral-extra*(*4*) *lfirst-def lhd-LCons-ltl llast-singleton llastlfirst-LCons llist.set-sel*(*1*) *llist-of.simps*(*2*) *lnull-llist-of lset-llist-of ltl-simps*(*2*) *not-in-set-insert not-lnull-llength*)
   have *2*: $xss = [] \Longrightarrow ?thesis$
    by (*simp add*: *lfirst-def*)
   have *3*: $as \in lset\ (llist\text{-}of\ xss) \Longrightarrow ?thesis$
   by (*metis 2 Cons.IH Cons.prems*(*1*) *Cons.prems*(*3*) *insert-absorb2 insert-iff lfirst-def lhd-LCons llastlfirst-LCons llist.simps*(*19*) *llist-of.simps*(*2*) *lnull-llist-of singletonD*)
   show *?thesis*
   using *1 2 3* **by** *linarith*
  qed
qed

lemma *lfusecat-eq-LCons-conv-all-the-same*:
 assumes $\forall\ ys \in lset\ (llist\text{-}of\ xss').\ llength\ ys = 1$
   $\neg\ lnull\ (llist\text{-}of\ xss')$
   $llastlfirst\ (llist\text{-}of\ xss')$
   $(llast\ (llist\text{-}of\ xss')) = (LCons\ x\ LNil)$
   $ys \in lset\ (llist\text{-}of\ xss')$
 shows $ys = (LCons\ x\ LNil)$
proof −
 have *1*: $lset\ (llist\text{-}of\ xss') = \{\ lfirst\ (llist\text{-}of\ xss')\}$
   using *assms lfusecat-eq-LCons-conv-all-lset-singleton* **by** *blast*
 have *2*: $llast\ (llist\text{-}of\ xss') = lnth\ (llist\text{-}of\ xss')\ (the\text{-}enat(epred(llength\ (llist\text{-}of\ xss'))))$
    by (*metis assms*(*2*) *co.enat.exhaust-sel enat-the-enat ile-eSuc infinity-ileE llast-conv-lnth llength-eq-0 llength-llist-of*)
 have *3*: $lnth\ (llist\text{-}of\ xss')\ (the\text{-}enat(epred(llength\ (llist\text{-}of\ xss')))) \in lset\ (llist\text{-}of\ xss')$
   by (*metis 2 assms*(*2*) *co.enat.exhaust-sel ile-eSuc in-lset-lappend-iff infinity-ileE lappend-lbutlast-llast-id llength-eq-0 llength-eq-infty-conv-lfinite llength-lbutlast llength-llist-of lset-intros*(*1*))
 have *4*: $(LCons\ x\ LNil) \in lset\ (llist\text{-}of\ xss')$
    using *2 3 assms*(*4*) **by** *force*
 show *?thesis*
 using *1 4 assms*(*5*) **by** *auto*
qed

lemma *lfusecat-eq-LCons-conv-all-the-same-a*:
 assumes *lfinite uss*
   $\forall\ ys \in lset\ (uss).\ llength\ ys = 1$
   $\neg\ lnull\ (uss)$
   $llastlfirst\ (uss)$
   $(llast\ (uss)) = (LCons\ x\ LNil)$
   $ys \in lset\ (uss)$
 shows $ys = (LCons\ x\ LNil)$
using *assms lfusecat-eq-LCons-conv-all-the-same llist-of-list-of*
by (*metis* (*full-types*))

lemma *lfusecat-eq-LCons-conv-alt*:
assumes *llastlfirst xss*
$\forall$ *ys* $\in$ *lset xss.* $\neg$ *lnull ys*
$\neg$ *lnull xs*
shows *lfusecat xss* = *LCons x xs* $\longleftrightarrow$
($\exists$ *xs′ xss′ xss″. xss* = *lappend* (*llist-of xss′*) (*LCons* (*LCons x xs′*) *xss″*) $\wedge$ $\neg$*lnull xs′* $\wedge$
xs = *lfuse xs′* (*lfusecat xss″*) $\wedge$ *set xss′* $\subseteq$ {*xs. lnull* (*ltl xs*)})
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?rhs*
then obtain *xs′ xss′ xss″*
where *xss* = *lappend* (*llist-of xss′*) (*LCons* (*LCons x xs′*) *xss″*)
and $\neg$*lnull xs′*
and *xs* = (*lfuse xs′* (*lfusecat xss″*))
and *set xss′* $\subseteq$ {*xs. lnull* (*ltl xs*)} **by** *blast*
moreover from ‹*set xss′* $\subseteq$ {*xs. lnull* (*ltl xs*)}›
have *00*: *llength* (*lfusecat* (*llist-of xss′*)) $\leq$*1*
by (*metis is-LMore-llength lfusecat-all-empty-or-LNil-a lset-llist-of lset-lltl-llength-var ltl-simps*(*2*) *mem-Collect-eq not-le-imp-less subset-eq*)
have *01*: $\forall$ *y* $\in$ *lset* (*llist-of xss′*). *llength y* = *1*
by (*metis assms*(*2*) *calculation*(*1*) *calculation*(*4*) *dual-order.strict-iff-order is-LMore-llength lset-lappend1 lset-llist-of ltl-simps*(*2*) *mem-Collect-eq not-lnull-llength subsetD*)
have *02*: *lfinite* (*llist-of xss′*)
using *lfinite-llist-of* **by** *blast*
have *03*: *llastlfirst* (*lappend* (*llist-of xss′*) (*LCons* (*LCons x xs′*) *xss″*))
using *assms calculation*(*1*) **by** *blast*
have *04*: *llastlfirst* (*llist-of xss′*)
using *assms*(*1*) *calculation*(*1*) *lfinite-llist-of llastlfirst-lappend-lfinite* **by** *blast*
have *05*: (*if lnull* (*llist-of xss′*) $\vee$ *lnull* (*LCons* (*LCons x xs′*) *xss″*)
then True
else llast (*llast* (*llist-of xss′*)) = *lfirst* (*lfirst* (*LCons* (*LCons x xs′*) *xss″*)))
using *03 02 llastlfirst-lappend-lfinite*[*of* (*llist-of xss′*) (*LCons* (*LCons x xs′*) *xss″*)]
by *blast*
have *06*: $\neg$ *lnull* (*LCons* (*LCons x xs′*) *xss″*)
by *simp*
have *07*: $\neg$ *lnull* (*llist-of xss′*) $\Longrightarrow$
llast (*llast* (*llist-of xss′*)) = *lfirst* (*lfirst* (*LCons* (*LCons x xs′*) *xss″*))
using *05 06* **by** *presburger*
have *08*: $\neg$ *lnull* (*llist-of xss′*) $\Longrightarrow$ (*llast* (*llist-of xss′*)) = (*LCons x LNil*)
using *07 lappend-lbutlast-llast-id*[*of* (*llist-of xss′*)]
by (*metis 01 02 eq-LConsD in-lset-lappend-iff is-LEmpty-llength lbutlast-lfinite lfirst-def llast-singleton lset-intros*(*1*))
have *09*: $\neg$ *lnull* (*llist-of xss′*) $\Longrightarrow$ ($\forall$ *ys* $\in$ *lset* (*llist-of xss′*). *ys* = (*LCons x LNil*))
using *lfusecat-eq-LCons-conv-all-the-same*
by (*metis 01 04 08*)
have *0*: (*lfusecat* (*llist-of xss′*)) = *LNil* $\vee$ (*lfusecat* (*llist-of xss′*)) = (*LCons x LNil*)
by (*metis 00 09 eq-LConsD is-LMore-llength lfirst-def lfirst-lfusecat lfusecat-LNil lhd-LCons-ltl*

linorder-not-le llist.collapse(*1*) *llist.set-sel*(*1*))
have *1*: *lfusecat xss* = *lfuse* (*lfusecat* (*llist-of xss'*)) (*lfusecat* (*LCons* (*LCons x xs'*) *xss''*))
by (*simp add*: *calculation*(*1*) *lfusecat-lappend*)
have *2*: *lfuse* (*lfusecat* (*llist-of xss'*)) (*lfusecat* (*LCons* (*LCons x xs'*) *xss''*)) =
(*lfusecat* (*LCons* (*LCons x xs'*) *xss''*))
using *0* **by** *fastforce*
have *3*: (*lfusecat* (*LCons* (*LCons x xs'*) *xss''*)) = *lfuse* (*LCons x xs'*) (*lfusecat xss''*)
by *simp*
have *4*: *lfuse* (*LCons x xs'*) (*lfusecat xss''*) =
(*LCons x* ((*lfuse xs'* ((*lfusecat xss''*)))
))
unfolding *lfuse-def*
using *calculation*(*2*) **by** *auto*
ultimately show *?lhs*
using *1 2* **by** *auto*
next
assume *?lhs*
hence ¬ *lnull* (*lfusecat xss*) **by** *simp*
hence ¬ *lset xss* ⊆ {*xs*. *lnull* (*ltl xs*)}
by (*metis* ‹*lfusecat* (*xss*::'*a llist llist*) = *LCons* (*x*::'*a*) (*xs*::'*a llist*)› *assms*(*3*)
lfusecat-all-empty-or-LNil-a lnull-ldrop lset-lltl-llength-var ltl-ldrop-one ltl-simps(*2*)
mem-Collect-eq subsetD)
hence ¬ *lnull* (*lfilter* (λ*xs*. ¬ *lnull* (*ltl xs*)) *xss*) **by**(*auto*)
then obtain *y ys* **where** *yss*: *lfilter* (λ*xs*. ¬ *lnull* (*ltl xs*)) *xss* = *LCons y ys*
unfolding *not-lnull-conv* **by** *auto*
from *lfilter-eq-LConsD*[*OF this*]
obtain *us vs* **where** *xss*: *xss* = *lappend us* (*LCons y vs*)
and *lfinite us*
and *lset us* ⊆ {*xs*. *lnull* (*ltl xs*)} ¬ *lnull* (*ltl y*)
and *ys*: *ys* = *lfilter* (λ*xs*. ¬ *lnull* (*ltl xs*)) *vs* **using** *lnull-ltlI* **by** *blast*
from ‹*lfinite us*› **obtain** *us'* **where** [*simp*]: *us* = *llist-of us'*
unfolding *lfinite-eq-range-llist-of* **by** *blast*
from ‹*lset us* ⊆ {*xs*. *lnull* (*ltl xs*)}› **have** *us*: *llength* (*lfusecat us*) ≤*1*
using *lfusecat-eq-LNil*
by (*metis lfusecat-all-empty-or-LNil-a lnull-ldrop lset-lltl-llength-var ltl-ldrop-one*
mem-Collect-eq subset-eq)
from ‹¬ *lnull* (*ltl y*)› **obtain** *y' ys'* **where** *y*: *y* = *LCons y' ys'*
unfolding *not-lnull-conv*
by (*metis* ‹¬ *lnull* (*ltl y*)› *lhd-LCons-ltl lnull-ltlI*)
have *100*: *llastlfirst us*
using ‹*lfinite us*› *assms*(*1*) *llastlfirst-lappend-lfinite xss* **by** *blast*
have *101*: ¬ *lnull ys'*
using ‹¬ *lnull* (*ltl y*)› *y* **by** *auto*
have *102*: ¬ *lnull* (*LCons y vs*)
by *simp*
have *103*: ¬ *lnull us* ⟹ *llast* (*llast us*) = *lfirst* (*lfirst* (*LCons y vs*))
using *llastlfirst-lappend-lfinite*[*of us* (*LCons y vs*)]
using *assms*(*1*) *xss* **by** *auto*
have *104*: (∀ *z* ∈ *lset us*. *llength z* =*1*)
by (*metis* ‹*lset us* ⊆ {*xs*. *lnull* (*ltl xs*)}› *assms*(*2*)

dual-order.strict-iff-order eq-LConsD in-lset-lappend-iff is-LMore-llength mem-Collect-eq not-lnull-llength subsetD xss)
have *1040*: ¬ *lnull us* ⟹ *llast us* ∈ *lset us*
by *simp*
have *1041*: ¬ *lnull us* ⟹ *llast us* = *LCons x LNil*
using *lappend-lbutlast-llast-id*[*of us*] *100 104 1040*
lfusecat-eq-LCons-conv-all-the-same-a[*of us x llast us*]
lfusecat-eq-LCons-conv-all-lset-singleton[*of us′*]
by (*metis* ‹¬ *lnull* (*lfusecat xss*)› ‹*lfusecat xss* = *LCons x xs*› ‹*us* = *llist-of us′*›
eq-LConsD is-LEmpty-llength lfirst-def lfirst-lfusecat-lfirst lfusecat-not-lnull lhd-lappend lset-eq-empty not-lnull-lset-conv-a singletonD xss)
have *105*: ¬ *lnull us* ⟹ (∀ *z* ∈ *lset us*. *z*= (*LCons x LNil*))
using *lfusecat-eq-LCons-conv-all-the-same-a*[*of us x*]
using *100 104 1041* ‹*lfinite us*› **by** *blast*
have *106*: ¬ *lnull us* ⟹ *lfusecat us* = (*LCons x LNil*)
by (*metis 100 104 1041* ‹*lfinite us*› *dual-order.antisym is-LEmpty-llength lfinite-LConsI lfinite-LNil lfusecat-not-lnull-var llast-singleton llastfirst-lfusecat-llast llength-LNil lset-eq-empty not-lnull-llength not-lnull-lset-conv-a us zero-one-enat-neq*(*1*))
from ‹*?lhs*› *us* **have** [*simp*]: *y′* = *x*
by (*metis 103 1041* ‹¬ *lnull* (*ltl y*)› *eq-LConsD lappend-lnull1 lfirst-def lfirst-lfusecat llast-singleton lnull-ltlI xss y*)
from ‹*?lhs*› *us* **have** [*simp*]: *xs* = ((*lfuse ys′* ((*lfusecat vs*))))
using *lfusecat-lappend*[*of us* (*LCons y vs*)] *lfusecat-LCons*[*of y vs*] *y xss*
101 ‹*lfinite us*› ‹*lfusecat xss* = *LCons x xs*›
by (*metis 103 1041 106 eq-LConsD lappend-code*(*1*) *lappend-lnull1 lbutlast-simps*(*2*) *lfirst-def lfuse-LCons-a lfuse-conv-lnull lfuse-lbutlast lnull-ltlI*)
from ‹*lset us* ⊆ {*xs*. *lnull* (*ltl xs*)}› *ys* **show** *?rhs* **unfolding** *xss y*
using *101* **by** *auto*
qed

lemma *lfusecat-eq-One-conv*:
lfusecat xss = *LCons x LNil* ⟷
(∃ *xss′ xss″*. *xss* = *lappend* (*llist-of xss′*) (*LCons* (*LCons x LNil*) *xss″*) ∧
LNil = (*ltl* (*lfusecat xss″*)) ∧ *set xss′* ⊆ {*xs*. *lnull xs*})
by (*metis lappend-eq-LNil-iff lfusecat-eq-LCons-conv*)

lemma *llength-lfusecat-eq-one-conv*:
llength (*lfusecat xss*) = *1* ⟷
(∃ *x xss′ xss″*. *xss* = *lappend* (*llist-of xss′*) (*LCons* (*LCons x LNil*) *xss″*) ∧
LNil = (*ltl* (*lfusecat xss″*)) ∧ *set xss′* ⊆ {*xs*. *lnull xs*})
by (*metis is-LEmpty-llength lfusecat-eq-One-conv*)

lemma *lfusecat-lfinite-b*:
assumes *lfinite*(*lfusecat xss*)
∀ *xs* ∈ *lset xss*. ¬ *lnull* (*ltl xs*)
∀ *xs* ∈ *lset xss*. *lfinite xs*
llastlfirst xss
shows *lfinite xss*

using *assms*
proof (*induct zs*≡(*lfusecat xss*) *arbitrary*: *xss*)
case *lfinite-LNil*
then show *?case*
by (*metis lfusecat-not-lnull-var llist.disc*(*1*) *lnull-imp-lfinite lset-eq-empty ltl-simps*(*1*)
not-lnull-lset-conv-a)
next
case (*lfinite-LConsI xs x*)
then show *?case*
proof –
have *1*: ($\exists$ *xs′ xss′ xss″*. *xss* = *lappend* (*llist-of xss′*) (*LCons* (*LCons x xs′*) *xss″*) $\wedge$
xs = *lappend xs′* (*ltl* (*lfusecat xss″*)) $\wedge$ *set xss′* $\subseteq$ {*xs*. *lnull xs*})
using *lfinite-LConsI.hyps*(*3*) *lfusecat-eq-LCons-conv* **by** *fastforce*
obtain *xs′ xss′ xss″* **where** *2*: *xss* = *lappend* (*llist-of xss′*) (*LCons* (*LCons x xs′*) *xss″*) $\wedge$
xs = *lappend xs′* (*ltl* (*lfusecat xss″*)) $\wedge$ *set xss′* $\subseteq$ {*xs*. *lnull xs*}
using *1* **by** *blast*
have *3*: (*llist-of xss′*) = *LNil*
by (*metis 2 in-lset-lappend-iff lconcat-eq-LNil lfinite-LConsI.prems*(*1*) *llist.collapse*(*1*)
llist.set-sel(*1*) *lnull-lconcat lset-eq-forall-lnull lset-llist-of ltl-simps*(*1*))
have *4*: *xss* = (*LCons* (*LCons x xs′*) *xss″*)
by (*simp add*: *2 3*)
have *5*: *llastlfirst xss*
by (*simp add*: *lfinite-LConsI.prems*(*3*))
have *6*: ¬ *lnull xs′*
using *4 lfinite-LConsI.prems*(*1*) **by** *force*
have *7*: *lnull xss″* $\Longrightarrow$ *?thesis*
by (*simp add*: *4*)
have *8*: ¬ *lnull xss″* $\Longrightarrow$ *?thesis*
proof –
assume *a*: ¬*lnull xss″*
have *9*: *llast* (*LCons x xs′*) = *lfirst* (*lfirst* (*xss″*))
using *a 6 4 5 llastlfirst-LCons* **by** *blast*
have *10*: $\forall$ *xs* $\in$ *lset xss″*. ¬ *lnull* (*ltl xs*)
by (*simp add*: *4 lfinite-LConsI.prems*)
have *11*: $\forall$ *xs* $\in$ *lset xss″*. *lfinite xs*
by (*simp add*: *4 lfinite-LConsI.prems*)
have *12*: *llastlfirst xss″*
using *4 5 a* **by** *auto*
have *13*: *lfinite*(*lfusecat xss″*)
using *2 lfinite-LConsI.hyps*(*1*) **by** *auto*
have *14*: *xs* = *lappend* (*lbutlast xs′*) (*lfusecat xss″*)
by (*metis 10 2 6 9 a lappend-lbutlast-llast-id lappend-snocL1-conv-LCons2 lfirst-def*
lfirst-lfusecat-lfirst lfusecat-not-lnull-var lhd-LCons-ltl llast-LCons llist.disc(*1*)
llist.set-sel(*1*) *lset-eq-empty ltl-simps*(*1*) *not-lnull-lset-conv-a*)
have *15*: *xs*= *lfuse xs′* (*lfusecat xss″*)
by (*simp add*: *2 6 lappend-lnull2 lfuse-def*)
have *16*: *xs* = *lfusecat* (*LCons xs′ xss″*)
by (*simp add*: *15*)
have *17*: *lnull* (*ltl xs′*) $\Longrightarrow$ *?thesis*
by (*metis 10 11 12 14 4 lappend-code*(*1*) *lbutlast.ctr*(*1*) *lfinite.lfinite-LConsI*

lfinite-LConsI.hyps(*2*) *llist.collapse*(*1*))
have *18*: ¬*lnull* (*ltl xs'*) ⟹ (∀ *xs* ∈ *lset* (*LCons xs' xss''*). ¬ *lnull* (*ltl xs*))
by (*simp add*: *10*)
have *19*: ∀ *xs* ∈ *lset* (*LCons xs' xss''*). *lfinite xs*
by (*metis 11 2 insert-iff lappend-inf lfinite-LConsI.hyps*(*1*) *llist.simps*(*19*))
have *20*: *llastlfirst* (*LCons xs' xss''*)
by (*metis 12 6 9 a llast-LCons llastlfirst-LCons*)
have *21*: *lfinite* (*LCons xs' xss''*)
by (*metis 16 17 18 19 20 4 lfinite-LConsI.hyps*(*2*) *lfinite-code*(*2*))
show *?thesis*
using *21 4* **by** *auto*
qed
show *?thesis*
using *7 8* **by** *blast*
qed
qed

lemma *lfusecat-lfinite*:
assumes ∀ *xs* ∈ *lset xss*. ¬ *lnull* (*ltl xs*)
∀ *xs* ∈ *lset xss*. *lfinite xs*
llastlfirst xss
shows *lfinite* (*lfusecat xss*) ⟷ *lfinite xss*
using *assms lfusecat-lfinite-a lfusecat-lfinite-b* **by** *blast*

lemma *ridx-lfusecat-ltake*:
assumes *ridx R* (*lfusecat xss*)
$n \leq$ *llength xss*
shows *ridx R* (*lfusecat* (*ltake n xss*))
using *assms*
by (*metis lfusecat-split ltake-all ltake-lfuse nle-le ridx-ltake-a*)

lemma *ridx-lfusecat-ldrop*:
assumes *ridx R* (*lfusecat xss*)
(*enat n*) < *llength xss*
llastlfirst xss
∀ *xs* ∈ *lset xss*. ¬ *lnull xs*
∀ *xs* ∈ *lset xss*. *lfinite xs*
shows *ridx R* (*lfusecat* (*ldrop n xss*))
proof −
have *1*: (*lfusecat* (*ldrop n xss*)) =
ldrop (*if n* = *0 then 0 else* (∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)
using *assms gr-implies-not-zero lfusecat-ldrop llength-eq-0* **by** *blast*
have *2*: (*if n* = *0 then 0 else* (∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) < *llength* (*lfusecat xss*)
by (*metis* (*no-types*, *lifting*) *1 add.commute add.right-neutral assms*(*2*) *assms*(*4*) *in-lset-conv-lnth lfusecat-not-lnull llist.set-sel*(*1*) *lnth-0-conv-lhd lnth-ldrop lnull-ldrop not-less-iff-gr-or-eq order.order-iff-strict the-enat.simps zero-enat-def*)
have *3*: ∃ *k*. (*enat k*) = (*if n* = *0 then 0 else* (∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
proof (*cases n*)

case *0*
then show *?thesis* **by** (*simp add*: *zero-enat-def*)
next
case (*Suc nat*)
then show *?thesis* **by** (*metis* (*mono-tags*, *lifting*) *2 enat-iless enat-less-imp-le leD*)
qed
have *4*: *ridx R* (*ldrop* (*if n* = *0 then 0 else* ($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*))
using *ridx-ldrop*[*of R* (*lfusecat xss*)]
using *2 assms*(*1*) *order.strict-implies-order* **by** *blast*
show *?thesis* **by** (*simp add*: *1 4*)
qed

lemma *ridx-lfusecat-a*:
assumes *llastlfirst xss*
∀ *xs* ∈ *lset xss*. ¬ *lnull xs*
∀ *xs* ∈ *lset xss*. *lfinite xs*
ridx R (*lfusecat xss*)
i < *llength xss*
shows *ridx R* (*lnth xss i*)
proof −
have *1*: *lsub i i xss* = (*LCons* (*lnth xss i*) *LNil*)
by (*simp add*: *assms*(*5*) *lsub-same*)
have *2*: *lsub i i xss* = *ltake* (*eSuc* (*enat* (*i* − *i*))) (*ldrop* (*enat i*) *xss*)
by (*simp add*: *lsub-def*)
have *3*: *ltake* (*eSuc* (*enat* (*i* − *i*))) (*ldrop* (*enat i*) *xss*) = *ltake 1* (*ldrop* (*enat i*) *xss*)
by (*simp add*: *eSuc-enat one-enat-def*)
have *4*: *ridx R* (*lfusecat* (*ltake 1* (*ldrop* (*enat i*) *xss*)))
by (*metis assms ltake-all nle-le ridx-lfusecat-ldrop ridx-lfusecat-ltake*)
have *5*: (*lfusecat* (*ltake 1* (*ldrop* (*enat i*) *xss*))) = (*lnth xss i*)
by (*metis 1 2 3 lfuse-LNil-2 lfusecat-LCons lfusecat-LNil*)
show *?thesis*
using *4 5* **by** *auto*
qed

lemma *ridx-lfuse-lfinite*:
assumes *lfinite l1*
llast l1 = *lfirst l2*
shows *ridx R* (*lfuse l1 l2*) ⟷ *ridx R l1* ∧ *ridx R l2*
using *assms*
proof (*cases* ¬ *lnull l1* ∧ ¬ *lnull l2*)
case *True*
then show *?thesis*
proof (*cases* ¬ *lnull* (*ltl l2*))
case *True*
then show *?thesis*
proof −
have *1*: (*lfuse l1 l2*) = *lappend l1* (*ltl l2*)
unfolding *lfuse-def* **using** ‹¬ *lnull l1* ∧ ¬ *lnull l2*› **by** *simp*

```
      have 2: ridx R (lappend l1 (ltl l2)) = (ridx R l1 ∧ ( R (llast l1) (lhd (ltl l2))) ∧ ridx R (ltl l2))
        using assms ridx-lappend-lfinite[of l1 R ltl l2 ]  True ‹¬ lnull l1 ∧ ¬ lnull l2›
          by auto
      have 3: ( R (llast l1) (lhd (ltl l2)) ∧ ridx R (ltl l2)) =  ridx R l2
         using  True ‹¬ lnull l1 ∧ ¬ lnull l2› ridx-LCons-1[of R lfirst l2 ltl l2]
         by (simp add: assms lfirst-def)
      show ?thesis
      by (simp add: 1 2 3)
    qed
  next
   case False
   then show ?thesis
     proof −
      have 4: (lfuse l1 l2) = lappend l1 (ltl l2)
         unfolding lfuse-def using ‹¬ lnull l1 ∧ ¬ lnull l2› by simp
      have 5: ridx R (lappend l1 (ltl l2)) =  (ridx R l1 ∧ ridx R l2)
         using assms ridx-lappend-lfinite[of l1 R ltl l2 ] False ‹¬ lnull l1 ∧ ¬ lnull l2›
          ridx-expand-1 by (metis lhd-LCons-ltl ridx-LCons-1)
      show ?thesis by (simp add: 4 5)
     qed
   qed
next
case False
then show ?thesis
by (metis lfuse-LNil-1 lfuse-LNil-2 llist.collapse(1) ridx-expand-1)
qed



lemma ridx-lfusecat-lfuse:
 assumes llastlfirst xss
         ∀ xs ∈ lset xss. ¬ lnull xs
         ∀ xs ∈ lset xss. lfinite xs
         ∀ i.  i < llength xss ⟶  ridx R (lnth xss i)
          (Suc n) < llength xss
 shows    ridx R (lfuse (lnth xss n) (lnth xss (Suc n)))
using assms
by (metis Suc-ile-eq  lfuse-inf llastlfirst-def order-less-imp-le ridx-lfuse-lfinite)


lemma ridx-lfusecat-ltake-a:
 assumes llastlfirst xss
         ∀ xs ∈ lset xss. ¬ lnull xs
         ∀ xs ∈ lset xss. lfinite xs
         ∀ i.  i < llength xss ⟶  ridx R (lnth xss i)
         n ≤ llength xss
 shows   ridx R (lfusecat (ltake (enat n) xss))
using assms
proof (induct n arbitrary: xss)
case 0
```

then show *?case*
by (*metis LNil-eq-ltake-iff gr-implies-not-zero lfusecat-LNil llength-eq-0 llist.disc*(*1*) *ridx-def zero-enat-def*)
next
case (*Suc n*)
then show *?case*
proof −
have *0*: *n=0* $\Longrightarrow$ *?thesis*
proof −
assume *a0*: *n=0*
have *000*: ¬*lnull xss*
using *Suc.prems*(*5*) *a0 one-enat-def* **by** *force*
have *001*: (*ltake* (*enat* (*Suc n*)) *xss*) = (*LCons* (*lnth xss 0*) *LNil*)
using *a0 000*
by (*metis One-nat-def lhd-LCons-ltl lnth-0-conv-lhd ltake.ctr*(*1*) *ltake-eSuc-LCons one-eSuc one-enat-def*)
have *002*: *lfusecat* (*ltake* (*enat* (*Suc n*)) *xss*) = (*lnth xss 0*)
using *a0 000 001 lfusecat-LCons*[*of* (*lnth xss 0*) *LNil*] **by** *simp*
show *?thesis* **using** *002 Suc.prems*(*4*) *Suc.prems*(*5*) *Suc-ile-eq a0* **by** *auto*
qed
have *01*: *n>0* $\Longrightarrow$ *?thesis*
proof −
assume *a*: *n>0*
have *1*: (*lfusecat* (*ltake* (*enat* (*Suc n*)) *xss*)) =
ltake (*eSuc* ($\sum$ *i*::*nat* = *0*::*nat*..*Suc n* − (*1*::*nat*). *epred* (*llength* (*lnth xss i*)))) (*lfusecat xss*)
using *lfusecat-ltake*[*of xss* ((*Suc n*))]
using *Suc.prems*(*1*) *Suc.prems*(*2*) *Suc.prems*(*3*) *Suc.prems*(*5*) *Suc-ile-eq* **by** *force*
have *2*: *ridx R* (*lfusecat* (*ltake* (*enat n*) *xss*))
using *Suc Suc-ile-eq order.strict-implies-order* **by** *blast*
have *20*: *lfinite* (*ltake* (*enat n*) *xss*)
by *simp*
have *21*: ¬ *lnull* (*ltake* (*enat n*) *xss*)
using *Suc.prems*(*5*) *a enat-0-iff*(*2*) **by** *fastforce*
have *22*: ∀ *xs*∈*lset* (*ltake* (*enat n*) *xss*). ¬ *lnull xs*
by (*meson Suc.prems*(*2*) *lset-ltake subsetD*)
have *23*: ∀ *xs*∈*lset* (*ltake* (*enat n*) *xss*). *lfinite xs*
by (*meson Suc.prems*(*3*) *lset-ltake subsetD*)
have *24*: *llastlfirst* (*ltake* (*enat n*) *xss*)
using *Suc.prems*(*1*) *Suc.prems*(*5*) *Suc-ile-eq less-imp-le llastlfirst-ltake* **by** *blast*
have *25*: *llast* (*lfusecat* (*ltake* (*enat n*) *xss*)) = *llast* (*llast* (*ltake* (*enat n*) *xss*))
using *20 21 22 23 24 llastfirst-lfusecat-llast* **by** *blast*
have *26*: *lfinite* (*llast* (*ltake* (*enat n*) *xss*))
by (*metis Suc.prems*(*3*) *Suc.prems*(*5*) *Suc-ile-eq Suc-pred′ a enat-ord-code*(*4*) *in-lset-conv-lnth lfinite-ltake llast-lappend-LCons llast-singleton ltake-Suc-conv-snoc-lnth order-less-imp-le*)
have *27*: *n*< *llength xss*
using *Suc.prems*(*5*) *Suc-ile-eq order-less-imp-le* **by** *blast*
have *271*: (*llast* (*ltake* (*enat n*) *xss*)) = (*lnth xss* (*n*−*1*))
by (*metis 27 Suc-diff-1 Suc-ile-eq a enat-ord-code*(*4*) *lfinite-ltake llast-lappend-LCons llast-singleton ltake-Suc-conv-snoc-lnth order.strict-implies-order*)
have *28*: *llast* (*llast* (*ltake* (*enat n*) *xss*)) = *llast* (*lnth xss* (*n*−*1*))

using *271* **by** *auto*
have *29*: *llast* (*lnth xss* (*n*−*1*)) = *lfirst* (*lnth xss n*)
by (*metis 27 Suc.prems*(*1*) *Suc-pred′ a llastlfirst-def*)
have *30*: *ridx R* (*lfuse* (*lfusecat* (*ltake* (*enat n*) *xss*)) (*lnth xss* (*n*)))
by (*metis 2 25 27 28 29 Suc.prems*(*4*) *lfuse-inf ridx-lfuse-lfinite*)
have *31*: (*lfuse* (*lfusecat* (*ltake* (*enat n*) *xss*)) (*lnth xss* (*n*))) =
(*lfusecat* (*ltake* (*enat* (*Suc n*)) *xss*))
by (*simp add*: *27 lfusecat-lappend ltake-Suc-conv-snoc-lnth*)
show *?thesis*
using *30 31* **by** *auto*
qed
show *?thesis*
using *0 01* **by** *fastforce*
qed
qed

lemma *lfusecat-ltake-llength-less-than-llength-lfusecat*:
assumes *llastlfirst xss*
∀ *xs* ∈ *lset xss*. *1* < *llength xs*
∀ *xs* ∈ *lset xss*. *lfinite xs*
(*enat n*)≤ *llength xss*
shows *min*
(*if n* =*0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*)) =
(*if n* =*0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
proof −
have *0*: ∀ *xs* ∈ *lset xss*. ¬*lnull xs*
using *assms*(*2*) **by** *fastforce*
have *1*: *lfinite xss* ⟹ ¬ *lnull xss*⟹ *llength*(*lfusecat xss*) =
eSuc(∑ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*)))
using *lfusecat-llength-lfinite*[*of xss*] *assms 0* **by** *fastforce*
have *2*: ¬ *lfinite xss* ⟹ ¬*lfinite* (*lfusecat xss*)
using *assms lfusecat-lfinite*[*of xss*] *0*
by (*metis less-numeral-extra*(*4*) *lhd-LCons-ltl llength-LCons llength-LNil llist.collapse*(*1*) *one-eSuc*)
have *3*: ¬ *lfinite xss* ⟹ *llength* (*lfusecat xss*) = ∞
using *not-lfinite-llength 2* **by** *blast*
have *50*: ¬ *lnull xss*⟹ *lfusecat* (*ltake* (*enat n*) *xss*) =
ltake (*if n* =*0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)
using *assms lfusecat-ltake*[*of xss n*] *0* **by** *fastforce*
have *51*: *lfinite* (*lfusecat* (*ltake* (*enat n*) *xss*))
by (*meson assms*(*3*) *enat-ord-code*(*4*) *lfinite-ltake lfusecat-lfinite-a lset-ltake subsetD*)
have *52*: *lfinite* (*ltake* (*if n* =*0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*))
using *50 51 llist.collapse*(*1*) **by** *fastforce*
have *53*: ∃ *m*. *llength* (*ltake* (*if n* =*0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) = (*enat m*)
using *52 lfinite-llength-enat* **by** *blast*
have *54*: *llength* (*ltake* (*if n* =*0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) ≤ *llength* (*lfusecat xss*)

by *auto*
have *55*: *llength* (*ltake* (*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)) =
min
(*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*))
using *llength-ltake*[*of* (*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*)] **by** *auto*
have *56*: *min*
(*if n* =*0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*)) < ∞
by (*metis 53 55 enat-ord-code*(*4*))
have *57*: (*llength* (*lfusecat xss*)) = *llength* (*lfusecat* (*lappend* (*ltake n xss*) (*ldrop n xss*)))
by (*simp add*: *lappend-ltake-ldrop*)
have *58*: *llength* (*lfusecat* (*lappend* (*ltake n xss*) (*ldrop n xss*))) =
llength (*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*)))
by (*metis 57 lfusecat-split*)
have *59*: *lnull* (*lfusecat* (*ltake* (*enat n*) *xss*)) ⟷ *n* = *0*
proof (*cases xss*)
case *LNil*
then show *?thesis* **using** *assms enat-0-iff*(*2*) **by** *force*
next
case (*LCons x21 x22*)
then show *?thesis* **using** *1 2 50* **by** *force*
qed
have *60*: *llength* (*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))) =
llength (*lfusecat* (*ltake* (*enat n*) *xss*)) +
(*if n*= *0 then llength* (*lfusecat* (*ldrop* (*enat n*) *xss*))
else epred (*llength* (*lfusecat* (*ldrop* (*enat n*) *xss*))))
using *lfuse-llength*[*of* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*))]
using *59* **by** *presburger*
have *62*: *n*= *0* ⟹ *llength* (*lfusecat* (*ldrop* (*enat n*) *xss*)) = *llength* (*lfusecat xss*)
using *57 58 59 llist.collapse*(*1*) **by** *force*
have *620*: *n*>*0* ⟹ *n*<*llength xss* ⟹ ¬ *lnull* (*ldrop* (*enat n*) *xss*)
using *Suc-ile-eq assms* **by** *simp*
have *621*: *n*>*0* ⟹ *n*<*llength xss* ⟹ (∃ *xs*∈*lset* (*ldrop* (*enat n*) *xss*). ¬ *lnull xs*)
by (*meson 0 620 in-lset-ldropD lfusecat-not-lnull lfusecat-not-lnull-var*)
have *622*: *n*>*0* ⟹ *n*<*llength xss* ⟹ (¬ *lnull* (*lfusecat* (*ldrop* (*enat n*) *xss*)))
using *620 621 lfusecat-not-lnull*[*of* (*ldrop* (*enat n*) *xss*)] **by** *blast*
have *623*: *n*>*0* ⟹ ¬ *lnull xss*
using *assms 59* **by** *force*
have *63*: *n*>*0* ⟹ *n*<*llength xss* ⟹ (*llength* (*lfusecat* (*ldrop* (*enat n*) *xss*))) =
(*llength* (*ldrop* (($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*)))
using *lfusecat-ldrop*[*of xss n*] *assms*
using *623 llength-LNil not-one-less-zero 0* **by** *presburger*
have *630*: *n*>*0* ⟹ *n*<*llength xss* ⟹ *epred*(*llength* (*lfusecat* (*ldrop* (*enat n*) *xss*))) =
epred(*llength* (*ldrop* (($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*)))
using *63* **by** *presburger*
have *631*: *n*>*0* ⟹ *n*<*llength xss* ⟹
(*llength* (*ldrop* (($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*))) > *0*

by (*metis 622 63 gr-zeroI llength-eq-0*)
have *64*: *llength* (*lfusecat* (*ltake* (*enat n*) *xss*)) ≤
llength (*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*)))
using *lfuse-llength-le-a* **by** *blast*
have *65*: $0 < n \Longrightarrow n<$*llength xss* $\Longrightarrow 0<$ (*llength* (*lfusecat* (*ldrop* (*enat n*) *xss*)))
using *63 631* **by** *presburger*
have *66*: $n>0 \Longrightarrow$ *llength xss* > 0
using *623 gr-zeroI llength-eq-0* **by** *blast*
have *67*: $n>0 \Longrightarrow n-1 \leq$ (*epred* (*llength xss*))
using *assms 66*
by (*metis epred-enat epred-le-epredI*)
have *68*: *min*
(*if* $n = 0$ *then* 0 *else* *eSuc*($\sum$ $i = 0$ .. $(n-1)$. *epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*)) =
(*if* $n = 0$ *then* 0 *else* *eSuc*($\sum$ $i = 0$ .. $(n-1)$. *epred*(*llength* (*lnth xss i*))))
proof (*cases lfinite xss*)
case *True*
then show *?thesis*
proof −
have *681*: $n>0 \Longrightarrow n<$*llength xss* $\Longrightarrow$ *eSuc*($\sum$ $i = 0$ .. $(n-1)$. *epred*(*llength* (*lnth xss i*))) ≤
eSuc ($\sum i = 0$..*the-enat* (*epred* (*llength xss*)). *epred* (*llength* (*lnth xss i*)))
by (*metis 1 631 True gr-implies-not-zero ileI1 linorder-not-less llength-lnull lnull-ldrop*)
have *682*: $n>0 \Longrightarrow n=$*llength xss* $\Longrightarrow$ *eSuc*($\sum$ $i = 0$ .. $(n-1)$. *epred*(*llength* (*lnth xss i*))) ≤
eSuc ($\sum i = 0$..*the-enat* (*epred* (*llength xss*)). *epred* (*llength* (*lnth xss i*)))
by (*metis dual-order.refl epred-enat the-enat.simps*)
show *?thesis*
using *1 681 682 True 623 assms*(*4*) *min.absorb1 order.order-iff-strict* **by** *auto*
qed
next
case *False*
then show *?thesis* **using** *3 min-enat-simps*(*4*) **by** *presburger*
qed
show *?thesis*
using *68* **by** *blast*
qed

lemma *lfusecat-ltake-llength-less-than-next*:
assumes *llastlfirst xss*
∀ *xs* ∈ *lset xss*. $1 <$ *llength xs*
∀ *xs* ∈ *lset xss*. *lfinite xs*
(*Suc n*) $<$ *llength xss*
shows (*if* $n = 0$ *then* 0 *else* *eSuc*($\sum$ $i = 0$ .. $(n-1)$. *epred*(*llength* (*lnth xss i*))))
$<$ (*if* (*Suc n*) $= 0$ *then* 0 *else* *eSuc*($\sum$ $i = 0$ .. (n) . *epred*(*llength* (*lnth xss i*))))
proof −
have *0*: $n<\infty$
by *simp*
have *1*: $\bigwedge i$. $i<$ *llength xss* $\Longrightarrow 1<$ *llength* (*lnth xss i*)

by (*meson assms*(*2*) *in-lset-conv-lnth*)
have *01*: $\bigwedge$*i. i< llength xss* $\Longrightarrow$ $\neg$*lnull* (*lnth xss i*)
by (*metis 1 llength-LNil llist.collapse*(*1*) *not-one-less-zero*)
have *02*: $\forall$ *xs* $\in$ *lset xss.* $\neg$*lnull xs*
using *assms*(*2*) **by** *fastforce*
have *2*: $\bigwedge$*i . i< llength xss* $\Longrightarrow$ *0<epred*(*llength* (*lnth xss i*))
by (*metis 1 co.enat.exhaust-sel gr-zeroI less-numeral-extra*(*4*) *not-one-less-zero one-eSuc*)
have *3*: *n= 0* $\Longrightarrow$ *?thesis*
by *auto*
have *4*: *0<n* $\Longrightarrow$ ($\sum$ *i = 0 ..* (*n*) *. epred*(*llength* (*lnth xss i*))) =
($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*))) +
epred(*llength* (*lnth xss n*))
by (*metis* (*no-types, lifting*) *Suc-diff-1 sum.atLeast0-atMost-Suc*)
have *5*: $\bigwedge$*i. i< llength xss* $\Longrightarrow$ *epred*(*llength* (*lnth xss i*)) < ∞
using *assms*(*3*)
by (*metis enat-ord-simps*(*4*) *epred-0 epred-Infty epred-inject in-lset-conv-lnth infinity-ne-i0 llength-eq-infty-conv-lfinite*)
have *50*: *llength* (*lfusecat* (*ltake n xss*)) $\leq$ *llength*(*lfusecat xss*)
by (*simp add*: *lprefix-lfusecatI lprefix-llength-le*)

have *6*: *0< n* $\Longrightarrow$ *eSuc*($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*)))< ∞
proof −
assume *a*: *n>0*
have *60*: (*lfusecat* (*ltake n xss*)) =
(*ltake* (*eSuc*($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*))
using *lfusecat-ltake*[*of xss n*] **using** *assms 02 a* **by** *simp*
(*metis Suc-ile-eq gr-implies-not-zero llength-eq-0 order.strict-implies-order*)
have *61*: *llength* (*lfusecat* (*ltake n xss*)) =
llength (*ltake* (*eSuc*($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*))
using *60* **by** *fastforce*
have *62*: *llength* (*ltake* (*eSuc*($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*)) =
min (*eSuc*($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*))
using *llength-ltake* **by** *blast*
have *63*: *min* (*eSuc*($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*))))
(*llength* (*lfusecat xss*)) =
(*eSuc*($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*))))
using *lfusecat-ltake-llength-less-than-llength-lfusecat*[*of xss n*] *a assms*
using *Suc-ile-eq order-less-imp-le* **by** *simp blast*
show *?thesis*
by (*metis 0 60 62 63 assms*(*3*) *enat-ord-simps*(*4*) *lfinite-ltake lfusecat-lfinite-a llength-eq-infty-conv-lfinite lset-ltake subsetD*)
qed
have *7*: *0<n* $\Longrightarrow$ *eSuc*($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*))) <
eSuc($\sum$ *i = 0 ..* (*n−1*) *. epred*(*llength* (*lnth xss i*))) +
epred(*llength* (*lnth xss n*))
by (*metis 2 6 Suc-ile-eq add.right-neutral assms*(*4*) *enat-add-mono less-infinityE order-less-imp-le*)
have *8*: *0<n* $\Longrightarrow$ *?thesis*
using *4 7 eSuc-plus* **by** *auto*

show *?thesis*
using *3 8* **by** *blast*
qed

lemma *ridx-lfusecat-b*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss. 1* $<$ *llength xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
$\forall$ *i.* *i* $<$ *llength xss* $\longrightarrow$ *ridx R* (*lnth xss i*)
shows *ridx R* (*lfusecat xss*)
proof –
have *0*: $\forall$ *xs* $\in$ *lset xss.* $\neg$ *lnull xs*
using *assms*(*2*) *gr-implies-not-zero llength-LNil llist.collapse*(*1*) **by** *blast*
have *1*: $\bigwedge$*n. n*$\leq$ *llength xss* $\Longrightarrow$ *ridx R* (*lfusecat* (*ltake* (*enat n*) *xss*))
using *assms 0 ridx-lfusecat-ltake-a* **by** *blast*
have *2*: $\bigwedge$*n. n*$\leq$ *llength xss* $\Longrightarrow$
ridx R (*ltake* (*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*))
using *lfusecat-ltake*[*of xss*] *1 assms* **using** *0 llist.collapse*(*1*) **by** *fastforce*
have *30*: $\bigwedge$*n* . (*enat n*)$\leq$ *llength xss* $\Longrightarrow$
(*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) $\leq$
llength (*lfusecat xss*)
proof –
fix *n*::*nat*
assume *a*: *n*$\leq$ *llength xss*
show (*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) $\leq$
llength (*lfusecat xss*)
using *lfusecat-ltake-llength-less-than-llength-lfusecat*[*of xss n*] *0 assms*
a min.order-iff [*of* (*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
llength (*lfusecat xss*)]
by *presburger*
qed
have *3*: $\bigwedge$*n* . (*enat n*)$\leq$ *llength xss* $\Longrightarrow$
(*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) $\leq$
llength (*lfuse* (*lfusecat* (*ltake n xss*)) (*lfusecat* (*ldrop n xss*)))
by (*metis 30 lfusecat-split*)
have *6*: $\bigwedge$*n.* (*Suc n*)$<$ *llength xss* $\Longrightarrow$
(*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) $<$
(*if* (*Suc n*) *=0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*) . *epred*(*llength* (*lnth xss i*))))
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *lfusecat-ltake-llength-less-than-next* **by** *blast*
have *61*: $\bigwedge$*i. i*$<$ *llength xss* $\Longrightarrow$ *epred*(*llength* (*lnth xss i*)) $<$ ∞
by (*metis assms*(*3*) *enat-ord-simps*(*4*) *epred-0 epred-Infty epred-inject i0-ne-infinity*
in-lset-conv-lnth llength-eq-infty-conv-lfinite)
have *62*: $\bigwedge$*n. n* $\leq$ *llength xss* $\Longrightarrow$
(*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))$<$ ∞
by (*metis* (*no-types, lifting*) *30 6 add-diff-cancel-left′ assms*(*3*) *eSuc-enat enat-ord-code*(*4*)
enat-ord-simps(*4*) *ileI1 leD lfusecat-lfinite-a llength-eq-infty-conv-lfinite plus-1-eq-Suc*)
have *7*: $\bigwedge$*k n. k* $\leq$ (*if n =0 then 0 else eSuc*($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
$\wedge$ *n*$\leq$ *llength xss*
$\Longrightarrow$

ridx R (*ltake* (*enat k*) (*lfusecat xss*))
proof −
fix *k n*
show *k* ≤ (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
∧ *n*≤ *llength xss*
⟹*ridx R* (*ltake* (*enat k*) (*lfusecat xss*))
using *ridx-ltake*[*of R* (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
(*lfusecat xss*) *k*]
using *2 30* **by** *presburger*
qed
have *5*: ⋀*k*. *k* < *llength xss* ⟹
ridx R (*ltake* (*enat k*) (*lfusecat xss*))
proof −
fix *k*
show *k* < *llength xss* ⟹
ridx R (*ltake* (*enat k*) (*lfusecat xss*))
proof (*cases lfinite xss*)
case *True*
then show *?thesis*
by (*metis 1 lfinite-llength-enat linorder-le-cases ltake-all ridx-ltake-a*)
next
case *False*
then show *?thesis*
proof −
have *51*: ¬*lfinite*(*lfusecat xss*)
using *assms 0*
by (*metis False iless-Suc-eq lfusecat-lfinite-b lhd-LCons-ltl*
llength-LCons not-lnull-llength one-enat-def)
have *52*: ⋀*k*. (∃ *n*. *enat k* < (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss*
i))))
∧ *n*≤*llength xss* ∧
ridx R (*ltake* (*if n =0 then 0 else*
eSuc(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*)))
proof −
fix *k*
show (∃ *n*. *enat k* < (*if n =0 then 0 else eSuc*(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
∧ *n*≤*llength xss* ∧
ridx R (*ltake* (*if n =0 then 0 else*
eSuc(∑ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*)))
using *2 6*
proof (*induct k*)
case *0*
then show *?case*
proof −
have *521*: (*enat 0*)< *eSuc*(∑ *i* = *0* .. (*1*−*1*) . *epred*(*llength* (*lnth xss i*)))
using *i0-iless-eSuc zero-enat-def* **by** *presburger*
have *522*: (*enat 1*) ≤ *llength xss*
using *False*
using *lnull-imp-lfinite not-lnull-llength one-enat-def* **by** *auto*
have *523*: *ridx R* (*ltake* (*if* (*enat 1*) =*0 then 0 else*

eSuc($\sum$ *i* = *0* .. (*1*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*))
using *0.prems*(*1*) *522 one-enat-def* **by** *fastforce*
show *?thesis* **using** *521 522 523 one-enat-def* **by** *force*
qed
next
case (*Suc k*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *False antisym-conv2 diff-Suc-1 eSuc-enat enat-ord-code*(*4*)
ileI1 llength-eq-infty-conv-lfinite)
qed
qed
have *53*: $\bigwedge$*k*. ($\exists$ *m*. (*enat k*) < *m* $\wedge$ *ridx R*(*ltake* (*enat m*) (*lfusecat xss*)))
proof −
fix *k*
show ($\exists$ *m*. (*enat k*) < *m* $\wedge$ *ridx R* (*ltake* (*enat m*) (*lfusecat xss*)))
proof −
have *54*: ($\exists$ *n*. *enat k* < (*if n* =*0 then 0 else*
eSuc($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*))))
$\wedge$ *n*$\leq$*llength xss* $\wedge$
ridx R (*ltake* (*if n* =*0 then 0 else*
eSuc($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*)))
using *52* **by** *blast*
obtain *n* **where** *55*: *enat k* < (*if n* =*0 then 0 else*
eSuc($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) $\wedge$
n$\leq$*llength xss* $\wedge$
ridx R (*ltake* (*if n* =*0 then 0 else*
eSuc($\sum$ *i* = *0* .. (*n*−*1*) . *epred*(*llength* (*lnth xss i*)))) (*lfusecat xss*))
using *54* **by** *blast*
show *?thesis*
using *55 62* **by** *force*
qed
qed
show *?thesis*
by (*metis 51 53 enat-ord-code*(*4*) *llength-eq-infty-conv-lfinite order.strict-implies-order*
ridx-ltake)
qed
qed
qed
show *?thesis*
by (*metis 1 5 dual-order.refl enat-ord-code*(*4*) *lfinite-conv-llength-enat*
llength-eq-infty-conv-lfinite ltake-all ridx-ltake-all)
qed

lemma *ridx-lfusecat*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss*. *1* < *llength xs*
$\forall$ *xs* $\in$ *lset xss*. *lfinite xs*
shows *ridx R* (*lfusecat xss*) $\longleftrightarrow$ ($\forall$ *i*. *i* < *llength xss* $\longrightarrow$ *ridx R* (*lnth xss i*))
using *ridx-lfusecat-a ridx-lfusecat-b assms*
using *gr-implies-not-zero llength-eq-0* **by** *blast*

lemma *lfusecat-split-lsub*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss. 1* $<$ *llength xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
shows ($\forall$ *i. i*$<$ *llength xss* $\longrightarrow$ *ridx* (λ *a b. f* (*lsub a b* σ)) (*lnth xss i*)) $\longleftrightarrow$
ridx (λ *a b. f* (*lsub a b* σ)) (*lfusecat xss*)
using *assms ridx-lfusecat* **by** *blast*

lemma *lidx-lfuse-lfinite*:
assumes *lfinite xs*
llast xs = *lfirst ys*
shows *lidx* (*lfuse xs ys*) $\longleftrightarrow$ *lidx xs* $\wedge$ *lidx ys*
using *assms*
using *ridx-lfuse-lfinite ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ltake*:
assumes *lidx* (*lfusecat xss*)
n $\leq$ *llength xss*
shows *lidx* (*lfusecat* (*ltake n xss*))
using *assms ridx-lfusecat-ltake ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ldrop*:
assumes *lidx* (*lfusecat xss*)
(*enat n*) $<$ *llength xss*
llastlfirst xss
$\forall$ *xs* $\in$ *lset xss.* $\neg$ *lnull xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
shows *lidx* (*lfusecat* (*ldrop n xss*))
using *assms ridx-lfusecat-ldrop ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-a*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss.* $\neg$ *lnull xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
lidx (*lfusecat xss*)
i $<$ *llength xss*
shows *lidx* (*lnth xss i*)
using *assms ridx-lfusecat-a ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-lfuse*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss.* $\neg$ *lnull xs*
$\forall$ *xs* $\in$ *lset xss. lfinite xs*
$\forall$ *i. i* $<$ *llength xss* $\longrightarrow$ *lidx* (*lnth xss i*)
(*Suc n*) $<$ *llength xss*
shows *lidx* (*lfuse* (*lnth xss n*) (*lnth xss* (*Suc n*)))
using *assms ridx-lfusecat-lfuse ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ltake-a*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss*. $\neg$ *lnull xs*
$\forall$ *xs* $\in$ *lset xss*. *lfinite xs*
$\forall$ *i*. *i* $<$ *llength xss* $\longrightarrow$ *lidx* (*lnth xss i*)
n $\leq$ *llength xss*
shows *lidx* (*lfusecat* (*ltake* (*enat n*) *xss*))
using *assms ridx-lfusecat-ltake-a ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-b*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss*. *1* $<$ *llength xs*
$\forall$ *xs* $\in$ *lset xss*. *lfinite xs*
$\forall$ *i*. *i* $<$ *llength xss* $\longrightarrow$ *lidx* (*lnth xss i*)
shows *lidx* (*lfusecat xss*)
using *assms ridx-lfusecat-b ridx-lidx* **by** *blast*

lemma *lidx-lfusecat*:
assumes *llastlfirst xss*
$\forall$ *xs* $\in$ *lset xss*. *1* $<$ *llength xs*
$\forall$ *xs* $\in$ *lset xss*. *lfinite xs*
shows *lidx* (*lfusecat xss*) $\longleftrightarrow$ ($\forall$ *i*. *i* $<$ *llength xss* $\longrightarrow$ *lidx* (*lnth xss i*))
using *assms ridx-lfusecat ridx-lidx* **by** *blast*

lemma *lnth-sum-expand*:
assumes $\neg$ *lnull xss*
llastlfirst xss
$\forall$ *xs* $\in$ *lset xss*. *1* $<$ *llength xs*
$\forall$ *xs* $\in$ *lset xss*. *lfinite xs*
(*enat i*)$<$ *llength xss*
lfinite xss
shows ($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*))) =
epred(*llength* (*lnth xss i*)) +
($\sum$ *j* $\in$ {*k*. *k*$\neq$*i* $\wedge$ *k*$\leq$ (*the-enat*(*epred*(*llength xss*)))} . *epred*(*llength* (*lnth xss j*)))
proof −
have *1*: {*k*. *k*$\leq$ (*the-enat*(*epred*(*llength xss*)))} = *insert i* {*k*. *k*$\neq$*i* $\wedge$ *k*$\leq$ (*the-enat*(*epred*(*llength xss*)))}
using *assms* **by** *auto*
(*metis eSuc-epred enat-ord-simps*(*1*) *epred-enat gr-implies-not-zero iless-Suc-eq*
lfinite-llength-enat the-enat.simps)
have *2*: {*0*.. (*the-enat*(*epred*(*llength xss*)))} = {*k*. *k*$\leq$ (*the-enat*(*epred*(*llength xss*)))}
by *auto*
have *3*: ($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*))) =
($\sum$ *i* $\in${*k*. *k*$\leq$ (*the-enat*(*epred*(*llength xss*)))} . *epred*(*llength* (*lnth xss i*)))
using *2* **by** *presburger*
have *4*: ($\sum$ *i* $\in${*k*. *k*$\leq$ (*the-enat*(*epred*(*llength xss*)))} . *epred*(*llength* (*lnth xss i*))) =

($\sum$ *j* ∈(*insert i* {*k*. *k*≠*i* ∧ *k*≤ (*the-enat*(*epred*(*llength xss*)))}) . *epred*(*llength* (*lnth xss j*)))
using *1* **by** *presburger*
have *5*: ($\sum$ *j* ∈(*insert i* {*k*. *k*≠*i* ∧ *k*≤ (*the-enat*(*epred*(*llength xss*)))}) . *epred*(*llength* (*lnth xss j*))) =
epred(*llength* (*lnth xss i*)) +
($\sum$ *j* ∈ {*k*. *k*≠*i* ∧ *k*≤ (*the-enat*(*epred*(*llength xss*)))} . *epred*(*llength* (*lnth xss j*)))
by *auto*
show *?thesis*
using *3 4 5* **by** *presburger*
qed

lemma *lfusecat-llength-a*:
assumes *llastlfirst xss*
∀ *xs* ∈ *lset xss*. *1* < *llength xs*
∀ *xs* ∈ *lset xss*. *lfinite xs*
i < *llength xss*
(*enat j*) < *llength* (*lnth xss i*)
shows (*enat j*) < *llength* (*lfusecat xss*)
proof (*cases lfinite xss*)
case *True*
then show *?thesis*
proof −
have *1*: *lnull xss* ⟹ *?thesis*
using *assms*(*4*) *gr-implies-not-zero llength-eq-0* **by** *blast*
have *2*: ¬ *lnull xss* ⟹
llength (*lfusecat xss*) =
eSuc($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*)))
using *True assms lfusecat-llength-lfinite* **by** *fastforce*
have *3*: ¬ *lnull xss* ⟹
llength (*lnth xss i*) ≤
eSuc($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength xss*))) . *epred*(*llength* (*lnth xss i*)))
using *lnth-sum-expand*[*of xss i*] *assms*
by (*metis* (*no-types*, *lifting*) *True co.enat.collapse eSuc-plus gr-implies-not-zero le-iff-add*)
have *4*: ¬ *lnull xss* ⟹ *?thesis*
using *2 3*
by (*metis assms*(*5*) *order-less-le-trans*)
show *?thesis*
using *1 4* **by** *blast*
qed
next
case *False*
then show *?thesis*
proof −
have *5*: ¬ *lfinite* (*lfusecat xss*)
by (*metis False assms*(*1*) *assms*(*2*) *assms*(*3*) *gr-implies-not-zero iless-Suc-eq lfusecat-lfinite-b lhd-LCons-ltl llength-LCons llength-eq-0 not-lnull-llength one-enat-def*)
show *?thesis*
using *5 enat-iless lfinite-conv-llength-enat not-less-iff-gr-or-eq* **by** *blast*
qed
qed

lemma *lmap-lfusecat*:
lmap f (*lfusecat xss*) = (*lfusecat* (*lmap* (*lmap f*) *xss*))
proof (*induct xss*)
case *adm*
then show *?case* **by** *simp*
next
case *LNil*
then show *?case*
by *simp*
next
case (*LCons xs xss*)
then show *?case*
by (*simp add*: *lfuse-def*)
(*metis lmap-eq-LNil lmap-lappend-distrib lnull-def ltl-lmap*)
qed

lemma *lfusecat-is-lfirst-conv*:
assumes $\forall$ *i*. *i*< *llength xss* $\longrightarrow$ *is-lfirst*(*lnth xss i*)
is-lfirst xss
shows *is-lfirst*(*lfusecat xss*)
using *assms*
proof (*induction xss*)
case *adm*
then show *?case*
by (*rule ccpo.admissibleI*) (*auto*, *metis lnth-0 zero-enat-def*)
next
case *LNil*
then show *?case* **by** *simp*
next
case (*LCons xs xss*)
then show *?case*
by (*simp add*: *zero-enat-def*)
qed

1.8 kfilter

lemma *kfilter-code* [*simp*, *code*]:
shows *kfilter-LNil*: *kfilter P n LNil* = *LNil*
and *kfilter-LCons*: *kfilter P n* (*LCons x xs*) =
(*if P x then LCons n* (*kfilter P* (*Suc n*) *xs*) *else kfilter P* (*Suc n*) *xs*)
by (*auto simp add*: *kfilter-def lzip.ctr*(*2*))

lemma *lmap-fst-imp-a*:
assumes *lnull* (*lfilter P xs*)
shows *lnull* (*lmap fst* (*lfilter* (*P* $\circ$ *fst*) (*lzip xs* (*iterates Suc n*))))
using *assms lset-lzipD1* **by** *fastforce*

lemma *lmap-fst-imp-b*:
assumes *lnull* (*lmap fst* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))
shows *lnull* (*lfilter P xs*)
proof −
 have *0*: *lnull* (*lmap fst* (*lfilter* (λ (*x*,*k*). *P x*) (*lzip xs* (*iterates Suc n*))))
  using *assms* **by** *auto*
 have *1*: *lnull* (*lfilter* (λ (*x*,*k*). *P x*) (*lzip xs* (*iterates Suc n*)))
  using *0* **by** *auto*
 have *2*: (∀ (*x*,*k*) ∈ *lset*(*lzip xs* (*iterates Suc n*)). ¬ *P x*)
  using *1* **by** (*simp add*: *prod.case-eq-if*)
 have *3*: (∀ (*x*,*k*) ∈ { (*lnth xs i* , *n*+*i*) | *i*. *enat i* < *llength xs*}. ¬ *P x*)
  using *2* **by** (*simp add*: *lset-lzip*)
 have *4*: (∀ *x* ∈ { *lnth xs i* | *i*. *enat i* < *llength xs*}. ¬ *P x*)
  using *3* **by** *blast*
 from *4* **show** *?thesis* **by** (*simp add*: *lset-conv-lnth*)
qed

lemma *lmap-snd-lnull*:
 lnull (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))) = *lnull*(*lfilter P xs*)
by (*metis llist.collapse*(*1*) *llist.disc*(*1*) *lmap-eq-LNil lmap-fst-imp-a lmap-fst-imp-b*)

lemma *kfilter-lnull-conv*:
 lnull (*kfilter P n xs*) ⟷ (∀ *x* ∈ *lset xs*. ¬ *P x*)
unfolding *kfilter-def* **using** *lmap-snd-lnull*[*of P xs*] *lnull-lfilter*[*of P xs*] **by** *blast*

lemma *kfilter-not-lnull-conv*:
 ¬*lnull* (*kfilter P n xs*) ⟷ (∃ *x* ∈ *lset xs*. *P x*)
by (*simp add*: *kfilter-lnull-conv*)

lemma *lmap-fst-lfilter*:
 lfilter P (*lmap fst* (*lzip xs* (*iterates Suc n*))) =
  lmap fst (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))
using *lfilter-lmap* **by** *blast*

lemma *lmap-fst-lzip*:
 (*lmap fst* (*lzip xs* (*iterates Suc n*))) = *xs*
by (*coinduction arbitrary*: *xs*) (*auto*, *simp add*: *lmap-fst-lzip-conv-ltake*)

lemma *lfilter-kfilter-1*:
 (*lmap fst* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))) =
  (*lfilter P xs*)
by (*metis* (*mono-tags*, *lifting*) *lmap-fst-lfilter lmap-fst-lzip*)

lemma *lfilter-kfilter-snd-llength*:
 llength (*lmap fst* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))) =
  llength (*lmap snd* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))))
by *simp*

lemma *kfilter-llength*:
 llength(*kfilter P n xs*) = *llength*(*lfilter P xs*)

proof −
 have *1*: *llength*(*kfilter P n xs*) = *llength* (*lmap snd* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))))
  by (*simp add*: *kfilter-def*)
 have *2*: *llength* (*lmap snd* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))) =
  llength (*lmap fst* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))))
  using *lfilter-kfilter-snd-llength* **by** *auto*
 have *3*: *llength* (*lmap fst* (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))) =
  llength(*lfilter P xs*)
  by (*metis lfilter-kfilter-1*)
 from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *ldropWhile-LEAST*:
assumes ∃ *n*<*llength xs*. (*Not* ∘ *P*) (*lnth xs n*)
shows *ldropWhile P xs* = *ldropn* (*LEAST n*. *n* < *llength xs* ∧ (*Not* ∘*P*) (*lnth xs n*)) *xs*
proof −
 from *assms* **obtain** *m* **where**
  m< *llength xs* ∧ (*Not* ∘ *P*) (*lnth xs m*)
  ⋀*n*. *n*< *llength xs* ⟶ (*Not* ∘*P*) (*lnth xs n*) ⟹ *m* ≤ *n*
 and ∗: (*LEAST n*. *n* < *llength xs* ∧ (*Not* ∘ *P*) (*lnth xs n*)) = *m*
  by *atomize-elim*
  (*metis* (*no-types*, *lifting*) *dual-order*.*strict-trans1* *enat-ord-code*(*2*) *less-imp-le not-le-imp-less*
   not-less-Least wellorder-Least-lemma(*1*))
 thus *?thesis* **unfolding** ∗
 proof (*induct m arbitrary*: *xs*)
 case *0*
  then show *?case*
   by (*cases xs*) *simp-all*
 next
 case (*Suc m*)
 then show *?case*
  proof −
   have *1*: *ldropWhile P* (*ltl xs*) = *ldropn m* (*ltl xs*) **using** *Suc*
    by (*cases xs*)
     (*simp*,
      metis Suc-le-mono ldrop-eSuc-ltl ldropn-Suc-conv-ldropn ldropn-eq-LNil llist.*simps*(*3*)
      lnth-Suc-LCons ltl-simps(*2*) *not-le-imp-less*)
   have *2*: *P* (*lnth xs 0*)
    using *Suc*.*prems*(*2*) **by** *auto*
   show *?thesis*
   by (*metis 1 2 ldropWhile-LCons ldrop-eSuc-ltl lhd-LCons-ltl llist*.*collapse*(*1*)
    lnth-0-conv-lhd ltl-simps(*1*))
 qed
qed
qed

lemma *ldropWhile-LEAST-not*:
assumes ∃ *n*<*llength xs*. *P* (*lnth xs n*)
shows *ldropWhile* (*Not* ∘ *P*) *xs* = *ldropn* (*LEAST n*. *n* < *llength xs* ∧ *P* (*lnth xs n*)) *xs*
using *assms ldropWhile-LEAST*[*of xs Not* ∘ *P*] **by** *simp*

lemma *ltakeWhile-LEAST*:
assumes $\exists$ *n*<*llength xs*. (*Not* $\circ$ *P*) (*lnth xs n*)
shows (*ltakeWhile P xs*) = (*ltake* (*lleast* (*Not* $\circ$ *P*) *xs*) *xs*)
proof–
from *assms* **obtain** *m* **where**
m< *llength xs* $\wedge$ (*Not* $\circ$ *P*) (*lnth xs m*)
$\bigwedge$*n*. *n*< *llength xs* $\longrightarrow$ (*Not* $\circ$*P*) (*lnth xs n*) $\Longrightarrow$ *m* $\leq$ *n*
and $*$: (*lleast* (*Not* $\circ$ *P*) *xs*) = *m* **unfolding** *lleast-def*
by *atomize-elim*
(*metis* (*no-types*, *lifting*) *Least-le dual-order.trans enat-ord-code*(*2*) *not-less not-less-iff-gr-or-eq*
wellorder-Least-lemma(*1*))
thus *?thesis* **unfolding** $*$
proof (*induct m arbitrary*: *xs*)
case *0*
then show *?case*
proof –
have *ltakeWhile P* (*LCons* (*lnth xs 0*) (*ldropn* (*Suc 0*) *xs*)) = *LNil*
using *0* **by** *fastforce*
then show *?thesis*
by (*metis* (*no-types*) *lappend.disc-iff*(*2*) *lappend-ltakeWhile-ldropWhile ldropn-0*
ldropn-Suc-conv-ldropn llist.collapse(*1*) *lnull-ldropn ltake.ctr*(*1*) *not-less zero-enat-def*)
qed
next
case (*Suc m*)
then show *?case*
proof –
have *1*: *ltakeWhile P* (*ltl xs*) = *ltake m* (*ltl xs*)
using *Suc* **by** (*cases xs*)
(*simp*,
metis Extended-Nat.eSuc-mono Suc-le-mono eSuc-enat llength-LCons lnth-Suc-LCons ltl-simps(*2*))
have *2*: *P* (*lnth xs 0*)
using *Suc* **by** *auto*
show *?thesis*
by (*metis 1 2 eSuc-enat lhd-LCons-ltl lnth-0-conv-lhd ltake.ctr*(*1*) *ltakeWhile.code*
ltake-eSuc-LCons)
qed
qed
qed

lemma *llength-LEAST-not*:
assumes $\exists$ *n* < *llength xs*. (*Not* $\circ$ *P*) (*lnth xs n*)
shows (*lleast* (*Not* $\circ$ *P*) *xs*) < *llength xs*
using *assms* **unfolding** *lleast-def*
by (*metis* (*no-types*, *lifting*) *LeastI*)

lemma *llength-LEAST*:
assumes $\exists$ *n* < *llength xs*. *P* (*lnth xs n*)
shows (*lleast P xs*) < *llength xs*
using *assms llength-LEAST-not*[*of xs Not* $\circ$ *P*] **unfolding** *lleast-def* **by** *simp*

lemma *llength-ltakeWhile-LEAST*:
assumes $\exists$ *n*< *llength xs*. (*Not* $\circ$ *P*) (*lnth xs n*)
shows (*llength* (*ltakeWhile P xs*)) = (*lleast* (*Not* $\circ$ *P*) *xs*)
using *assms ltakeWhile-LEAST*[*of xs P*] **unfolding** *lleast-def*
by (*metis* (*no-types*, *lifting*) *dual-order.strict-trans1 enat-ord-simps*(*2*) *le-cases llength-ltake min-def not-less-Least*)

lemma *llength-ltakeWhile-LEAST-not*:
assumes $\exists$ *n*< *llength xs*. *P* (*lnth xs n*)
shows (*llength* (*ltakeWhile* (*Not* $\circ$*P*) *xs*)) = (*lleast P xs*)
using *assms llength-ltakeWhile-LEAST*[*of xs Not* $\circ$ *P*] **unfolding** *lleast-def* **by** *simp*

lemma *lzip-ldropWhile-fst*:
assumes *llength* (*lzip xs ys*) = *llength xs*
shows *lzip* (*ldropWhile P xs*) (*lmap snd* (*ldropWhile* (*P* $\circ$ *fst*) (*lzip xs ys*))) =
ldropWhile (*P* $\circ$ *fst*) (*lzip xs ys*)
using *assms*
proof –
have *f1*: *ldropWhile P xs* = *lmap fst* (*ldropWhile* (*P* $\circ$ *fst*) (*lzip xs ys*))
by (*metis* (*no-types*) *llength-lzip assms ldropWhile-lmap lmap-fst-lzip-conv-ltake ltake-all order-refl*)
have *ltake* (*min* (*llength* (*ldrop* (*llength* (*ltakeWhile* (*P* $\circ$ *fst*) (*lzip xs ys*))) *xs*))
(*llength* (*ldrop* (*llength* (*ltakeWhile* (*P* $\circ$ *fst*) (*lzip xs ys*))) *ys*)))
(*ldrop* (*llength* (*ltakeWhile* (*P* $\circ$ *fst*) (*lzip xs ys*))) (*lzip xs ys*)) =
ldrop (*llength* (*ltakeWhile* (*P* $\circ$ *fst*) (*lzip xs ys*))) (*lzip xs ys*)
by (*simp add*: *ltake-all*)
then show *?thesis*
using *f1* **by** (*simp add*: *ldropWhile-eq-ldrop lmap-fst-lzip-conv-ltake lmap-snd-lzip-conv-ltake ltake-lzip*)
qed

lemma *lzip-ldropWhile-fst-iterates*:
ldropWhile (*Not* $\circ$ *P* $\circ$ *fst*) (*lzip xs* (*iterates Suc n*)) =
lzip (*ldropWhile* (*Not* $\circ$*P*) *xs*) (*lmap snd* (*ldropWhile* (*Not* $\circ$*P* $\circ$ *fst*) (*lzip xs* (*iterates Suc n*))))
by (*metis llength-lmap lmap-fst-lzip lzip-ldropWhile-fst*)

lemma *ldropWhile-iterates-split*:
assumes $\exists$ *n*<*llength xs*. *P* (*lnth xs n*)
shows ((*ldropWhile* (*Not* $\circ$*P* $\circ$ *fst*) (*lzip xs* (*iterates Suc n*)))) =
(*lzip* (*ldropWhile* (*Not* $\circ$*P*) *xs*)
(*iterates Suc* (*n*+(*lleast P xs*))))
proof –
have *1*: $\exists$*na*. *enat na* < *llength* (*lzip xs* (*iterates Suc n*)) $\wedge$
(*Not*$\circ$ (*Not*$\circ$ *P*) $\circ$ *fst*) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)
using *lmap-fst-lzip*[*of xs n*]
by (*metis assms comp-apply llength-lmap lnth-lmap*)
have *2*: (*ldropWhile* ((*Not* $\circ$*P*) $\circ$ *fst*) (*lzip xs* (*iterates Suc n*))) =
(*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) $\wedge$
((*Not* $\circ$ (*Not* $\circ$*P*$\circ$ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*lzip xs* (*iterates Suc n*)))
using *1 ldropWhile-LEAST*[*of* (*lzip xs* (*iterates Suc n*)) (*Not* $\circ$*P*) $\circ$ *fst*] **by** *auto*
have *3*: (*ldropn* (*LEAST na*. *na* < *llength*(*lzip xs* (*iterates Suc n*)) $\wedge$

((*Not* ∘ (*Not* ∘*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*lzip xs* (*iterates Suc n*))) =
(*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*lzip xs* (*iterates Suc n*)))
by *auto*
have *4*: (*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*lzip xs* (*iterates Suc n*))) =
(*lzip* (*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) *xs*)
(*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*iterates Suc n*)))
using *ldropn-lzip* **by** *blast*
have *5*: (*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*iterates Suc n*)) =
(*iterates Suc* (*n*+(*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*))))
by (*metis* (*no-types*, *lifting*) *funpow-Suc-conv ldropn-iterates*)
have *6*: (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) =
(*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P* ∘*fst*))) (*lnth* (*xs*) *na*, *lnth* (*iterates Suc n*) *na*))
by (*metis* (*no-types*, *opaque-lifting*) *comp-def fst-conv lmap-fst-lzip lnth-lmap*)
have *7*: *llength*(*lzip xs* (*iterates Suc n*)) = *llength xs*
by *simp*
have *8*: (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P* ∘*fst*))) (*lnth* (*xs*) *na*, *lnth* (*iterates Suc n*) *na*)) =
(*LEAST na. na* < *llength xs* ∧ *P* (*lnth xs na*))
by *auto*
have *9*: (*lzip* (*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) *xs*)
(*ldropn* (*LEAST na. na* < *llength*(*lzip xs* (*iterates Suc n*)) ∧
(((*P*∘ *fst*))) (*lnth* (*lzip xs* (*iterates Suc n*)) *na*)) (*iterates Suc n*))) =
(*lzip* (*ldropn* (*LEAST na. na* < *llength xs* ∧ *P* (*lnth xs na*)) *xs*)
(*iterates Suc* (*n*+(*LEAST na. na* < *llength xs* ∧ *P* (*lnth xs na*)))))
using *5 6* **by** *auto*
show *?thesis* **unfolding** *lleast-def*
using *assms 2 3 4 9 ldropWhile-LEAST-not*[*of xs P*] **by** *presburger*
qed

lemma *kfilter-ldropWhile* :
assumes ¬ *lnull*(*kfilter P n xs*)
shows *kfilter P n xs* =
(*LCons* (*n*+ (*lleast P xs*))
(*lmap snd* (*lfilter* (*P*∘*fst*)
(*lzip* (*ldrop* (*Suc* (*lleast P xs*)) (*xs*))
(*iterates Suc* (*Suc* (*n*+(*lleast P xs*))))))))
proof −
let *?Least* = (*lleast P xs*)
have *1*: *lhd*(*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))) =
lhd (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))
by *simp*

have *2*: *ltl*(*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))) =
(*lfilter* (*P*∘*fst*) (*ltl* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))))
by (*simp add*: *ltl-lfilter*)
have *3*: *lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)) =
(*LCons* (*lhd*(*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*))))
(*ltl*(*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))))
by (*metis assms kfilter-llength llength-eq-0 llength-lmap llist.disc*(*1*) *llist.exhaust-sel lmap-fst-lfilter lmap-fst-lzip*)
have *4*: *kfilter P n xs* =
lmap snd (*lfilter* (*P*∘*fst*) (*lzip xs* (*iterates Suc n*)))
by (*simp add*: *kfilter-def*)
have *5*: *ltl* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))) =
ltl((*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*)
(*lmap snd* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))))
by (*metis lzip-ldropWhile-fst-iterates*)
have *6*: *ltl*((*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*)
(*lmap snd* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))))) =
(*lzip* (*ltl* (*ldropWhile* (*Not* ∘*P*) *xs*))
(*ltl* (*lmap snd* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))))
using *lzip-ldropWhile-fst-iterates*[*of P xs n*]
ltl-lzip[*of* (*ldropWhile* (*Not* ∘ *P*) *xs*)
(*lmap snd* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))]
by *force*
have *7*: *lmap snd* (
(*LCons* (*lhd* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))
(*lfilter* (*P*∘*fst*) (*ltl* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))))) =
(*LCons* (*snd* (*lhd* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))))
(*lmap snd* (*lfilter* (*P*∘*fst*) (*ltl* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*)))))))
by *simp*
have *8*: ∃ *n*. *enat n* < *llength xs* ∧ *P* (*lnth xs n*)
by (*metis assms in-lset-conv-lnth kfilter-llength llength-eq-0 lnull-lfilter*)
have *9*: (*snd* (*lhd* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))))) =
(*snd* (*lhd* (*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*)
(*iterates Suc* (*n*+*?Least*)))))
using *8 ldropWhile-iterates-split*[*of xs P n*] **by** *simp*
have *10*: (*snd* (*lhd* (*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*)
(*iterates Suc* (*n*+*?Least*))))) =
lhd (*iterates Suc* (*n*+*?Least*))
by (*metis* (*no-types*, *lifting*) *3 comp-assoc iterates.disc-iff lfilter-eq-LCons lhd-lzip llist.disc*(*2*) *lzip.disc*(*1*) *lzip-ldropWhile-fst-iterates snd-conv*)
have *11*: *lhd* (*iterates Suc* (*n*+*?Least*)) = (*n*+*?Least*)
by *simp*
have *12*: *ltl* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))) =
ltl (*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*)
(*iterates Suc* (*n*+*?Least*)))
using *8 ldropWhile-iterates-split*[*of xs P n*] **by** *simp*
have *13*: *ltl* (*lzip* (*ldropWhile* (*Not* ∘*P*) *xs*) (*iterates Suc* (*n*+*?Least*))) =
(*lzip* (*ltl* (*ldropWhile* (*Not* ∘*P*) *xs*)) (*ltl* (*iterates Suc* (*n*+*?Least*))))
by (*metis* (*no-types*, *lifting*) *3 comp-assoc iterates.disc-iff lfilter-eq-LCons llist.discI*(*2*) *ltl-lzip lzip.disc-iff*(*2*) *lzip-ldropWhile-fst-iterates*)

have *14*: (*ltl* (*iterates Suc* (*n*+*?Least*))) = (*iterates Suc* (*Suc* (*n*+*?Least*)))
by *simp*
have *15*: *ltl* (*ldropWhile* (*Not* ∘*P*) *xs*) = *ltl* (*ldrop* (*llength* (*ltakeWhile* (*Not* ∘*P*) *xs*)) *xs*)
by (*simp add*: *ldropWhile-eq-ldrop*)
have *16*: *ltl* (*ldrop* (*llength* (*ltakeWhile* (*Not* ∘*P*) *xs*)) *xs*) =
ldrop (*eSuc* (*llength* (*ltakeWhile* (*Not* ∘*P*) *xs*))) (*xs*)
by (*simp add*: *ldrop-eSuc-conv-ltl*)
have *17*: (*llength* (*ltakeWhile* (*Not* ∘*P*) *xs*)) = (*llength* (*ltake* *?Least xs*))
using *8 ltakeWhile-LEAST*[*of xs Not* ∘*P*] **unfolding** *lleast-def* **by** *auto*
have *18*: (*llength* (*ltake* *?Least xs*)) = *enat* *?Least*
using *17 8 llength-ltakeWhile-LEAST-not* **unfolding** *lleast-def* **by** *fastforce*
have *19*: *ldrop* (*eSuc* (*llength* (*ltakeWhile* (*Not* ∘*P*) *xs*))) (*xs*) = *ldrop* (*Suc* *?Least*) (*xs*)
using *17 18* **by** (*simp add*: *eSuc-enat*)
have *20*: *ltl* (*ldropWhile* (*Not* ∘ *P* ∘ *fst*) (*lzip xs* (*iterates Suc n*))) =
(*lzip* (*ldrop* (*Suc* *?Least*) (*xs*)) (*iterates Suc* (*Suc* (*n*+*?Least*))))
using *12 13 15 16 19* **by** *auto*
show *?thesis*
using *2 3 4 10 20 9* **by** *auto*
qed

lemma *kfilter-eq-LCons*:
kfilter P n xs = *LCons x xs*′ ⟹
x = *n*+(*lleast P xs*) ∧
xs′ = (*lmap snd* (*lfilter* (*P*∘*fst*)
(*lzip* (*ldrop* (*Suc* (*lleast P xs*)) (*xs*))
(*iterates Suc* (*Suc* (*n*+(*lleast P xs*)))))))
using *kfilter-ldropWhile*[*of P n xs*] **by** *auto*

lemma *kfilter-eq-LCons-1*:
kfilter P n xs = *LCons x xs*′ ⟹
x = (*n*+(*lleast P xs*)) ∧
xs′ = *kfilter P* (*Suc* (*n*+(*lleast P xs*))) (*ldrop* (*Suc* (*lleast P xs*)) *xs*)
using *kfilter-eq-LCons*[*of P n xs x xs*′]
kfilter-def[*of P* (*Suc* (*n* + (*lleast P xs*)))
(*ldrop* (*Suc* (*lleast P xs*)) *xs*)]
by *auto*

lemma *kfilter-eq-conv*:
kfilter P n xs = *LNil* ∨
kfilter P n xs =
LCons (*n*+(*lleast P xs*)) (*kfilter P* (*Suc* (*n*+(*lleast P xs*))) (*ldrop* (*Suc* (*lleast P xs*)) *xs*))
proof (*cases kfilter P n xs*)
case *LNil*
then show *?thesis* **by** *simp*
next
case (*LCons x21 x22*)
then show *?thesis*
proof −
let *?Least* = (*lleast P xs*)
have *1*: *x21* = *n*+*?Least*

using *kfilter-eq-LCons-1*[*of P n xs x21 x22*] **by** (*meson LCons*)
have *2*: *x22*= (*kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*))
using *kfilter-eq-LCons-1*[*of P n xs x21 x22*] **by** (*meson LCons*)
show *?thesis*
by (*metis 1 2 LCons*)
qed
qed

lemma *kfilter-lnth-zero*:
assumes ¬*lnull*(*kfilter P n xs*)
shows *lnth* (*kfilter P n xs*) *0* = *n*+ (*lleast P xs*)
using *assms*
by (*metis kfilter-eq-LCons-1 lhd-LCons-ltl lnth-0-conv-lhd*)

lemma *length-LEAST-a*:
assumes ¬*lnull*(*kfilter P n xs*)
shows *lleast P xs*< *llength xs*
using *assms exist-lset-lnth*[*of xs P*] *kfilter-not-lnull-conv*[*of P n xs*]
llength-LEAST[*of xs P*] **by** *blast*

lemma *kfilter-upperbound*:
assumes *i* < *llength*(*kfilter P n xs*)
shows (*lnth* (*kfilter P n xs*) *i*) < *n* + *llength xs*
proof (*cases lfinite* (*kfilter P n xs*))
case *True*
then show *?thesis* **using** *assms*
proof (*induct zs*≡*kfilter P n xs arbitrary*: *xs n i rule*: *lfinite-induct*)
case (*LNil xs*)
then show *?case*
using *gr-implies-not-zero llength-lnull* **by** *blast*
next
case (*LCons xs*)
then show *?case*
proof −
let *?Least* = (*lleast P xs*)
have *1*: ∃ *x xs′*. *kfilter P n xs* = *LCons x xs′*
using *LCons.hyps*(*2*) *kfilter-ldropWhile* **by** *blast*
obtain *x xs′* **where** *2*:*kfilter P n xs* = *LCons x xs′*
using *1* **by** *auto*
have *3*: *x* = *n*+*?Least*
using *kfilter-ldropWhile*[*of P n xs*] **by** (*simp add*: *2*)
have *4*: *i*=*0* ⟹ (*lnth* (*kfilter P n xs*) *i*) = *x*
by (*simp add*: *2*)
have *5*: *enat n*+ *enat ?Least* < *enat n* + *llength xs*
using *length-LEAST-a*[*of P n xs*] *LCons.hyps*(*2*) *enat-add-mono* **by** *blast*
have *6*: *i*=*0* ⟹ *enat* (*lnth* (*kfilter P n xs*) *i*) < *enat n* + *llength xs*
using *3 4 5* **by** *auto*
have *7*: *xs′* = *kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*)
using *kfilter-ldropWhile*[*of P n xs*] *2 kfilter-eq-LCons-1* **by** *blast*
have *8*: *i*>*0* ⟹ *enat* (*lnth* (*kfilter P n xs*) *i*) = *enat* (*lnth xs′* (*i*−*1*))

by (*simp add*: *2 lnth-LCons'*)
have *9*: *i>0 ∧ lnull xs' ⟹ enat (lnth (kfilter P n xs) i) < enat n + llength xs*
by (*metis 2 LCons.prems*(*1*) *One-nat-def enat-ord-simps*(*2*) *llength-LCons llength-LNil llist.collapse*(*1*) *not-less-eq one-eSuc one-enat-def*)
have *10*: *i>0 ∧ ¬lnull xs' ⟹ (i−1) < llength xs'*
using *2 LCons.prems*(*1*) *Suc-ile-eq* by *fastforce*
have *11*: *i>0 ∧ ¬lnull xs' ⟹*
enat (lnth xs' (i−1)) < enat (Suc (n+?Least)) + llength (ldrop (Suc ?Least) xs)
by (*metis* (*no-types*, *lifting*) *10 2 7 LCons.hyps*(*3*) *ltl-simps*(*2*))
have *111*: *lappend (ltake (Suc (?Least)) xs) (ldrop (Suc (?Least)) xs) = xs*
using *lappend-ltake-ldrop* by *blast*
have *112*: *enat (Suc (n+?Least)) = n + (Suc ?Least)*
by *auto*
have *113*: *Suc ?Least ≤ (llength xs)*
using *5 Suc-ile-eq enat-add-mono* by *blast*
have *114*: *llength (ltake (Suc (?Least)) xs) = Suc ?Least*
using *llength-ltake*[*of (Suc (?Least)) xs*] *113* by *linarith*
have *12*: *enat (Suc (n+?Least)) + llength (ldrop (Suc ?Least) xs) ≤ enat n + llength xs*
using *llength-lappend*[*of (ltake (Suc (?Least)) xs) (ldrop (Suc (?Least)) xs)*]
by (*metis* (*no-types*, *lifting*) *111 112 114 eq-refl group-cancel.add1 plus-enat-simps*(*1*))
have *14*: *i>0 ∧ ¬lnull xs' ⟹ enat (lnth (kfilter P n xs) i) < enat n + llength xs*
using *11 12 8* by *auto*
have *15*: *i>0 ⟹ enat (lnth (kfilter P n xs) i) < enat n + llength xs*
using *14 9 dual-order.strict-implies-order* by *blast*
show *?thesis*
using *15 6* by *blast*
qed
qed
next
case *False*
then show *?thesis*
by (*metis enat-ord-simps*(*4*) *iadd-le-enat-iff kfilter-llength leD leI llength-eq-enat-lfiniteD llength-eq-infty-conv-lfinite llength-lfilter-ile*)
qed

lemma *kfilter-lowerbound*:
assumes *i < llength(kfilter P n xs)*
shows *n ≤(lnth (kfilter P n xs) i)*
using *assms*
proof (*induct i arbitrary*: *xs n*)
case *0*
then show *?case*
using *kfilter-ldropWhile zero-enat-def* by *fastforce*
next
case (*Suc i*)
then show *?case*
proof −
let *?Least = lleast P xs*
have *7*: *∃ x xs'. (kfilter P n xs) = LCons x xs'*
using *Suc.prems gr-implies-not-zero kfilter-ldropWhile llength-lnull* by *blast*

```
    obtain x xs' where 8: (kfilter P n xs) = LCons x xs'
      using 7 by auto
    have 9: lnth (kfilter P n xs) (Suc i) = lnth xs' i
      by (simp add: 8)
    have 10: xs' = kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)
      using kfilter-ldropWhile[of P n xs] 8 kfilter-eq-LCons-1 by blast
    have 11: enat i < llength (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs))
      by (metis 10 8 Extended-Nat.eSuc-mono Suc.prems(1) eSuc-enat llength-LCons)
    have 12: lnull xs' ⟹ n ≤ lnth (kfilter P n xs) (Suc i)
      using 10 11 gr-implies-not-zero llength-eq-0 by blast
    have 13: ¬lnull xs' ⟹ (Suc (n+?Least)) ≤ lnth xs' i
      using 10 11 Suc.hyps by blast
    have 14: ¬ lnull xs' ⟹ n ≤ lnth (kfilter P n xs) (Suc i)
      using 13 9 by linarith
    show ?thesis using 12 14 by blast
  qed
qed

lemma kfilter-mono:
  assumes (Suc i) < llength(kfilter P n xs)
  shows (lnth (kfilter P n xs) i) < (lnth (kfilter P n xs) (Suc i))
using assms
  proof (induct i arbitrary: xs n)
  case 0
  then show ?case
   proof –
    let ?Least = lleast P xs
    have 1: ∃ x xs'. kfilter P n xs = LCons x xs'
      using 0.prems gr-implies-not-zero kfilter-ldropWhile llength-lnull by blast
    obtain x xs' where 2:kfilter P n xs = LCons x xs'
      using 1 by auto
    have 3: x = n+?Least
      using kfilter-ldropWhile[of P n xs] by (simp add: 2)
    have 4: lnth (kfilter P n xs) 0 = n+?Least
      by (simp add: 2 3)
    have 5: lnth (kfilter P n xs) (Suc 0) = lnth xs' 0
      by (simp add: 2)
    have 6: xs' = kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)
      using kfilter-ldropWhile[of P n xs] 2 kfilter-eq-LCons-1 by blast
    have 7: n+?Least < lnth xs' 0
      by (metis 0.prems 2 6 Extended-Nat.eSuc-mono One-nat-def Suc-le-lessD kfilter-lowerbound
          llength-LCons one-eSuc one-enat-def zero-enat-def)
    show ?thesis by (simp add: 4 5 7)
  qed
  next
  case (Suc i)
  then show ?case
   proof –
    let ?Least = lleast P xs
    have 8: ∃ x xs'. kfilter P n xs = LCons x xs'
```

using *Suc.prems gr-implies-not-zero kfilter-ldropWhile llength-lnull* **by** *blast*
obtain *x xs′* **where** *9*:*kfilter P n xs = LCons x xs′*
using *8* **by** *auto*
have *10*: *xs′ = kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)*
using *kfilter-ldropWhile[of P n xs] 9 kfilter-eq-LCons-1* **by** *blast*
have *11*: *lnth (kfilter P n xs) (Suc (Suc i)) = lnth xs′ (Suc i)*
by (*simp add*: *9*)
have *12*: *lnth (kfilter P n xs) (Suc i) < lnth xs′ (Suc i)*
by (*metis (no-types, lifting) 10 9 Extended-Nat.eSuc-mono Suc(1) Suc(2) eSuc-enat llength-LCons lnth-Suc-LCons*)
show *?thesis* **by** (*simp add*: *11 12*)
qed
qed

lemma *lfilter-kfilter*:
assumes *i < llength(kfilter P n xs)*
shows (*lnth xs ((lnth (kfilter P n xs) i)−n)) = (lnth (lfilter P xs) i)*
using *assms*
proof (*induct i arbitrary*: *xs n*)
case *0*
then show *?case*
proof −
let *?Least = lleast P xs*
have *1*: *lnth (kfilter P n xs) 0 = n +?Least*
using *0.prems kfilter-ldropWhile zero-enat-def* **by** *fastforce*
have *2*: (*lnth xs ((lnth (kfilter P n xs) 0)−n)) =*
(*lnth xs ?Least*)
by (*simp add*: *1*)
have *3*: (*lnth (lfilter P xs) 0) = lhd (ldropWhile (Not ∘ P) xs)*
by (*metis (full-types) 0.prems kfilter-llength lhd-conv-lnth lhd-lfilter llength-eq-0 not-iless0*)
have *4*: ∃ *n<llength xs. P (lnth xs n)*
by (*metis 0.prems dual-order.irrefl in-lset-conv-lnth kfilter-llength llength-eq-0 lnull-lfilter zero-enat-def*)
have *5*: (*ldropWhile (Not ∘ P) xs) = ldropn ?Least xs*
using *ldropWhile-LEAST-not[of xs P] 4* **unfolding** *lleast-def* **by** *blast*
have *6*: *lhd (ldropn ?Least xs) = (lnth xs ?Least)*
by (*simp add*: *4 lhd-ldropn llength-LEAST*)
show *?thesis* **using** *2 3 5 6* **by** *auto*
qed
next
case (*Suc i*)
then show *?case*
proof −
let *?Least= lleast P xs*
have *7*: ∃ *x xs′. (kfilter P n xs) = LCons x xs′*
using *Suc.prems gr-implies-not-zero kfilter-ldropWhile llength-lnull* **by** *blast*
obtain *x xs′* **where** *8*: (*kfilter P n xs) = LCons x xs′*
using *7* **by** *auto*
have *9*: *lnth (kfilter P n xs) (Suc i) = lnth xs′ i*
by (*simp add*: *8*)

have *10*: $xs' = kfilter\ P\ (Suc\ (n+?Least))\ (ldrop\ (Suc\ ?Least)\ xs)$
using *kfilter-ldropWhile*[*of P n xs*] **by** (*simp add*: *8 kfilter-eq-LCons-1*)
have *11*: $enat\ i < llength\ (kfilter\ P\ (Suc\ (n+?Least))\ (ldrop\ (Suc\ ?Least)\ xs))$
by (*metis 10 8 Extended-Nat.eSuc-mono Suc.prems*(*1*) *eSuc-enat llength-LCons*)
have *12*: $lnull\ xs' \Longrightarrow$
$lnth\ xs\ (lnth\ (kfilter\ P\ n\ xs)\ (Suc\ i) - n) = lnth\ (lfilter\ P\ xs)\ (Suc\ i)$
by (*metis 10 11 llength-LNil llist.collapse*(*1*) *not-less-zero*)
have *13*: $\neg lnull\ xs' \Longrightarrow$
$lnth\ (ldrop\ (Suc\ ?Least)\ xs)\ (lnth\ xs'\ i - (Suc\ (n+?Least))) =$
$lnth\ (lfilter\ P\ (ldrop\ (Suc\ ?Least)\ xs))\ i$
by (*metis* (*no-types*, *lifting*) *10 11 Suc.hyps*)
have *14*: $(lnth\ (lfilter\ P\ xs)\ (Suc\ i)) = (lnth\ (ltl(lfilter\ P\ xs))\ i)$
by (*metis 8 kfilter-not-lnull-conv llist.disc*(*2*) *lnth-ltl lnull-lfilter*)
have *15*: $(ltl(lfilter\ P\ xs)) = lfilter\ P\ (ltl\ (ldropWhile\ (Not \circ P)\ xs))$
using *ltl-lfilter* **by** *blast*
have *16*: $(ltl\ (ldropWhile\ (Not \circ P)\ xs)) =$
$(ltl\ (ldropn\ (LEAST\ n.\ n < llength\ xs \wedge (Not \circ (Not \circ P))\ (lnth\ xs\ n)\ )\ xs))$
using *ldropWhile-LEAST*[*of xs Not* ∘ *P*]
by (*metis* (*no-types*, *lifting*) *8 comp-apply in-lset-conv-lnth kfilter-lnull-conv llist.disc*(*2*))
have *17*: $(ltl\ (ldropn\ (LEAST\ n.\ n < llength\ xs \wedge (Not \circ (Not \circ P))\ (lnth\ xs\ n)\ )\ xs)) =$
$(ltl\ (ldropn\ ?Least\ xs))$
unfolding *lleast-def* **by** *auto*
have *18*: $(ltl\ (ldropn\ ?Least\ xs)) =$
$ldropn\ (Suc\ ?Least)\ xs$
by (*simp add*: *ldropn-ltl ltl-ldropn*)
have *19*: $ldropn\ (Suc\ ?Least)\ xs =$
$ldrop\ (Suc\ ?Least)\ xs$
by (*simp add*: *ldrop-enat*)
have *20*: $lnth\ (lfilter\ P\ xs)\ (Suc\ i) = lnth\ (lfilter\ P\ (ldrop\ (Suc\ ?Least)\ xs))\ i$
using *14 15 16 18 19* **unfolding** *lleast-def* **by** *auto*
have *21*: $lnth\ xs\ (lnth\ (kfilter\ P\ n\ xs)\ (Suc\ i) - n) = lnth\ xs\ (lnth\ xs'\ i - n)$
by (*simp add*: *10 9*)
have *22*: $n \leq lnth\ xs'\ i$
using *9 Suc.prems*(*1*) *kfilter-lowerbound* **by** *fastforce*
have *23*: $(Suc\ (n+?Least)) \leq lnth\ (kfilter\ P\ (Suc\ (n+?Least))\ (ldrop\ (Suc\ ?Least)\ xs))\ i$
using *kfilter-lowerbound*[*of i P Suc* (*n*+*?Least*) (*ldrop* (*Suc ?Least*) *xs*)] *11* **by** *force*
have *24*: $(Suc\ ?Least) > llength\ (ltake\ ?Least\ xs)$
by (*simp add*: *min.strict-coboundedI1*)
have *25*: $(ldrop\ (Suc\ ?Least)\ (lappend\ (ltake\ ?Least\ xs)\ (ldrop\ ?Least\ xs))) =$
$ldrop\ ((Suc\ ?Least) - llength(ltake\ ?Least\ xs)\ )\ (ldrop\ ?Least\ xs)$
by (*simp add*: *ldrop-lappend*)
have *26*: $lappend\ (ltake\ (Suc\ ?Least)\ xs)\ (ldrop\ (Suc\ ?Least)\ xs) = xs$
by (*simp add*: *lappend-ltake-ldrop*)
have *27*: $(Suc\ ?Least) \leq (lnth\ xs'\ i - n)$
using *10 23* **by** *auto*
have *271*: $lnth\ xs'\ i - n - (Suc\ ?Least) = lnth\ xs'\ i - (Suc\ (n+?Least))$
by *auto*
have *28*: $lnth\ (lappend\ (ltake\ (Suc\ ?Least)\ xs)\ (ldrop\ (Suc\ ?Least)\ xs))\ (lnth\ xs'\ i - n) =$
$lnth\ (ldrop\ (Suc\ ?Least)\ xs)\ (lnth\ xs'\ i - (Suc\ (n+?Least)))$
using *lnth-lappend*[*of* (*ltake* (*Suc ?Least*) *xs*) (*ldrop* (*Suc ?Least*) *xs*) (*lnth xs' i* −*n*)]

27 271 11 19
by (*metis* (*no-types*, *lifting*) *gr-implies-not-zero kfilter-LNil ldropn-eq-LNil le-cases llength-lnull llength-ltake llist.disc*(*1*) *lnth-lappend2 min-def*)
have *29*: *lnth xs* (*lnth xs'* *i* −*n*) = *lnth* (*ldrop* (*Suc ?Least*) *xs*) (*lnth xs'* *i* − (*Suc* (*n*+*?Least*)))
using *28* **by** (*metis* (*no-types*, *lifting*) *26*)
have *30*: ¬*lnull xs'* ⟹ *lnth xs* (*lnth* (*kfilter P n xs*) (*Suc i*) − *n*) = *lnth* (*lfilter P xs*) (*Suc i*)
by (*metis 13 20 21 29*)
show *?thesis*
using *12 30* **by** *blast*
qed
qed

lemma *in-kfilter-lset*:
shows *x* ∈ *lset* (*kfilter P n xs*) ⟷ *x* ∈ { *n*+*i* | *i*. *i*<*llength xs* ∧ *P* (*lnth xs i*) }
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
thus *?rhs*
proof (*induct zs*≡*kfilter P n xs arbitrary*: *n xs rule*:*llist-set-induct*)
case (*find*)
then show *?case*
proof −
let *?Least* = *lleast P xs*
have *1*: *lhd* (*kfilter P n xs*) = *n*+*?Least*
using *kfilter-ldropWhile*[*of P n xs*] *find* **by** *auto*
have *2*: *P* (*lnth xs ?Least*) **unfolding** *lleast-def*
by (*metis* (*mono-tags*, *lifting*) *LeastI exist-lset-lnth find.hyps kfilter-lnull-conv*)
have *3*: *?Least* < *llength xs*
by (*metis find.hyps length-LEAST-a*)
have *4*: *n*+*?Least* ∈ {*n* + *i* |*i*. *enat i* < *llength xs* ∧ *P* (*lnth xs i*)}
using *2 3* **by** *blast*
show *?thesis* **using** *1 4* **by** *auto*
qed
next
case (*step y*)
then show *?case*
proof −
let *?Least* = *lleast P xs*
have *5*: *ltl* (*kfilter P n xs*) = *kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*)
using *kfilter-ldropWhile*[*of P n xs*]
by (*metis kfilter-def ltl-simps*(*2*) *step.hyps*(*1*))
have *6*: *lnull* (*kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*)) ⟹ *y* ∈ {*n* + *i* |*i*. *enat i* < *llength xs* ∧ *P* (*lnth xs i*)}
by (*metis 5 gr-implies-not-zero in-lset-conv-lnth llength-eq-0 step.hyps*(*2*))
have *7*: ¬ *lnull* (*kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*)) ⟹ *y* ∈ {(*Suc* (*n*+*?Least*)) + *i* |*i*. *enat i* < *llength* (*ldrop* (*Suc ?Least*) *xs*) ∧ *P* (*lnth* (*ldrop* (*Suc ?Least*) *xs*) *i*)}
using *step.hyps* **using** *5* **by** *blast*

have *8*: (*Suc* (*n*+*?Least*)) = *n* + *Suc ?Least*
by *auto*
have *9*: ¬ *lnull*(*kfilter P* (*Suc* (*n* + *?Least*)) (*ldrop* (*enat* (*Suc ?Least*)) *xs*)) ⟹
∃ *i*. *y* = *n* + *Suc* (*?Least*) + *i* ∧ *enat i* < *llength* (*ldrop* (*enat* (*Suc ?Least*)) *xs*) ∧
P (*lnth* (*ldrop* (*enat* (*Suc ?Least*)) *xs*) *i*) ⟹
∃ *i*. *y* = *n* + *i* ∧ *enat i* < *llength xs* ∧ *P* (*lnth xs i*)
proof −
assume *a0*: ¬ *lnull*(*kfilter P* (*Suc* (*n* + *?Least*)) (*ldrop* (*enat* (*Suc ?Least*)) *xs*))
assume *a1*: ∃ *i*. *y* = *n* + *Suc* (*?Least*) + *i* ∧ *enat i* < *llength* (*ldrop* (*enat* (*Suc ?Least*)) *xs*) ∧
P (*lnth* (*ldrop* (*enat* (*Suc ?Least*)) *xs*) *i*)
show ∃ *i*. *y* = *n* + *i* ∧ *enat i* < *llength xs* ∧ *P* (*lnth xs i*)
proof −
obtain *i* **where** *10*: *y* = *n* + *Suc* (*?Least*) + *i* ∧ *enat i* < *llength* (*ldrop* (*enat* (*Suc ?Least*)) *xs*) ∧
P (*lnth* (*ldrop* (*enat* (*Suc ?Least*)) *xs*) *i*)
using *a1* **by** *auto*
have *11*: *Suc* (*?Least*) + *i* < *llength xs*
by (*metis 10 add.commute ldrop-enat ldrop-ldrop leD le-less-linear lnull-ldropn*
plus-enat-simps(*1*))
have *12*: *P* (*lnth xs* (*Suc* (*?Least*) + *i*))
by (*metis 10 11 add.commute ldrop-enat lnth-ldropn*)
show *?thesis*
using *10 11 12 ab-semigroup-add-class.add-ac*(*1*) **by** *blast*
qed
qed
have *13*: ¬ *lnull* (*kfilter P* (*Suc* (*n*+*?Least*)) (*ldrop* (*Suc ?Least*) *xs*)) ⟹
y ∈ {*n* + *i* |*i*. *enat i* < *llength xs* ∧ *P* (*lnth xs i*)}
using *7 9* **by** *auto*
show *?thesis*
using *13 6* **by** *blast*
qed
qed
next
assume *?rhs*
then obtain *i* **where** *15*: *x* = *n*+*i* ∧ *enat i* < *llength xs* ∧ *P* (*lnth xs i*)
by *blast*
thus *?lhs*
proof (*induct i arbitrary*: *xs n*)
case *0*
then show *?case*
proof −
have *16*: *lhd* (*kfilter P n xs*) = *n*+(*lleast P xs*)
using *kfilter-ldropWhile*[*of P n xs*] **using** *0.prems*
by (*metis in-lset-conv-lnth kfilter-lnull-conv lhd-LCons*)
show *?thesis*
by (*metis 0.prems add.right-neutral kfilter-LCons ldropn-0 ldropn-Suc-conv-ldropn*
llist.set-intros(*1*))
qed
next
case (*Suc i*)
then show *?case*

proof −
have *18*: *lnull xs* $\Longrightarrow$ *x* ∈ *lset* (*kfilter P n xs*)
using *Suc*(*2*) **by** *auto*
have *19*: ¬ *lnull xs* $\Longrightarrow$ *P* (*lnth xs* (*Suc i*)) = *P* (*lnth* (*ltl xs*) (*i*))
by (*simp add*: *lnth-ltl*)
have *20*: ¬ *lnull xs* $\Longrightarrow$ *x* = (*Suc n*) +*i* ∧ *enat i* < *llength* (*ltl xs*) ∧ *P* (*lnth* (*ltl xs*) (*i*))
by (*metis 19 Extended-Nat.eSuc-mono Suc.prems*(*1*) *add-Suc-shift eSuc-enat lhd-LCons-ltl llength-LCons*)
have *21*: ¬ *lnull xs* $\Longrightarrow$ *lnull* (*kfilter P n* (*ltl xs*)) $\Longrightarrow$ *x* ∈ *lset* (*kfilter P n xs*)
by (*metis 20 in-lset-conv-lnth kfilter-lnull-conv*)
have *22*: ¬ *lnull xs* $\Longrightarrow$ ¬*lnull* (*kfilter P n* (*ltl xs*)) $\Longrightarrow$ *x* ∈ *lset* (*kfilter P* (*Suc n*) (*ltl xs*))
by (*metis 20 Suc.hyps*)
have *23*: ¬*lnull xs* $\Longrightarrow$ *x* ∈ *lset* (*kfilter P n xs*)
using *kfilter-LCons*[*of P n lhd xs ltl xs*]
in-lset-ltlD lhd-LCons-ltl[*of* (*kfilter P n* (*ltl xs*))]
using *21 22* **by** *fastforce*
show *?thesis*
using *18 23* **by** *blast*
qed
qed
qed

lemma *kfilter-lset*:
shows *lset* (*kfilter P n xs*) = { *n*+*i* | *i*. *i*<*llength xs* ∧ *P* (*lnth xs i*) }
using *in-kfilter-lset* **by** *blast*

lemma *lfilter-lnth-exist*:
assumes
i < *llength* (*lfilter P xs*)
shows (∃ *k* < *llength xs*. *lnth* (*lfilter P xs*) *i* = *lnth xs k*)
using *assms lset-lfilter*[*of P xs*]
by (*metis* (*no-types*, *opaque-lifting*) *add.left-neutral diff-zero kfilter-llength kfilter-upperbound lfilter-kfilter zero-enat-def*)

lemma *ldistinct-kfilter*:
ldistinct(*kfilter P n xs*)
proof (*coinduction arbitrary*: *n xs*)
case (*ldistinct n1 xs1*)
then show *?case*
proof −
have *1*: *lhd* (*kfilter P n1 xs1*) ∉ *lset* (*ltl* (*kfilter P n1 xs1*))
proof −
have *f1*: *kfilter P n1 xs1* =
LCons (*n1* + *lleast P xs1*)
(*lmap snd*
(*lfilter* (*P* ∘ *fst*)
(*lzip*
(*ldrop* (*enat* (*Suc* (*lleast P xs1*))) *xs1*)
(*iterates Suc* (*Suc* (*n1* + *lleast P xs1*))))))
by (*meson kfilter-ldropWhile ldistinct*)

then have *lmap snd*
(lfilter (P ∘ fst)
(lzip (ldrop (enat (Suc (lleast P xs1))) xs1)
(iterates Suc (Suc (n1 + lleast P xs1))))) =
kfilter P (Suc (n1 + lleast P xs1)) (ldrop (enat (Suc (lleast P xs1))) xs1)
using *kfilter-eq-LCons-1* **by** *blast*
then show *?thesis*
using *f1* **by** *(metis (no-types) in-lset-conv-lnth kfilter-lowerbound leD lessI lhd-LCons ltl-simps(2))*
qed
have *2*: $((\exists\, n\ xs.\ ltl\ (kfilter\ P\ n1\ xs1) = kfilter\ P\ n\ xs) \vee ldistinct\ (ltl\ (kfilter\ P\ n1\ xs1)))$
by *(metis kfilter-eq-LCons-1 ldistinct llist.discI(1) llist.exhaust-sel)*
show *?thesis*
using *1 2* **by** *auto*
qed
qed

lemma *kfilter-llength-ltake*:
$llength(kfilter\ P\ n\ (ltake\ k\ xs)) \leq llength(kfilter\ P\ n\ xs)$
by *(simp add: kfilter-llength lprefix-lfilterI lprefix-llength-le)*

lemma *kfilter-ldropn-lset*:
assumes $k < llength\ xs$
shows $lset(kfilter\ P\ n\ (ldropn\ k\ xs)) =$
$\{\ n+i \mid i.\ i < llength\ xs - k \wedge P\ (lnth\ xs\ (k+i))\ \}$
using *assms*
kfilter-lset[of P n (ldropn k xs)]
by *auto*
(metis (no-types, lifting) add.commute enat-min lnth-ldropn order.strict-implies-order
plus-enat-simps(1),
metis (no-types, lifting) add.commute dual-order.strict-implies-order enat-min lnth-ldropn
plus-enat-simps(1))

lemma *kfilter-ldropn-lset-a*:
assumes $k < llength\ xs$
shows $lset(kfilter\ P\ n\ (ldropn\ k\ xs)) =$
$\{\ n+(i-k) \mid i.\ k \leq i \wedge i < llength\ xs \wedge P\ (lnth\ xs\ i)\ \}$
proof –
have *1*: $\bigwedge x.\ x \in \{\ n+i \mid i.\ i < llength\ xs - k \wedge P\ (lnth\ xs\ (k+i))\ \} \longleftrightarrow$
$x \in \{\ n+(i-k) \mid i.\ k \leq i \wedge i < llength\ xs \wedge P\ (lnth\ xs\ i)\ \}$
proof *auto*
show $\bigwedge i.\ enat\ i < llength\ xs - enat\ k \Longrightarrow$
$P\ (lnth\ xs\ (k + i)) \Longrightarrow$
$\exists\, ia.\ i = ia - k \wedge k \leq ia \wedge enat\ ia < llength\ xs \wedge P\ (lnth\ xs\ ia)$
using *assms*
by *(metis add.commute add-diff-cancel-left′ enat-min le-add1 less-imp-le plus-enat-simps(1))*
show $\bigwedge i.\ k \leq i \Longrightarrow$
$enat\ i < llength\ xs \Longrightarrow$
$P\ (lnth\ xs\ i) \Longrightarrow$
$\exists\, ia.\ n + i - k = n + ia \wedge enat\ ia < llength\ xs - enat\ k \wedge P\ (lnth\ xs\ (k + ia))$

using *assms ldropn-Suc-conv-ldropn*[*of - xs*] *ldropn-eq-LConsD*[*of - ldropn k xs*] *ldropn-ldropn*[*of - - xs*]
by (*metis Nat.add-diff-assoc le-add-diff-inverse le-add-diff-inverse2 llength-ldropn*)
qed
show *?thesis* **using** *assms 1 kfilter-ldropn-lset*[*of k xs P n*] **by** *auto*
qed

lemma *kfilter-ldropn-lset-b*:
assumes $k < llength\ xs$
shows $lset(kfilter\ P\ n\ (ldropn\ k\ xs)) =$
$\{\ n+i \mid i.\ i < llength\ xs\ -k \wedge P\ (lnth\ xs\ (i+k))\ \}$
proof −
have *1*: $\bigwedge x.\ x \in \{\ n+i \mid i.\ i < llength\ xs\ -\ k \wedge P\ (lnth\ xs\ (k+i))\ \} \longleftrightarrow$
$x \in \{\ n+i \mid i.\ i < llength\ xs\ -k \wedge P\ (lnth\ xs\ (i+k))\ \}$
by (*auto simp add*: *add.commute*)
show *?thesis* **using** *assms 1 kfilter-ldropn-lset*[*of k xs P n*] **by** *auto*
qed

lemma *kfilter-llength-n-zero*:
shows $llength(kfilter\ P\ n\ xs) = llength(kfilter\ P\ 0\ xs)$
by (*simp add*: *kfilter-llength*)

lemma *kfilter-lnth-n-zero-a*:
assumes $k < llength\ (kfilter\ P\ n\ xs)$
shows $n \leq (lnth\ (kfilter\ P\ n\ xs)\ k)$
using *assms* **by** (*simp add*: *kfilter-lnull-conv kfilter-lowerbound*)

lemma *kfilter-lnth-n-zero*:
assumes $k < llength\ (kfilter\ P\ n\ xs)$
shows $(lnth\ (kfilter\ P\ n\ xs)\ k)\ -n = (lnth\ (kfilter\ P\ 0\ xs)\ k)$
using *assms*
proof (*induct k arbitrary*: *xs n*)
case *0*
then show *?case* **by** (*cases* (*kfilter P n xs*))
(*simp*,
metis add.left-neutral add-left-cancel kfilter-ldropWhile kfilter-lnull-conv kfilter-lowerbound le-add-diff-inverse llength-eq-0 lnth-0 not-gr-zero zero-enat-def)
next
case (*Suc k*)
then show *?case*
proof −
have *1*: $lnull(kfilter\ P\ n\ xs) \Longrightarrow lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k)\ -\ n = lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)$
using *Suc.prems gr-implies-not-zero llength-lnull* **by** *blast*
have *2*: $\neg lnull(kfilter\ P\ n\ xs) \Longrightarrow lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k)\ -n = lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)$
proof −
assume *a0*: $\neg lnull(kfilter\ P\ n\ xs)$
show $lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k)\ -n = lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)$
proof −
let *?Least* = *lleast P xs*
have *3*: $\exists\ x\ xs'.(kfilter\ P\ n\ xs) = LCons\ x\ xs'$

using *a0 kfilter-ldropWhile* **by** *blast*
obtain $x\ xs'$ **where** 4: $(kfilter\ P\ n\ xs) = LCons\ x\ xs'$
using *3* **by** *auto*
have 5: $x = n+?Least$
using *kfilter-ldropWhile*[*of P n xs*]
by (*simp add*: *4*)
have 6: $\neg lnull(kfilter\ P\ n\ xs) \longleftrightarrow \neg lnull(kfilter\ P\ 0\ xs)$
by (*simp add*: *kfilter-not-lnull-conv*)
have 7: $\exists\ y\ ys'.(kfilter\ P\ 0\ xs) = LCons\ y\ ys'$
using *6 a0 kfilter-ldropWhile* **by** *blast*
obtain $y\ ys'$ **where** 8: $(kfilter\ P\ 0\ xs) = LCons\ y\ ys'$
using *7* **by** *auto*
have 9: $y = ?Least$
using *kfilter-ldropWhile*[*of P 0 xs*] **by** (*simp add*: *8*)
have 10: $x-n = y$
using *5 9 diff-add-inverse* **by** *blast*
have 11: $xs' = kfilter\ P\ (Suc\ (n+?Least))\ (ldrop\ (Suc\ ?Least)\ xs)$
using *kfilter-ldropWhile*[*of P n xs*]
by (*metis* (*no-types*, *lifting*) *4 a0 kfilter-def ltl-simps*(*2*))
have 12: $ys' = kfilter\ P\ (Suc\ (0+?Least))\ (ldrop\ (Suc\ ?Least)\ xs)$
using *kfilter-ldropWhile*[*of P 0 xs*]
by (*metis* (*no-types*, *lifting*) *6 8 a0 kfilter-def ltl-simps*(*2*))
have 13: $lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k) - n\ = lnth\ xs'\ k - n$
by (*simp add*: *4*)
have 14: $lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)\ \ = lnth\ ys'\ k$
by (*simp add*: *8*)
have 15: $\neg(\exists\ x \in lset\ (ldrop\ (Suc\ ?Least)\ xs).\ P\ x) \Longrightarrow$
$lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k) -n = lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)$
using *8 Suc* **by** *auto*
(*metis* (*full-types*) *gr-implies-not-zero kfilter-eq-conv kfilter-not-lnull-conv*
ldropn-Suc-LCons ldropn-Suc-conv-ldropn ldropn-eq-LConsD llength-LNil llist.disc(*2*))
have 16: $enat\ k < llength\ (kfilter\ P\ (Suc\ (n+?Least))\ (ldrop\ (Suc\ ?Least)\ xs))$
by (*metis* (*no-types*, *lifting*) *12 8 Extended-Nat.eSuc-mono Suc.prems eSuc-enat*
kfilter-llength llength-LCons)
have 17: $(\exists\ x \in lset\ (ldrop\ (Suc\ ?Least)\ xs).\ P\ x) \Longrightarrow$
$lnth\ xs'\ k\ - (Suc\ (n+?Least)) = lnth\ (kfilter\ P\ 0\ (ldrop\ (Suc\ ?Least)\ xs))\ k$
using *Suc.hyps* **using** *11 16* **by** *blast*
have 18: $enat\ k < llength\ (kfilter\ P\ (Suc\ (?Least))\ (ldrop\ (Suc\ ?Least)\ xs))$
by (*metis 16 kfilter-llength*)
have 19: $(\exists\ x \in lset\ (ldrop\ (Suc\ ?Least)\ xs).\ P\ x) \Longrightarrow$
$lnth\ ys'\ k - (Suc\ (0+?Least)) =$
$lnth\ (kfilter\ P\ 0\ (ldrop\ (Suc\ ?Least)\ xs))\ k$
using *Suc.hyps* **by** (*simp add*: *12 18*)
have 20: $(\exists\ x \in lset\ (ldrop\ (Suc\ ?Least)\ xs).\ P\ x) \Longrightarrow$
$lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k) -n = lnth\ xs'\ k\ -n$
using *13* **by** *blast*
have 21: $(\exists\ x \in lset\ (ldrop\ (Suc\ ?Least)\ xs).\ P\ x) \Longrightarrow$
$lnth\ xs'\ k\ -n = lnth\ (kfilter\ P\ 0\ (ldrop\ (Suc\ ?Least)\ xs))\ k + (Suc\ (?Least))$
using *11 16 17 kfilter-lowerbound* **by** *fastforce*
have 22: $(\exists\ x \in lset\ (ldrop\ (Suc\ ?Least)\ xs).\ P\ x) \Longrightarrow$

lnth (*kfilter P 0* (*ldrop* (*Suc ?Least*) *xs*)) *k* + (*Suc* (*?Least*)) = *lnth ys′ k*
using *12 18 19 kfilter-lowerbound* **by** *fastforce*
have *23*: (∃ *x* ∈ *lset* (*ldrop* (*Suc ?Least*) *xs*). *P x*) ⟹
lnth (*kfilter P n xs*) (*Suc k*) −*n* = *lnth* (*kfilter P 0 xs*) (*Suc k*)
using *14 20 21 22* **by** *linarith*
show *?thesis*
using *15 23* **by** *blast*
qed
qed
show *?thesis*
using *1 2* **by** *blast*
qed
qed

lemma *kfilter-n-zero*:
shows (*kfilter P n xs*) = *lmap* (λ*i*. *i*+*n*) (*kfilter P 0 xs*)
proof −
have *1*: *llength*(*kfilter P n xs*) = *llength* (*lmap* (λ*i*. *i*+*n*) (*kfilter P 0 xs*))
by (*simp add*: *kfilter-llength*)
have *2*: ⋀*k*. *k* < *llength*(*kfilter P n xs*) ⟶
lnth (*kfilter P n xs*) *k* = *lnth* (*lmap* (λ*i*. *i*+*n*) (*kfilter P 0 xs*)) *k*
using *kfilter-lnth-n-zero kfilter-lnth-n-zero-a*
by (*metis 1 le-add-diff-inverse2 llength-lmap lnth-lmap*)
show *?thesis*
by (*simp add*: *1 2 llist-eq-lnth-eq*)
qed

lemma *kfilter-n-zero-a*:
shows (*kfilter P 0 xs*) = *lmap* (λ*i*. *i*−*n*) (*kfilter P n xs*)
proof −
have *1*: *llength*(*kfilter P 0 xs*) = *llength* (*lmap* (λ*i*. *i*−*n*) (*kfilter P n xs*))
by (*simp add*: *kfilter-llength*)
have *2*: ⋀*k*. *k* < *llength*(*kfilter P 0 xs*) ⟶
lnth (*kfilter P 0 xs*) *k* = *lnth* (*lmap* (λ*i*. *i*−*n*) (*kfilter P n xs*)) *k*
by (*simp add*: *kfilter-llength kfilter-lnth-n-zero*)
show *?thesis*
using *1 2 llist-eq-lnth-eq* **by** *blast*
qed

lemma *kfilter-holds*:
assumes *y* ∈ *lset*(*kfilter P n xs*)
shows *P* (*lnth xs* (*y*−*n*))
using *assms in-kfilter-lset*[*of y P n xs*] *kfilter-lnull-conv*[*of P n xs*]
using *lset-lnull* **by** *fastforce*

lemma *kfilter-holds-not*:
assumes *y* ∈ ({*i*+*n*| *i*. *i* < *llength xs*} − (*lset* (*kfilter P n xs*)))
shows ¬ *P* (*lnth xs* (*y*−*n*))
using *assms kfilter-lset*[*of P n xs*] *kfilter-lnull-conv*[*of P n xs*]
by *auto*

lemma *kfilter-holds-a*:
assumes $i < llength\ xs$
$(i+n) \in lset(kfilter\ P\ n\ xs)$
shows $P\ (lnth\ xs\ i)$
using *assms kfilter-holds*[*of* $i+n$ P n xs] **by** *simp*

lemma *kfilter-holds-not-a*:
assumes $i < llength\ xs$
$P\ (lnth\ xs\ i)$
shows $(i+n) \in lset(kfilter\ P\ n\ xs)$
using *assms*
by (*simp add*: *in-kfilter-lset kfilter-lnull-conv*)

lemma *kfilter-holds-b*:
assumes $i < llength\ xs$
shows $(i+n) \in lset(kfilter\ P\ n\ xs) = \ P\ (lnth\ xs\ i)$
using *assms*
by (*meson kfilter-holds-a kfilter-holds-not-a*)

lemma *kfilter-holds-c*:
assumes $n \le i$
$i - n < llength\ xs$
shows $i \in lset(kfilter\ P\ n\ xs) = P\ (lnth\ xs\ (i-n))$
using *assms*
by (*metis diff-add idiff-enat-enat kfilter-holds kfilter-holds-not-a*)

lemma *kfilter-holds-not-b*:
assumes $n \le i$
$i - n < llength\ xs$
shows $i \notin lset(kfilter\ P\ n\ xs) = (\neg\ P\ (lnth\ xs\ (i-n)))$
using *assms* **by** (*simp add*: *kfilter-holds-c*)

lemma *kfilter-disjoint-lset-coset*:
shows $(\{i+n|\ i.\ i < llength\ xs\} - (lset\ (kfilter\ P\ n\ xs))) \cap lset\ (kfilter\ P\ n\ xs) = \{\}$
by *blast*

lemma *lidx-kfilter-expand*:
assumes $(Suc\ na) < llength(kfilter\ P\ n\ xs)$
shows $lnth\ (kfilter\ P\ n\ xs)\ na < lnth\ (kfilter\ P\ n\ xs)\ (Suc\ na)$
using *assms kfilter-mono* **by** *force*

lemma *lidx-kfilter*:
shows $lidx\ \ (kfilter\ P\ n\ xs)$
unfolding *lidx-def*
using *lidx-kfilter-expand* **by** *blast*

lemma *lidx-kfilter-gr-eq*:
assumes
$k \le j$

j < *llength*(*kfilter P n xs*)
shows *lnth* (*kfilter P n xs*) *k* ≤ *lnth* (*kfilter P n xs*) *j*
using *assms*
using *lidx-kfilter lidx-less-eq* **by** *blast*

lemma *lidx-kfilter-gr*:
shows ∀ *j* . *k*<*j* ∧ *j* < *llength*(*kfilter P n xs*) ⟶
lnth (*kfilter P n xs*) *k* < *lnth* (*kfilter P n xs*) *j*
using *less-imp-Suc-add lidx-kfilter lidx-less* **by** *blast*

lemma *kfilter-not-before*:
assumes *0*<*llength*(*kfilter P 0 xs*)
i< *lnth* (*kfilter P 0 xs*) *0*
shows ¬ *P* (*lnth xs i*)
proof −
have *0*: (*lnth* (*kfilter P 0 xs*) *0*) < *llength xs*
by (*metis assms*(*1*) *gen-llength-def kfilter-upperbound llength-code zero-enat-def*)
have *1*: ¬ *lnull* (*kfilter P 0 xs*)
using *assms*(*1*) **by** *auto*
have *2*: *i* ∈ *lset* (*kfilter P 0 xs*) ⟹
¬ *lnull* (*kfilter P 0 xs*) ⟹
i< *lnth* (*kfilter P 0 xs*) *0* ⟹
False
proof (*induct zs*≡(*kfilter P 0 xs*) *arbitrary*: *xs rule*: *lset-induct*)
case (*find xs*)
then show *?case* **by** (*metis less-not-refl3 lhd-LCons lnth-0-conv-lhd*)
next
case (*step x′ xs*)
then show *?case*
proof −
have ∀ *ns n*. ∃ *na*. ((*n*::*nat*) ∉ *lset ns* ∨ *lnth ns na* = *n*) ∧ (*n* ∉ *lset ns* ∨ *enat na* < *llength ns*)
by (*meson in-lset-conv-lnth*)
then show *?thesis* **using** *step*
by (*metis* (*no-types*) *leD less-nat-zero-code lidx-kfilter-gr-eq llist.set-intros*(*2*) *not-le-imp-less*)
qed
qed
have *3*: *i* ∉ *lset* (*kfilter P 0 xs*) ∧ *i* < *llength xs* ⟶ ¬ *P* (*lnth xs i*)
by (*simp add*: *1 in-kfilter-lset*)
have *4*: *i* < *llength xs*
using *0 assms*(*2*) *dual-order.strict-trans enat-ord-simps*(*2*) **by** *blast*
show *?thesis*
using *1 2 3 4 assms*(*2*) **by** *blast*
qed

lemma *kfilter-n-not-before*:
assumes *0* < *llength* (*kfilter P n* (*ldropn n xs*))
n < *llength xs*
n ≤ *i*
i< *lnth* (*kfilter P n* (*ldropn n xs*)) *0*
shows ¬ *P* (*lnth xs i*)

proof −
 have *00*: ¬ *lnull* (*kfilter P n* (*ldrop n xs*))
 by (*metis assms*(*1*) *ldrop-enat less-numeral-extra*(*3*) *llength-LNil llist.collapse*(*1*))
 have *0*: *lnth* (*kfilter P n* (*ldrop n xs*)) *0* < *llength xs*
 using *assms kfilter-upperbound*[*of 0 P n* (*ldropn n xs*)]
 by (*metis lappend-ltake-enat-ldropn ldrop-enat llength-lappend llength-ltake min.strict-order-iff zero-enat-def*)
 have *1*: *i* ∈ *lset* (*kfilter P n* (*ldropn n xs*)) ⟹
 ¬ *lnull* (*kfilter P n* (*ldropn n xs*)) ⟹
 i< *lnth* (*kfilter P n* (*ldropn n xs*)) *0* ⟹
 n < *llength xs* ⟹
 n ≤ *i*⟹
 False
 proof (*induct zs*≡(*kfilter P n* (*ldropn n xs*)) *arbitrary*: *n xs rule*: *lset-induct*)
 case (*find xs*)
 then show *?case*
 by (*metis lnth-0 nat-less-le*)
 next
 case (*step x′ xs*)
 then show *?case*
 by (*metis* (*no-types*, *lifting*) *in-lset-conv-lnth kfilter-eq-LCons-1 kfilter-lowerbound leD less-SucI lnth-0*)
 qed
 have *2*: *i* ∉ *lset* (*kfilter P n* (*ldropn n xs*)) ∧ *n* ≤ *i* ∧ *i* < *llength xs* ⟶ ¬ *P* (*lnth xs i*)
 using *assms kfilter-holds-not-a*[*of i*] *00*
 by (*simp add*: *kfilter-ldropn-lset-a ldrop-enat*)
 have *3*: *n* ≤ *i* ∧ *i* < *llength xs*
 using *assms 00 kfilter-ldropWhile*[*of P n ldropn n xs*]
 by (*metis 0 enat-ord-simps*(*2*) *ldrop-enat less-trans*)
 show *?thesis*
 using *1 2 3 assms*(*1*) *assms*(*2*) *assms*(*4*) *kfilter-not-lnull-conv* **by** *auto*
qed

lemma *kfilter-not-after*:
 assumes *0*<*llength*(*kfilter P 0 xs*)
 lnth (*kfilter P 0 xs*) (*k*−*1*) < *i*
 llength (*kfilter P 0 xs*) = (*enat k*)
 i < *llength xs*
 shows ¬ *P* (*lnth xs i*)
proof −
 have *0*: ¬ *lnull* (*kfilter P 0 xs*)
 using *assms*(*1*) **by** *auto*
 have *01*: *0*<*k*
 using *0 assms*(*3*) *gr0I zero-enat-def* **by** *fastforce*
 have *02*: *i* ∉ *lset* (*kfilter P 0 xs*)
 by (*metis 01 One-nat-def Suc-pred assms*(*2*) *assms*(*3*) *diff-less enat-ord-simps*(*2*) *in-lset-conv-lnth leD less-Suc-eq-le lidx-kfilter-gr-eq zero-less-one*)
 from *0 01 02* **show** *?thesis*
 by (*metis add.right-neutral assms*(*4*) *kfilter-holds-not-a*)

qed

lemma *kfilter-n-not-after*:
 assumes *0* < *llength* (*kfilter P n* (*ldropn n xs*))
 n < *llength xs*
 lnth (*kfilter P n* (*ldropn n xs*)) (*k*−*1*) < *i*
 llength (*kfilter P n* (*ldropn n xs*)) = (*enat k*)
 i < *llength xs*
 shows ¬ *P* (*lnth xs i*)
proof −
 have *0*: ¬ *lnull* (*kfilter P n* (*ldropn n xs*))
 using *assms*(*1*) **by** *auto*
 have *1*: *0*<*k*
 using *assms*(*1*) *assms*(*4*) **by** (*simp add*: *zero-enat-def*)
 have *2*: *i* ∉ *lset*(*kfilter P n* (*ldropn n xs*))
 using *assms*
 by (*metis 1 Suc-diff-1 enat-ord-simps*(*2*) *in-lset-conv-lnth leD less-Suc-eq-le lidx-kfilter-gr-eq order-refl*)
 from *0 1 2* **show** *?thesis*
using *assms kfilter-ldropn-lset-a*[*of n xs P n*] *kfilter-lowerbound*[*of - P n* (*ldropn n xs*)]
 by *auto* (*metis One-nat-def diff-less le-trans less-imp-le zero-less-one*)
qed

lemma *kfilter-not-between*:
 assumes *lnth* (*kfilter P 0 xs*) (*k*) < *i*
 i<*lnth* (*kfilter P 0 xs*) (*Suc k*)
 (*Suc k*) < *llength* (*kfilter P 0 xs*)
 shows ¬ *P* (*lnth xs i*)
proof −
 have *0*: ∃ *x* ∈ *lset xs*. *P x*
 using *assms*(*3*) *gr-implies-not-zero kfilter-not-lnull-conv* **by** *fastforce*
 have *1*: ¬ *lnull* (*kfilter P 0 xs*)
 by (*simp add*: *0 kfilter-not-lnull-conv*)
 have *2*: *lnth* (*kfilter P 0 xs*) (*Suc k*) ≤ *llength xs*
 using *kfilter-upperbound*[*of Suc k P 0 xs*]
 using *1 assms*(*3*) *zero-enat-def* **by** *auto*
 have *3*: *i* ∉ *lset*(*kfilter P 0 xs*)
 using *assms*
 by (*metis Suc-ile-eq dual-order.strict-implies-order in-lset-conv-lnth leD lidx-kfilter-gr-eq not-less-eq-eq*)
 from *1 2 3* **show** *?thesis*
 by (*metis add.right-neutral assms*(*2*) *enat-ord-simps*(*2*) *kfilter-holds-not-a less-le-trans*)
qed

lemma *lfilter-kfilter-ltake-lidx-a*:
 assumes *k* < *llength*(*lfilter P xs*)
 shows *lidx* (*ltake k* (*kfilter P n xs*))
unfolding *lidx-def*
using *assms*
by (*metis Suc-ile-eq kfilter-mono less-imp-le llength-ltake lnth-ltake min.strict-boundedE*)

```
lemma lfilter-kfilter-ldropn-lidx-a:
 assumes  k < llength(lfilter P xs)
 shows   lidx (ldropn k (kfilter P n xs))
using assms unfolding lidx-def
  proof auto
   fix na
   assume a0: enat k < llength (lfilter P xs)
   assume a1: enat (Suc na) < llength (kfilter P n xs) − enat k
   show lnth (ldropn k (kfilter P n xs)) na < lnth (ldropn k (kfilter P n xs)) (Suc na)
   proof −
    have 1: enat (k + (Suc na)) < llength (kfilter P n xs)
      proof −
       have Suc na + k = k + Suc na
         by presburger
       then show ?thesis
         by (metis (no-types) a1 ldropn-eq-LNil ldropn-ldropn leD leI llength-ldropn)
      qed
    have 2 : enat (k + na) < llength (kfilter P n xs)
      by (metis 1 Suc-ile-eq add-Suc-right less-imp-le)
    have 3: lnth (kfilter P n xs) (k + (na)) <  lnth (kfilter P n xs) (k + (Suc na))
      using 1 kfilter-mono by auto
    show ?thesis by (metis 1 2 3 add.commute lnth-ldropn)
   qed
qed

lemma lfilter-kfilter-ldropn-lidx-b:
 assumes  k < llength(lfilter P xs)
 shows   lidx (kfilter P (lnth (kfilter P n xs) k) (ldropn (lnth (kfilter P n xs) k) xs))
using assms using lidx-kfilter by blast

lemma ltake-lset:
 assumes k < llength xs
 shows    lset (ltake k xs) = {(lnth xs i) | i. i < k}
using assms
by (auto simp add: in-lset-conv-lnth lnth-ltake)
   (blast, meson less-trans lnth-ltake)

lemma ldropn-lset:
 assumes  k < llength xs
 shows  lset (ldropn k xs) = {(lnth xs i) | i. k ≤ i ∧ i < llength xs}
using assms
proof (auto simp add: in-lset-conv-lnth )
  fix n :: nat
  assume a1: enat n < llength xs − enat k
  assume enat k < llength xs
  then have enat k ≤ llength xs
    by (meson dual-order.strict-implies-order)
  then have enat n + enat k < llength xs
    using a1 by (meson enat-min)
```

then show ∃ *na. lnth (ldropn k xs) n = lnth xs na* ∧ *k* ≤ *na* ∧ *enat na* < *llength xs*
  using *a1* **by** (*metis ldropn-ldropn le-add2 lhd-ldropn llength-ldropn plus-enat-simps*(*1*))
next
 fix *i* :: *nat*
 assume *a1*: *enat i* < *llength xs*
 assume *a2*: *k* ≤ *i*
 have *1*: *enat* (*i*−*k*) < *llength xs* −*enat k*
  by (*metis a1 a2 diff-add ldropn-Suc-conv-ldropn ldropn-ldropn llength-ldropn llist.disc*(*2*)
   lnull-ldropn not-le-imp-less)
 have *2*: *lnth* (*ldropn k xs*) (*i*−*k*) = *lnth xs i*
  by (*simp add*: *a1 a2*)
 show ∃ *n. enat n* < *llength xs* − *enat k* ∧ *lnth* (*ldropn k xs*) *n* = *lnth xs i*
  using *1 2* **by** *blast*
qed

lemma *lfilter-kfilter-ltake-lset-eq*:
 assumes
   k < *llength*(*lfilter P xs*)
 shows *lset* (*ltake k* (*kfilter P 0 xs*)) =
   lset (*kfilter P 0* (*ltake* ((*lnth* (*kfilter P 0 xs*) *k*)) *xs*))
proof −
 have *1*: (*lnth* (*kfilter P 0 xs*) *k*) < *llength xs*
  using *kfilter-llength kfilter-upperbound*
  by (*metis assms gen-llength-def kfilter-llength kfilter-upperbound llength-code*)
 have *2*: ∃ *x* ∈ *lset xs* . *P x*
  using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
 have *3*: {*i. i*< *llength*(*ltake* ((*lnth* (*kfilter P 0 xs*) *k*)) *xs*) ∧
   P (*lnth* (*ltake* ((*lnth* (*kfilter P 0 xs*) *k*)) *xs*) *i*)} =
   {*i. i*< (*lnth* (*kfilter P 0 xs*) *k*) ∧ *P* (*lnth xs i*)}
  by (*auto simp add*: *lnth-ltake 1 order-less-subst2*)
 have *5*: {*i. i*< (*lnth* (*kfilter P 0 xs*) *k*) ∧ *P* (*lnth xs i*)} =
   {*i. i*< (*lnth* (*kfilter P 0 xs*) *k*) ∧ *i* ∈ *lset*(*kfilter P 0 xs*)}
  by (*auto simp add*: *1 kfilter-holds-c order-less-subst2*)
 have *6*: {*i. i*< (*lnth* (*kfilter P 0 xs*) *k*) ∧ *i* ∈ *lset*(*kfilter P 0 xs*)} =
   {*i. i*< (*lnth* (*kfilter P 0 xs*) *k*) ∧ *i* ∈ {(*lnth* (*kfilter P 0 xs*) *j*) | *j. j* < *llength*(*kfilter P 0 xs*)}}
  by (*simp add*: *lset-conv-lnth*)
 have *7*: {*i. i*< (*lnth* (*kfilter P 0 xs*) *k*) ∧ *i* ∈ {(*lnth* (*kfilter P 0 xs*) *j*) | *j. j* < *llength*(*kfilter P 0 xs*)}} =
    {(*lnth* (*kfilter P 0 xs*) *j*) | *j. j*<*k*}
  by (*auto simp add*: *assms lidx-kfilter-gr kfilter-llength*)
   (*metis kfilter-llength leD lidx-kfilter-gr-eq not-le-imp-less*,
   metis assms enat-ord-simps(*2*) *less-trans*)
 have *8*: *k* < *llength*(*kfilter P 0 xs*)
  by (*simp add*: *assms kfilter-llength*)
 have *9*: {(*lnth* (*kfilter P 0 xs*) *j*) | *j. j*<*k*} = *lset*(*ltake k* (*kfilter P 0 xs*))
  using *ltake-lset*[*of k* (*kfilter P 0 xs*)] *8* **by** *auto*
 have *10*: *lset* (*kfilter P 0* (*ltake* ((*lnth* (*kfilter P 0 xs*) *k*)) *xs*)) =
   {*i. i*< (*lnth* (*kfilter P 0 xs*) *k*) ∧
    P (*lnth* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) *i*) }
  by (*auto simp add*: *in-kfilter-lset*)
   (*meson 1 enat-ord-simps*(*2*) *less-trans*)

have *11*: (*lnth* (*kfilter P 0 xs*) *k*) = *llength*(*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)
by (*simp add*: *1 dual-order.strict-implies-order min-def*)
have *12*: {*i*. *i*< (*lnth* (*kfilter P 0 xs*) *k*) ∧
P (*lnth* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) *i*) } =
{*i*. *i*< *llength*(*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) ∧
P (*lnth* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) *i*) }
using *11* **by** *auto*
show *?thesis*
using *10 12 3 5 6 7 9* **by** *auto*
qed

lemma *lfilter-kfilter-ldropn-lset-eq*:
assumes *k* < *llength*(*lfilter P xs*)
shows *lset*(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lset(*ldropn* *k* (*kfilter P 0 xs*))
proof −
have *1*: (*lnth* (*kfilter P 0 xs*) *k*) < *llength xs*
using *kfilter-llength kfilter-upperbound*
by (*metis assms gen-llength-def kfilter-llength kfilter-upperbound llength-code*)
have *2*: ∃ *x* ∈ *lset xs* . *P x*
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have *10*: *lset*(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
{(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*. *i*< *llength xs* − (*lnth* (*kfilter P 0 xs*) *k*) ∧
P (*lnth xs* (*i*+(*lnth* (*kfilter P 0 xs*) *k*))) }
using *1 kfilter-ldropn-lset-b*[*of* (*lnth* (*kfilter P 0 xs*) *k*) *xs P* (*lnth* (*kfilter P 0 xs*) *k*)]
by *linarith*
have *5*: {(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*. *i*< *llength xs* − (*lnth* (*kfilter P 0 xs*) *k*) ∧
P (*lnth xs* (*i*+(*lnth* (*kfilter P 0 xs*) *k*))) } =
{(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*. *i*< *llength xs* − (*lnth* (*kfilter P 0 xs*) *k*) ∧
(*i*+(*lnth* (*kfilter P 0 xs*) *k*)) ∈ *lset*(*kfilter P 0 xs*) }
using *kfilter-holds-b*[*of - xs 0 P*] **using** *1 2*
by *auto*
(*metis ldropn-eq-LNil ldropn-ldropn leD llength-ldropn not-le-imp-less*,
metis gen-llength-def in-lset-conv-lnth kfilter-upperbound llength-code)
have *51*: {(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*. *i*< *llength xs* − (*lnth* (*kfilter P 0 xs*) *k*) ∧
(*i*+(*lnth* (*kfilter P 0 xs*) *k*)) ∈ *lset*(*kfilter P 0 xs*) } =
{(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*.
(*lnth* (*kfilter P 0 xs*) *k*) ≤ *i* + (*lnth* (*kfilter P 0 xs*) *k*) ∧
i + (*lnth* (*kfilter P 0 xs*) *k*) < *llength xs* ∧
(*i*+(*lnth* (*kfilter P 0 xs*) *k*)) ∈ *lset*(*kfilter P 0 xs*) }
by *auto*
(*metis 1 enat-min less-imp-le plus-enat-simps*(*1*),
metis ldropn-eq-LNil ldropn-ldropn leD llength-ldropn not-le-imp-less)
have *52*: {(*lnth* (*kfilter P 0 xs*) *k*) + *i* | *i*.
(*lnth* (*kfilter P 0 xs*) *k*) ≤ *i* + (*lnth* (*kfilter P 0 xs*) *k*) ∧
i + (*lnth* (*kfilter P 0 xs*) *k*) < *llength xs* ∧
(*i*+(*lnth* (*kfilter P 0 xs*) *k*)) ∈ *lset*(*kfilter P 0 xs*) } =
{*j*. (*lnth* (*kfilter P 0 xs*) *k*) ≤*j* ∧ *j* < *llength xs* ∧ *j* ∈ *lset*(*kfilter P 0 xs*)}
by (*metis* (*no-types*, *lifting*) *add.commute le-iff-add*)
have *53*: {*j*. (*lnth* (*kfilter P 0 xs*) *k*) ≤*j* ∧ *j* < *llength xs* ∧ *j* ∈ *lset*(*kfilter P 0 xs*)} =

{*j*. (*lnth* (*kfilter P 0 xs*) *k*) ≤*j* ∧ *j* < *llength xs* ∧
j ∈ { (*lnth* (*kfilter P 0 xs*) *jj*)| *jj*. *jj*< *llength*(*kfilter P 0 xs*)}}
by (*simp add*: *lset-conv-lnth*)
have *54*: {*j*. (*lnth* (*kfilter P 0 xs*) *k*) ≤*j* ∧ *j* < *llength xs* ∧
j ∈ { (*lnth* (*kfilter P 0 xs*) *jj*)| *jj*. *jj*< *llength*(*kfilter P 0 xs*)}} =
{ (*lnth* (*kfilter P 0 xs*) *j*) | *j*. *k*≤*j* ∧ *j* < *llength* (*kfilter P 0 xs*)}
by *auto*
(*metis assms kfilter-llength lidx-kfilter-gr not-less*,
meson lidx-kfilter-gr-eq,
metis gen-llength-def kfilter-upperbound llength-code)
have *8*: *k* < *llength*(*kfilter P 0 xs*)
by (*simp add*: *assms kfilter-llength*)
have *9*: { (*lnth* (*kfilter P 0 xs*) *j*) | *j*. *k*≤*j* ∧ *j* < *llength* (*kfilter P 0 xs*)} =
lset(*ldropn k* (*kfilter P 0 xs*))
using *ldropn-lset*[*of k* (*kfilter P 0 xs*)] **using** *8* **by** *blast*
show *?thesis*
using *10 5 51 52 53 54 9* **by** *auto*
qed

lemma *kfilter-kfilter-ltake*:
assumes *k* < *llength*(*lfilter P xs*)
shows (*ltake k* (*kfilter P 0 xs*)) =
(*kfilter P 0* (*ltake* ((*lnth* (*kfilter P 0 xs*) *k*)) *xs*))
using *assms*
by (*simp add*: *lfilter-kfilter-ltake-lidx-a lfilter-kfilter-ltake-lset-eq lidx-kfilter lidx-lset-eq*)

lemma *kfilter-kfilter-ldropn*:
assumes *k* < *llength*(*lfilter P xs*)
shows (*ldropn k* (*kfilter P 0 xs*)) =
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
using *assms*
by (*simp add*: *lfilter-kfilter-ldropn-lidx-a lfilter-kfilter-ldropn-lidx-b*
lfilter-kfilter-ldropn-lset-eq lidx-lset-eq)

lemma *kfilter-lmap-lfilter*:
shows *lmap* (λ*n*. (*lnth xs n*)) (*kfilter P 0 xs*) = *lfilter P xs*
using *lfilter-kfilter*[*of - P 0 xs*]
by (*metis* (*no-types*, *lifting*) *diff-zero kfilter-llength llength-lmap*
llist-eq-lnth-eq lnth-lmap)

lemma *lfilter-kfilter-ltake*:
assumes *k* < *llength*(*lfilter P xs*)
shows *ltake k* (*lfilter P xs*) =
(*lfilter P* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
proof −
have *2*: *lmap* (λ*n*. (*lnth xs n*)) (*kfilter P 0 xs*) = *lfilter P xs*
using *kfilter-lmap-lfilter* **by** *blast*
have *3*: *ltake k* (*lfilter P xs*) =
ltake k (*lmap* (λ*n*. (*lnth xs n*)) (*kfilter P 0 xs*))
by (*simp add*: *2*)

have *4*: *ltake k* (*lmap* (λ*n*. (*lnth xs n*)) (*kfilter P 0 xs*)) =
lmap (λ*s*. *lnth xs s*) (*ltake k* (*kfilter P 0 xs*))
using *ltake-lmap* **by** *blast*
have *6*: (*lfilter P* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lmap (λ*s*. (*lnth* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) *s*))
(*kfilter P 0* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
by (*simp add*: *kfilter-lmap-lfilter*)
have *7*: *lmap* (λ*s*. *lnth xs s*) (*ltake k* (*kfilter P 0 xs*)) =
lmap (λ*s*. *lnth xs s*) (*kfilter P 0* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
by (*simp add*: *assms kfilter-kfilter-ltake*)
have *8*: *lmap* (λ*s*. *lnth xs s*) (*kfilter P 0* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lmap (λ*s*. (*lnth* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) *s*))
(*kfilter P 0* (*ltake* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
using *kfilter-kfilter-ltake*[*of k P xs*]
by (*metis gen-llength-def kfilter-upperbound lappend-ltake-enat-ldropn ldistinct-Ex1 ldistinct-kfilter llength-code llist.map-cong0 lnth-lappend*)
show *?thesis*
using *2 4 6 7 8* **by** *auto*
qed

lemma *kfilter-lmap-shift-ldropn*:
shows *lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lmap (λ*s*. *lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
proof −
have *1*: *llength* (*lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) =
llength (*lmap* (λ*s*. *lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)))
by (*simp add*: *kfilter-llength*)
have *2*: $\bigwedge$*i*. *i*< *llength* (*lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) $\Longrightarrow$
lnth (*lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) *i* =
lnth xs ((*lnth* (*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) *i*)+
(*lnth* (*kfilter P 0 xs*) *k*))
by *simp*
have *3*: $\bigwedge$*i*. *i*< *llength* (*lmap* (λ*s*. *lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) $\Longrightarrow$
lnth (*lmap* (λ*s*. *lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*)
(*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) *i* =
lnth xs (*lnth* (*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*)
(*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) *i*)
by *simp*
have *4*: $\bigwedge$*i*. *i*< *llength* (*lmap* (λ*s*. *lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))) $\Longrightarrow$
lnth xs ((*lnth* (*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) *i*)+
(*lnth* (*kfilter P 0 xs*) *k*)) =

lnth xs (*lnth* (*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*)
(*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) *i*)
using *kfilter-lnth-n-zero*[*of - P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)]
kfilter-lowerbound[*of - P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)]
using *1 diff-add* **by** *fastforce*
show *?thesis*
using *llist-eq-lnth-eq*[*of lmap* (λ*s. lnth xs* (*s*+(*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))]
using *1 2 3 4* **by** *presburger*
qed

lemma *lfilter-kfilter-ldropn*:
assumes *k* < *llength*(*lfilter P xs*)
shows (*ldropn k* (*lfilter P xs*)) =
(*lfilter P* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
proof −
have *1*: ∃ *x* ∈ *lset xs*. *P x*
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have *2*: (*lfilter P xs*) = *lmap* (λ*s. lnth xs s*) (*kfilter P 0 xs*)
by (*simp add*: *kfilter-lmap-lfilter*)
have *3*: *ldrop k* (*lfilter P xs*) =
ldrop k (*lmap* (λ*s. lnth xs s*) (*kfilter P 0 xs*))
by (*simp add*: *2*)
have *4*: (*ldrop k* (*lmap* (λ*s. lnth xs s*) (*kfilter P 0 xs*))) =
(*lmap* (λ*s. lnth xs s*) (*ldrop k* (*kfilter P 0 xs*)))
using *ldrop-lmap* **by** *blast*
have *6*: (*lfilter P* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lmap (λ*s. lnth* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) *s*)
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
by (*simp add*: *kfilter-lmap-lfilter*)
have *61*: ⋀*z. z* ∈ *lset* ((*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)))
⟹ (λ*s. lnth* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) *s*) *z* =
(λ*s. lnth xs* (*s* + (*lnth* (*kfilter P 0 xs*) *k*))) *z*
using *assms in-lset-conv-lnth*[*of -* ((*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)))]
by *simp*
(*metis add.commute add.left-neutral kfilter-upperbound ldropn-eq-LNil ldropn-ldropn leD*
lnth-ldropn not-le-imp-less zero-enat-def)
have *7*: *lmap* (λ*s. lnth* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*) *s*)
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lmap (λ*s. lnth xs* (*s* + (*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
using *61* **by** *auto*
have *8*: *lmap* (λ*s. lnth xs* (*s* + (*lnth* (*kfilter P 0 xs*) *k*)))
(*kfilter P 0* (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lmap (λ*s. lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*))
by (*simp add*: *kfilter-lmap-shift-ldropn*)
have *9*: *lmap* (λ*s. lnth xs s*)
(*kfilter P* (*lnth* (*kfilter P 0 xs*) *k*) (*ldropn* (*lnth* (*kfilter P 0 xs*) *k*) *xs*)) =
lmap (λ*s. lnth xs s*) (*ldropn k* (*kfilter P 0 xs*))

by (*simp add*: *assms kfilter-kfilter-ldropn*)
show *?thesis*
by (*metis 4 7 8 9 kfilter-lmap-lfilter ldrop-enat*)
qed

lemma *lfilter-lnth-aa*:
assumes $n < llength\ (lfilter\ P\ xs)$
shows $P\ (lnth\ (lfilter\ P\ xs)\ n)$
using *assms*
by (*meson in-lset-conv-lnth lfilter-id-conv lfilter-idem*)

lemma *exist-one-conv*:
$(\exists!\ i.\ i{<}llength\ xs \wedge P\ (lnth\ xs\ i)) \longleftrightarrow$
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j{\neq}k \longrightarrow \neg\ P\ (lnth\ xs\ j)))$
by *blast*

lemma *lfilter-llength-one-conv-a*:
assumes $llength(lfilter\ P\ xs) = 1$
shows $\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j{\neq}k \longrightarrow \neg\ P\ (lnth\ xs\ j))$
proof −
have *1*: $P\ (lnth\ xs\ (lnth\ (kfilter\ P\ 0\ xs)\ 0))$
by (*metis assms*(*1*) *kfilter-llength kfilter-lmap-lfilter less-numeral-extra*(*1*) *lfilter-lnth-aa lnth-lmap zero-enat-def*)
have *2*: $(lnth\ (kfilter\ P\ 0\ xs)\ 0) < llength\ xs$
by (*metis assms*(*1*) *gen-llength-def kfilter-llength kfilter-upperbound less-numeral-extra*(*1*) *llength-code zero-enat-def*)
have *3*: $(\forall\ j < llength\ xs.\ j{\neq}\ (lnth\ (kfilter\ P\ 0\ xs)\ 0) \longrightarrow \neg\ P\ (lnth\ xs\ j))$
using *assms kfilter-not-after*[*of P xs*] *kfilter-not-before*[*of P xs*]
by (*metis One-nat-def diff-Suc-1 kfilter-llength linorder-neqE-nat one-enat-def zero-less-one*)
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *lfilter-llength-one-conv-c*:
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j{\neq}k \longrightarrow \neg\ P\ (lnth\ xs\ j))) \longleftrightarrow$
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j{<}k \vee k{<}j \longrightarrow \neg\ P\ (lnth\ xs\ j)))$
using *antisym-conv3* **by** *auto*

lemma *lfilter-llength-one-conv-d*:
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j < llength\ xs.\ j{<}k \vee k{<}j \longrightarrow \neg\ P\ (lnth\ xs\ j))) \longleftrightarrow$
$(\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
$(\forall\ j\ .\ j{<}k \longrightarrow \neg\ P\ (lnth\ xs\ j)) \wedge$
$(\forall\ j < llength\ xs.\ k{<}j \longrightarrow \neg\ P\ (lnth\ xs\ j)))$
by (*meson enat-ord-simps*(*2*) *less-trans*)

lemma *exist-one-lfilter-llength-one*:
assumes $(\exists!\ i.\ i{<}llength\ xs \wedge P\ (lnth\ xs\ i))$
shows $llength(lfilter\ P\ xs) \leq 1$
using *assms*
proof *auto*
 fix i :: *nat*
 assume *a1*: $\forall y\ y'.\ enat\ y < llength\ xs \wedge P\ (lnth\ xs\ y) \wedge enat\ y' < llength\ xs \wedge P\ (lnth\ xs\ y') \longrightarrow$
 $y = y'$
 show $llength\ (lfilter\ P\ xs) \leq 1$
 proof –
 have *f1*: $\forall e\ ea.\ (e{::}enat) \leq ea \vee ea < e$
 by (*meson not-le-imp-less*)
 have *f3*: $llength\ (kfilter\ P\ 0\ xs) = gen\text{-}llength\ 0\ (lfilter\ P\ xs)$
 by (*metis kfilter-llength llength-code*)
 have *f4*: $enat\ 0 + llength\ xs = llength\ xs$
 by (*metis gen-llength-def llength-code*)
 have $llength\ xs = enat\ 0 + llength\ xs$
 by (*metis* (*full-types*) *gen-llength-def llength-code*)
 then have *f5*: $\neg\ 1 < llength\ (lfilter\ P\ xs)$
 proof –
 have *g1*: $Suc\ 0 = 0 + Suc\ 0$
 by *auto*
 have *g2*: $\bigwedge i.\ enat\ i < llength\ (kfilter\ P\ 0\ xs) \Longrightarrow$
 $lnth\ xs\ (lnth\ (kfilter\ P\ 0\ xs)\ i - 0) = lnth\ (lfilter\ P\ xs)\ i$
 using *lfilter-kfilter*[*of - P 0 xs*] **by** *auto*
 have *g3*: $\bigwedge k.\ enat\ k < llength\ (kfilter\ P\ 0\ xs) \Longrightarrow$
 $lnth\ (kfilter\ P\ 0\ xs)\ k - 0 = lnth\ (kfilter\ P\ 0\ xs)\ k$
 using *kfilter-mono*[*of - P 0 xs*] **by** *auto*
 have *g4*: $\bigwedge n.\ enat\ n < llength\ (lfilter\ P\ xs) \Longrightarrow P\ (lnth\ (lfilter\ P\ xs)\ n)$
 using *lfilter-lnth-aa*[*of - P xs*] **by** *auto*
 have *g5*: $enat\ (0 + Suc\ 0) = 1$
 using *one-enat-def* **by** *auto*
 have $\neg\ 1 < gen\text{-}llength\ 0\ (lfilter\ P\ xs) \vee enat\ 0 < gen\text{-}llength\ 0\ (lfilter\ P\ xs)$
 using *zero-enat-def* **by** *fastforce*
 then show *?thesis*
 by (*metis g1 g2 g3 g4 g5 a1 f3 f4 kfilter-upperbound ldistinct-conv-lnth ldistinct-kfilter*
 llength-code n-not-Suc-n)
 qed
 show *?thesis*
 using *f5 not-le* **by** *blast*
 qed
qed

lemma *lfilter-llength-one-conv-b*:
assumes $\exists\ k < llength\ xs.\ P\ (lnth\ xs\ k) \wedge$
 $(\forall\ j < llength\ xs.\ j{\neq}k \longrightarrow \neg\ P\ (lnth\ xs\ j))$
shows $llength(lfilter\ P\ xs) = 1$
proof –
 have *1*: $llength(lfilter\ P\ xs) \leq 1$
 by (*metis assms exist-one-lfilter-llength-one*)

have *2*: *0<llength*(*lfilter P xs*)
by (*metis assms gr-zeroI in-lset-conv-lnth llength-eq-0 lnull-lfilter*)
show *?thesis*
by (*metis 1 2 One-nat-def Suc-ile-eq dual-order.antisym one-enat-def zero-enat-def*)
qed

lemma *lfilter-llength-one-conv*:
shows ($\exists$ *k* < *llength xs*. *P* (*lnth xs k*) $\wedge$
($\forall$ *j* < *llength xs*. *j*$\neq$*k* $\longrightarrow$ $\neg$ *P* (*lnth xs j*))) $\longleftrightarrow$
llength(*lfilter P xs*) =*1*
using *lfilter-llength-one-conv-a*[*of P xs*] *lfilter-llength-one-conv-b*[*of xs P*] **by** *blast*

lemma *lfilter-llength-one-conv-1*:
shows ($\exists$ *k* < *llength xs*. *P* (*lnth xs k*) $\wedge$
($\forall$ *j* < *llength xs*. *j*<*k* $\vee$ *k*<*j* $\longrightarrow$ $\neg$ *P* (*lnth xs j*))) $\longleftrightarrow$
llength(*lfilter P xs*) =*1*
using *lfilter-llength-one-conv*[*of xs P*] *lfilter-llength-one-conv-c*[*of xs P*]
by *blast*

lemma *lfilter-llength-one-conv-2*:
shows ($\exists$ *k* < *llength xs*. *P* (*lnth xs k*) $\wedge$
($\forall$ *j* . *j*<*k* $\longrightarrow$ $\neg$ *P* (*lnth xs j*)) $\wedge$
($\forall$ *j* < *llength xs*. *k*<*j* $\longrightarrow$ $\neg$ *P* (*lnth xs j*))) $\longleftrightarrow$
llength(*lfilter P xs*) =*1*
using *lfilter-llength-one-conv-1*[*of xs P*] *lfilter-llength-one-conv-d*[*of xs P*]
by *blast*

lemma *lfilter-lappend-ltake*:
assumes *k* < *llength xs*
shows *lfilter P* (*ltake k xs*) = *ltake* (*llength*(*lfilter P* (*ltake k xs*))) (*lfilter P xs*)
proof −
have *1*: *lfilter P xs* =
lappend (*lfilter P* (*ltake* (*the-enat k*) *xs*)) (*lfilter P* (*ldropn* (*the-enat k*) *xs*))
by (*metis enat-ord-code*(*4*) *lappend-ltake-enat-ldropn lfilter-lappend-lfinite lfinite-ltake*)
have *2*: *ltake* (*llength*(*lfilter P* (*ltake k xs*)))
(*lappend* (*lfilter P* (*ltake* (*the-enat k*) *xs*)) (*lfilter P* (*ldropn* (*the-enat k*) *xs*))) =
lfilter P (*ltake k xs*)
by (*metis assms enat-the-enat lfilter-idem lfinite-ltake llength-eq-infty-conv-lfinite*
llength-lfilter-ile llength-ltake ltake-all ltake-lappend1 min.strict-order-iff)
show *?thesis* **using** *1 2* **by** *simp*
qed

lemma *kfilter-lappend-lfinite*:
lfinite xs $\Longrightarrow$
kfilter P n (*lappend xs ys*) =
lappend (*kfilter P n xs*) (*kfilter P* (*n*+ (*the-enat*(*llength xs*))) *ys*)
unfolding *kfilter-def*
proof (*induct arbitrary*: *n rule*: *lfinite.induct*)
case *lfinite-LNil*
then show *?case* **by** *simp*

next
case (*lfinite-LConsI xs x*)
then show *?case*
  proof −
   have *1*: (*lzip* (*lappend* (*LCons x xs*) *ys*) (*iterates Suc n*)) =
          (*LCons* (*x* , *n*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*))))
     by (*simp add*: *lzip.ctr*(*2*))
   have *3*: *lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend* (*LCons x xs*) *ys*) (*iterates Suc n*))) =
          (*if P x then* (*LCons n* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*))))))
                else lmap snd (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*)))))
     using *1* **by** *auto*
   have *4*: (*if P x then* (*LCons n* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*))))))
                else lmap snd (*lfilter* (*P* ∘ *fst*) (*lzip* (*lappend xs ys*) (*iterates Suc* (*Suc n*))))) =
          (*if P x then*
            (*LCons n* (*lappend* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc* (*Suc n*)))))
                              (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* ((*Suc n*) + *the-enat* (*llength*
xs))))))))
           else (*lappend* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc* (*Suc n*)))))
                       (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* ((*Suc n*) + *the-enat* (*llength xs*))))))))
     using *lfinite-LConsI.hyps*(*2*) **by** *auto*
   have *5*: (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*LCons x xs*) (*iterates Suc n*)))) =
          (*if P x then* (*LCons n* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*xs*) (*iterates Suc* (*Suc n*))))))
                else (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*xs*) (*iterates Suc* (*Suc n*))))))
            by (*simp add*: *lzip.ctr*(*2*))
   have *6*: (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* (*n* + *the-enat* (*llength* (*LCons x xs*))))))) =
          (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* ((*Suc n*) + *the-enat* (*llength* (*xs*)))))))
     using *eSuc-enat lfinite-LConsI.hyps*(*1*) *llength-eq-infty-conv-lfinite* **by** *force*
   have *7*: *lappend* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip* (*LCons x xs*) (*iterates Suc n*))))
          (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* (*n* + *the-enat* (*llength* (*LCons x xs*))))))) =
          lappend (*if P x then* (*LCons n* (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc* (*Suc n*))))))
                    else (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip xs* (*iterates Suc* (*Suc n*))))))
               (*lmap snd* (*lfilter* (*P* ∘ *fst*) (*lzip ys* (*iterates Suc* ((*Suc n*) + ((*the-enat* (*llength xs*))))))))

     using *5 6* **by** *auto*
   show *?thesis* **using** *3 4 7* **by** *auto*
 qed
qed

lemma *kfilter-lappend-ltake*:
 assumes *k* < *llength xs*
 shows   *kfilter P n* (*ltake k xs*) = *ltake* (*llength*(*kfilter P n* (*ltake k xs*))) (*kfilter P n xs*)
proof −
 have *1*: *kfilter P n xs* = *lappend* (*kfilter P n* (*ltake* (*the-enat k*) *xs*))
                              (*kfilter P* (*n*+ (*the-enat k*)) (*ldropn* (*the-enat k*) *xs*))
   using *kfilter-lappend-lfinite*[*of* (*ltake* (*the-enat k*) *xs*) *P n* (*ldropn* (*the-enat k*) *xs*)]
   by (*metis assms enat-iless enat-ord-code*(*4*) *lappend-ltake-ldrop ldrop-enat lfinite-ltake*
      llength-ltake min.strict-order-iff neq-iff the-enat.simps)
 have *2*: *ltake* (*llength*(*kfilter P n* (*ltake k xs*)))
          (*lappend* (*kfilter P n* (*ltake* (*the-enat k*) *xs*))
                (*kfilter P* (*n*+ (*the-enat k*)) (*ldropn* (*the-enat k*) *xs*))) =

kfilter P n (*ltake k xs*)
by (*metis assms enat-the-enat lfinite-ltake llength-eq-infty-conv-lfinite llength-ltake ltake-all ltake-lappend1 min.strict-order-iff order-refl*)
show *?thesis* **using** *1 2* **by** *simp*
qed

lemma *lfilter-ldropn-llength*:
assumes $k < llength\ xs$
shows *llength* (*lfilter P* ((*ldropn k xs*))) ≤ *llength* (*lfilter P* (*xs*))
using *assms*
proof (*induct k arbitrary*: *xs*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
proof (*cases xs*)
case *LNil*
then show *?thesis* **by** *simp*
next
case (*LCons x21 x22*)
then show *?thesis* **using** *Suc Suc-ile-eq* **by** *auto*
(*meson Suc-ile-eq dual-order.trans ile-eSuc*)
qed
qed

lemma *lfilter-nlength-imp*:
shows *llength* (*lfilter* (λ*x*. *P x* ∧ *Q x*) *xs*) ≤ *llength* (*lfilter P xs*)
proof –
have *0*: *lfilter* (λ*x*. *P x* ∧ *Q x*) *xs* = *lfilter* (λ*x*. *Q x* ∧ *P x*) *xs*
by *meson*
have *1*: *lfilter* (λ*x*. *Q x* ∧ *P x*) *xs* = *lfilter Q* (*lfilter P xs*)
using *lfilter-lfilter*[*of Q P xs*] **by** *auto*
have *2*: *llength* (*lfilter Q* (*lfilter P xs*)) ≤ *llength* (*lfilter P xs*)
using *llength-lfilter-ile* **by** *blast*
show *?thesis* **by** (*simp add*: *1 2 0*)
qed

lemma *lnths-kfilter-lfilter*:
lnths xs (*lset*(*kfilter P 0 xs*)) = *lfilter P xs*
using *lfilter-conv-lnths*[*of P xs*] *kfilter-lset*[*of P 0 xs*]
by *simp*

1.9 Transfer rules

context includes *lifting-syntax*
begin

lemma *lbutlast-transfer* [*transfer-rule*]:
(*llist-all2 A* ===> *llist-all2 A*) *lbutlast lbutlast*

by (*auto simp add*: *rel-fun-def lbutlast-conv-ltake llist-all2-llengthD llist-all2-ltakeI*)

lemma *lleast-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> *llist-all2 A* ===> (=)) *lleast lleast*
unfolding *lleast-def*[*abs-def*]
by (*auto simp add*: *rel-fun-def*)
(*metis* (*full-types*, *opaque-lifting*) *llist-all2-conv-all-lnth*)

lemma *lfuse-transfer* [*transfer-rule*]:
(*llist-all2 A* ===> *llist-all2 A* ===> *llist-all2 A*) *lfuse lfuse*
by(*auto simp add*: *rel-fun-def intro*: *llist-all2-lfuseI*)

lemma *ridx-transfer* [*transfer-rule*]:
((*R* ===> *R* ===> (=)) ===> *llist-all2 R* ===> (=)) *ridx ridx*
by (*simp add*: *llist-all2-rsp rel-fun-def ridx-def llist-all2-conv-all-lnth*)
(*meson Suc-ile-eq order-less-imp-le*)

lemma *lsub-transfer* [*transfer-rule*]:
((=) ===> (=) ===> *llist-all2 A* ===> *llist-all2 A*) *lsub lsub*
by (*auto simp add*: *lsub-def rel-fun-def intro*: *llist-all2-ltakeI llist-all2-ldropI*)

lemma *lsubc-transfer* [*transfer-rule*]:
((=) ===> (=) ===> *llist-all2 A* ===> *llist-all2 A*) *lsubc lsubc*
by (*auto simp add*: *lsubc-def rel-fun-def min-def llist-all2-llengthD*
intro: *llist-all2-ltakeI llist-all2-ldropI*)

lemma *lfusecat-transfer* [*transfer-rule*]:
(*llist-all2* (*llist-all2 A*) ===> *llist-all2 A*) *lfusecat lfusecat*
by(*auto intro*: *llist-all2-lfusecatI*)

end

end

2 Coinductive non-empty lists and their operations

Coinductive lists are formalised by Andreas Lochbihler in [3]. We define coinductive non-empty lists *'a nellist* as a subtype of coinductive lists using the quotient type construction. The usual operators, like take, drop, length, nth, filter etc. are defined for *'a nellist*. The formalisation is based on terminated coinductive list defined by Andreas Lochbihler.

theory *NELList* **imports**
LList-Extras
begin

Coinductive non-empty lists *'a nellist* are the codatatype defined by the constructors *NNil* of type $'a \Rightarrow 'a$ *nellist* and *NCons* of type $'a \Rightarrow 'a$ *nellist* $\Rightarrow 'a$ *nellist*.

2.1 Type definition

```
consts nlast0 :: 'a

codatatype (nset: 'a) nellist =
    NNil (nlast : 'a)
  | NCons (nhd : 'a) (ntl : 'a nellist)
for
  map: nmap
  rel: nellist-all2
where
  nhd (NNil -) = undefined
| ntl (NNil b) = NNil b
| nlast (NCons - nell) = nlast0 nell


overloading
  nlast0 == nlast0::'a nellist ⇒ 'a
begin

partial-function (tailrec) nlast0
where nlast0 nell = (case nell of (NNil x) ⇒ x | (NCons y nell') ⇒ nlast0 nell')

end

lemma nlast0-nlast [simp]: nlast0 = nlast
proof –
 have 1: ⋀x. nlast0 x = nlast x
 by (simp add: nlast0.simps nlast-def)
 show ?thesis using 1 by (rule ext)
qed

lemmas nlast-NNil [code, nitpick-simp] = nellist.sel(1)

lemma nlast-NCons [simp, code, nitpick-simp]: nlast (NCons x nell) = nlast nell
by simp

declare nellist.sel(2) [simp del]

definition nfirst :: 'a nellist ⇒ 'a
 where nfirst nell = (case nell of (NNil b) ⇒ b | (NCons b nell') ⇒ b)

primcorec unfold-nellist :: ('a ⇒ bool) ⇒ ('a ⇒ 'b) ⇒ ('a ⇒ 'b) ⇒ ('a ⇒ 'a) ⇒ 'a ⇒ 'b nellist
 where
  p a ⟹ unfold-nellist p g1 g21 g22 a = NNil (g1 a) |
  - ⟹ unfold-nellist p g1 g21 g22 a =
     NCons (g21 a) (unfold-nellist p g1 g21 g22 (g22 a))

declare
  unfold-nellist.ctr(1) [simp]
  nellist.corec(1) [simp]
```

2.2 Code generator setup

More lemmas about generated constants

lemma *ntl-unfold-nellist*:
ntl (*unfold-nellist IS-NNIL NNIL NHD NTL a*) =
(*if IS-NNIL a then NNil* (*NNIL a*) *else unfold-nellist IS-NNIL NNIL NHD NTL* (*NTL a*))
by(*simp*)

lemma *is-NNil-ntl* [*simp*]:
is-NNil nell $\Longrightarrow$ *is-NNil* (*ntl nell*)
by(*cases nell*) *simp-all*

lemma *nlast-ntl* [*simp*]: *nlast* (*ntl nell*) = *nlast nell*
by(*cases nell*) *simp-all*

lemma *unfold-nellist-eq-NNil* [*simp*]:
unfold-nellist IS-NNIL NNIL NHD NTL a = *NNil b* $\longleftrightarrow$ *IS-NNIL a* $\wedge$ *b* = *NNIL a*
by(*auto simp add*: *unfold-nellist.code*)

lemma *NNil-eq-unfold-nellist* [*simp*]:
NNil b = *unfold-nellist IS-NNIL NNIL NHD NTL a* $\longleftrightarrow$ *IS-NNIL a* $\wedge$ *b* = *NNIL a*
by(*auto simp add*: *unfold-nellist.code*)

lemma *nmap-is-NNil*:
is-NNil nell $\Longrightarrow$ *nmap f nell* = *NNil* (*f* (*nlast nell*))
by(*clarsimp simp add*: *is-NNil-def*)

declare *nellist.map-sel*(*2*)[*simp*]

lemma *ntl-nmap* [*simp*]:
ntl (*nmap f nell*) = *nmap f* (*ntl nell*)
by(*cases nell*) *simp-all*

lemma *nmap-eq-NNil-conv*:
nmap f nell = *NNil y* $\longleftrightarrow$ ($\exists$ *y'*. *nell* = *NNil y'* $\wedge$ *f y'* = *y*)
by(*cases nell*) *simp-all*

lemma *NNil-eq-nmap-conv*:
NNil y = *nmap f nell* $\longleftrightarrow$ ($\exists$ *y'*. *nell* = *NNil y'* $\wedge$ *f y'* = *y*)
by(*cases nell*) *auto*

declare *nellist.set-sel*(*1*)[*simp*]

lemma *nset-ntl*: *nset* (*ntl nell*) $\subseteq$ *nset nell*
by(*cases nell*) *auto*

lemma *in-nset-ntlD*: *x* $\in$ *nset* (*ntl nell*) $\Longrightarrow$ *x* $\in$ *nset nell*
using *nset-ntl*[*of nell*] **by** *auto*

theorem *nellist-set-induct*[*consumes 1*, *case-names findnil find step*]:

assumes $x \in$ *nset nell*
and $\bigwedge$*nell. is-NNil nell* $\Longrightarrow$ *P* (*nlast nell*) *nell*
and $\bigwedge$*nell.* $\neg$ *is-NNil nell* $\Longrightarrow$ *P* (*nhd nell*) *nell*
and $\bigwedge$*nell y.* $[\![\neg$ *is-NNil nell*; $y \in$ *nset* (*ntl nell*); *P y* (*ntl nell*)$]\!] \Longrightarrow$ *P y nell*
shows *P x nell*
using *assms*
proof (*induct*)
case (*NNil z*)
then show *?case* **by** *force*
next
case (*NCons1 z1 z2*)
then show *?case* **by** (*fastforce simp del*: *nellist.disc*(*2*) *iff*: *nellist.disc*(*2*))
next
case (*NCons2 z1 z2 xa*)
then show *?case* **by** *auto*
qed

lemma *nset-induct′* [*consumes 1*, *case-names findnil find step*]:
assumes *major*: $x \in$ *nset nell*
and *0*: $\bigwedge$*nell. is-NNil nell* $\Longrightarrow$ *P* (*nlast nell*)
and *1*: $\bigwedge$*nell. P* (*NCons x nell*)
and *2*: $\bigwedge$ *x′ nell.* $[\![$ $x \in$ *nset nell*; *P nell* $]\!] \Longrightarrow$ *P* (*NCons x′ nell*)
shows *P nell*
using *major*
proof (*induct y*≡*x nell rule*: *nellist-set-induct*)
case (*findnil nell*)
then show *?case* **using** *0 1 2* **by** (*metis nellist.collapse*(*1*) *nellist.map-disc-iff nellist.map-sel*(*1*))
next
case (*find nell*)
then show *?case* **by** (*metis 1 nellist.collapse*(*2*))
next
case (*step nell*)
then show *?case* **by** (*metis 2 nellist.collapse*(*2*))
qed

lemma *nset-induct* [*consumes 1*, *case-names findnil find step*, *induct set*: *nset*]:
assumes *major*: $x \in$ *nset nell*
and *findnil*: $\bigwedge$*nell. is-NNil nell* $\Longrightarrow$ *P* (*nlast nell*)
and *find*: $\bigwedge$*nell. P* (*NCons x nell*)
and *step*: $\bigwedge$ *x′ nell.* $[\![$ $x \in$ *nset nell*; $x \neq x'$; *P nell* $]\!] \Longrightarrow$ *P* (*NCons x′ nell*)
shows *P nell*
using *major*
proof (*induct rule*: *nset-induct′*)
case (*findnil nell*)
then show *?case* **by** (*simp add*: *assms*(*2*))
next
case (*find nell*)
then show *?case* **by** (*simp add*: *assms*(*3*))
next
case (*step x′ nell*)

then show *?case* **by** (*metis assms*(*4*) *find*)
qed

2.3 Connection with *'a llist*

primcorec *llist-of-nellist* :: *'a nellist* ⇒ *'a llist*
where *llist-of-nellist nell* = (*case nell of NNil b* ⇒ *LCons b LNil* |
NCons x nell' ⇒ *LCons x* (*llist-of-nellist nell'*))

context
fixes
b :: *'a*
begin
primcorec *nellist-of-llist-a* :: *'a llist* ⇒ *'a nellist* **where**
nellist-of-llist-a ll = (*case ll of LNil* ⇒ *NNil b* |
LCons x ll' ⇒ *NCons x* (*nellist-of-llist-a ll'*))
end

abbreviation *nellist-of-llist* == (λ *ll*. *nellist-of-llist-a* (*llast ll*) (*lbutlast ll*))

simps-of-case *nellist-of-llist-a-simps* [*simp*, *code*, *nitpick-simp*]: *nellist-of-llist-a.code*

lemmas *nellist-of-llist-a-LNil* = *nellist-of-llist-a-simps*(*1*)
and *nellist-of-llist-a-LCons* = *nellist-of-llist-a-simps*(*2*)

lemma *nlast-nellist-of-llist-a-lnull* [*simp*]:
lnull ll ⟹ *nlast* (*nellist-of-llist-a b ll*) = *b*
unfolding *lnull-def* **by** *simp*

declare *nellist-of-llist-a.sel*(*1*)[*simp del*]

lemma *lhd-LNil*:
lhd LNil = *undefined*
by(*simp add*: *lhd-def*)

lemma *nhd-NNil*:
nhd (*NNil b*) = *undefined*
by(*simp add*: *nhd-def*)

lemma *nhd-nellist-of-llist-a* [*simp*]:
nhd (*nellist-of-llist-a b ll*) = *lhd ll*
by (*cases ll*)
(*simp-all add*: *lhd-LNil nhd-NNil*)

lemma *ntl-nellist-of-llist-a* [*simp*]:
ntl (*nellist-of-llist-a b ll*) = *nellist-of-llist-a b* (*ltl ll*)
by(*cases ll*) *simp-all*

lemma *llist-of-nellist-eq-LNil*:

llist-of-nellist nell = *LCons* (*nlast nell*) *LNil* ⟷ *is-NNil nell*
by (*simp add*: *nellist.case-eq-if llist-of-nellist.code*)

simps-of-case *llist-of-nellist-simps* [*simp*, *code*, *nitpick-simp*]: *llist-of-nellist.code*

lemmas *llist-of-nellist-NNil* = *llist-of-nellist-simps*(*1*)
and *llist-of-nellist-NCons* = *llist-of-nellist-simps*(*2*)

declare *llist-of-nellist.sel* [*simp del*]

lemma *lhd-llist-of-nellist* [*simp*]:
¬ *is-NNil nell* ⟹ *lhd* (*llist-of-nellist nell*) = *nhd nell*
by(*cases nell*) *simp-all*

lemma *lhd-llist-of-nellist1* [*simp*]:
is-NNil nell ⟹ *lhd* (*llist-of-nellist nell*) = *nlast nell*
by (*cases nell*) *simp-all*

lemma *lhd-llist-of-nellist2* [*simp*]:
(*case nell of* (*NNil b*) ⇒ *lhd LNil* | (*NCons b nell′*) ⇒ *lhd* (*llist-of-nellist nell*)) = *nhd nell*
by (*cases nell*) (*simp-all add*: *lhd-LNil nhd-NNil*)

lemma *ltl-llist-of-nellist* [*simp*]:
¬ *is-NNil nell* ⟹ *ltl* (*llist-of-nellist nell*) = *llist-of-nellist* (*ntl nell*)
by(*cases nell*) *simp-all*

lemma *ltl-llist-of-nellist1* [*simp*]:
is-NNil nell ⟹ *ltl* (*llist-of-nellist nell*) = *LNil*
by(*cases nell*) *simp-all*

lemma *ltl-llist-of-nellist2* [*simp*]:
(*case nell of* (*NNil b*) ⇒ (*LCons b LNil*) |
(*NCons b nell′*) ⇒ *ltl* (*llist-of-nellist nell*)) = *llist-of-nellist* (*ntl nell*)
by (*simp add*: *llist-of-nellist.code nellist.case-eq-if*)

lemma *nellist-of-llist-a-cong* [*cong*]:
assumes *ll* = *ll′ lfinite ll′* ⟹ *b* = *b′*
shows *nellist-of-llist-a b ll* = *nellist-of-llist-a b′ ll′*
proof(*unfold* ‹*ll* = *ll′*›)
from *assms* **have** *lfinite ll′* ⟶ *b* = *b′* **by** *simp*
thus *nellist-of-llist-a b ll′* = *nellist-of-llist-a b′ ll′*
by(*coinduction arbitrary*: *ll′*) *auto*
qed

primcorec *snocn* :: *′a nellist* ⇒ *′a* ⇒ *′a nellist*
where *snocn nell a* =
(*case nell of* (*NNil x*) ⇒ *NCons x* (*NNil a*) |
(*NCons x nell′*) ⇒ *NCons x* (*snocn nell′ a*))

simps-of-case *snocn-code* [*code*, *simp*, *nitpick-simp*]: *snocn.code*

lemma *snocn-simps* [*simp*]:
 shows *nhd-snocn*: *nhd*(*snocn nell a*) = *nfirst nell*
 and *ntl-snocn*: *ntl*(*snocn nell a*) = (*if is-NNil nell then* (*NNil a*) *else snocn* (*ntl nell*) *a*)
by (*case-tac* [!] *nell*)
 (*auto simp add*: *nfirst-def*)

lemma *is-NNil-snocn*:
 is-NNil(*snocn nell a*) ⟷ *False*
by (*auto simp add*: *snocn-def*)

lemma *nmap-snocn-distrib*:
 nmap f (*snocn nell a*) = *snocn* (*nmap f nell*) (*f a*)
proof (*coinduction arbitrary*: *nell rule*: *nellist.coinduct-strong*)
case (*Eq-nellist nella*)
then show *?case*
by (*auto simp add*: *nellist.case-eq-if nellist.map-sel*(*1*))
qed

definition *nfinite* :: *'a nellist* ⇒ *bool*
where *nfinite nell* ≡ *lfinite* (*llist-of-nellist nell*)

lemma *nfinite-induct* [*consumes 1*, *case-names NNil NCons*]:
 assumes *nfinite nell*
 and ⋀*y*. *P* (*NNil y*)
 and ⋀*x nell*. ⟦*nfinite nell*; *P nell*⟧ ⟹ *P* (*NCons x nell*)
 shows *P nell*
using *assms*
unfolding *nfinite-def*
proof (*induct ll*≡*llist-of-nellist nell arbitrary*: *nell rule*: *lfinite-induct*)
case *LNil*
then show *?case* **by** *simp*
next
case *LCons*
then show *?case* **by** (*metis lfinite.cases lnull-def ltl-llist-of-nellist ltl-simps*(*2*)
 nellist.collapse(*1*) *nellist.exhaust-sel*)
qed

lemma
 shows *nfinite-NNil*: *nfinite* (*NNil x*)
 and *nfinite-NConsI*: *nfinite nell* ⟹ *nfinite* (*NCons x nell*)
unfolding *nfinite-def*
by *auto*

declare *nfinite-NNil* [*iff*]

lemma *is-NNil-imp-nfinite* [*simp*]:
 is-NNil nell ⟹ *nfinite nell*
using *lfinite.simps llist-of-nellist-eq-LNil* **by** (*auto simp add*: *nfinite-def*)

lemma *nfinite-NCons* [*simp*]:
  nfinite (*NCons x nell*) = *nfinite nell*
by (*simp add*: *nfinite-def*)

lemma *nfinite-ntl* [*simp*]:
  nfinite (*ntl nell*) = *nfinite nell*
by (*cases nell*) *simp-all*

lemma *nfinite-code* [*code*]:
  nfinite (*NNil x*) = *True*
  nfinite (*NCons x nell*) = *nfinite nell*
by *simp-all*

lemma *nfinite-imp-finite-nset*:
 assumes *nfinite nell*
 shows *finite* (*nset nell*)
using *assms*
by (*induct nell rule*:*nfinite-induct*) *simp-all*

lemma *nfinite-snocn* [*simp*]:
 nfinite(*snocn nell a*) ⟷ *nfinite nell*
(**is** *?lhs* ⟷ *?rhs*)
proof
 assume *?lhs* **thus** *?rhs*
 proof (*induct zs*≡*snocn nell a arbitrary*: *nell rule*: *nfinite-induct*)
 case (*NNil y*)
 then show *?case*
 by (*metis is-NNil-snocn nellist.disc*(*1*))
 next
 case (*NCons x nell*)
 then show *?case*
   by (*cases nell*) *simp-all*
 qed
next
 assume *?rhs* **thus** *?lhs*
  by (*induct rule*: *nfinite-induct*) *auto*
qed

lemma *snocn-inf*:
 ¬ *nfinite nell* ⟹ *snocn nell a* = *nell*
proof (*coinduction arbitrary*: *nell*)
case (*Eq-nellist nella*)
then show *?case*
   proof −
     have *1*: *is-NNil* (*snocn nella a*) = *is-NNil nella*
       using *Eq-nellist* **by** *auto*
     have *2*: (*is-NNil* (*snocn nella a*) ⟶ *is-NNil nella* ⟶ *nlast* (*snocn nella a*) = *nlast nella*)

by *auto*
have *3*: (¬ *is-NNil* (*snocn nella a*) ⟶ ¬ *is-NNil nella* ⟶
nhd (*snocn nella a*) = *nhd nella* ∧
(∃ *nell*. *ntl* (*snocn nella a*) = *snocn nell a* ∧ *ntl nella* = *nell* ∧ ¬ *nfinite nell*))
by (*simp add*: *Eq-nellist nellist.case-eq-if*)
from *1 2 3* **show** *?thesis* **by** *blast*
qed
qed

lemma *nfinite-nmap* [*simp*]:
nfinite (*nmap f nell*) = *nfinite nell* (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs* **thus** *?rhs*
proof (*induct zs*≡*nmap f nell arbitrary*: *nell rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*metis nellist.disc*(*1*) *nellist.map-disc-iff is-NNil-imp-nfinite*)
next
case (*NCons x nell*)
then show *?case* **by** (*metis nellist.sel*(*5*) *nfinite-ntl ntl-nmap*)
qed
next
assume *?rhs* **thus** *?lhs*
by (*induct rule*: *nfinite-induct*) *simp-all*
qed

lemma *nset-snocn-nfinite* [*simp*]:
nfinite nell ⟹ *nset*(*snocn nell a*) = *nset nell* ∪ {*a*}
by (*induct rule*: *nfinite-induct*) *auto*

lemma *nset-snocn1*:
nset (*snocn nell a*) ⊆ *nset nell* ∪ {*a*}
proof (*cases nfinite nell*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then show *?thesis* **by** (*auto simp add*: *snocn-inf*)
qed

lemma *nset-snocn-conv*:
nset (*snocn nell a*) = (*if nfinite nell then nset nell* ∪ {*a*} *else nset nell*)
by (*simp add*: *snocn-inf*)

lemma *in-nset-snocn-iff*:
x ∈ *nset* (*snocn nell a*) ⟷ *x* ∈ *nset nell* ∨ *nfinite nell* ∧ *x* = *a*
by (*metis Un-iff empty-iff insert-iff nset-snocn-conv*)

lemma *llist-of-nellist-inverse-1*:
assumes ¬ *nfinite nell*
shows *nellist-of-llist-a b* (*llist-of-nellist nell*) = *snocn nell b*

using *assms*
proof (*coinduction arbitrary*: *nell*)
case (*Eq-nellist nella*)
then show *?case*
  proof −
   have *1*: *is-NNil* (*nellist-of-llist-a b* (*llist-of-nellist nella*)) = *is-NNil* (*snocn nella b*)
    by *auto*
   have *2*: (*is-NNil* (*nellist-of-llist-a b* (*llist-of-nellist nella*)) ⟶
        is-NNil (*snocn nella b*) ⟶
        nlast (*nellist-of-llist-a b* (*llist-of-nellist nella*)) = *nlast* (*snocn nella b*))
    by *simp*
   have *3*: (¬ *is-NNil* (*nellist-of-llist-a b* (*llist-of-nellist nella*)) ⟶
        ¬ *is-NNil* (*snocn nella b*) ⟶
        nhd (*nellist-of-llist-a b* (*llist-of-nellist nella*)) = *nhd* (*snocn nella b*) ∧
       (∃ *nell*.
         ntl (*nellist-of-llist-a b* (*llist-of-nellist nella*)) =
          nellist-of-llist-a b (*llist-of-nellist nell*) ∧
         ntl (*snocn nella b*) = *snocn nell b* ∧ ¬ *nfinite nell*))
    by (*metis Eq-nellist lhd-llist-of-nellist ltl-llist-of-nellist nfinite-ntl*
      nhd-nellist-of-llist-a ntl-nellist-of-llist-a snocn-inf)
   from *1 2 3* **show** *?thesis* **by** *blast*
 qed
qed

lemma *llist-of-nellist-inverse-2*:
 assumes *nfinite nell*
 shows *nellist-of-llist-a b* (*llist-of-nellist nell*) = *snocn nell b*
using *assms*
by (*induct rule*: *nfinite-induct*) *simp-all*

lemma *llist-of-nellist-inverse* [*simp*]:
 shows *nellist-of-llist-a b* (*llist-of-nellist nell*) = *snocn nell b*
using *llist-of-nellist-inverse-1 llist-of-nellist-inverse-2* **by** *fastforce*

lemma *llist-of-nellist-inverse-3*:
assumes ¬ *nfinite nell*
shows *nellist-of-llist-a* (*nlast nell*) (*lbutlast* (*llist-of-nellist nell*)) = *nell*
using *assms*
proof (*coinduction arbitrary*: *nell*)
case (*Eq-nellist nella*)
then show *?case*
  proof −
   have *1*: *is-NNil* (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) =
       is-NNil nella
    by (*metis Eq-nellist is-NNil-imp-nfinite lbutlast.disc*(*2*) *llist-of-nellist.disc-iff*
     ltl-llist-of-nellist nellist-of-llist-a.disc(*2*))
   have *2*: (*is-NNil* (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) ⟶
        is-NNil nella ⟶
        nlast (*nellist-of-llist-a* (*nlast nella*)
          (*lbutlast* (*llist-of-nellist nella*))) = *nlast nella*)

    using *Eq-nellist is-NNil-imp-nfinite* **by** *blast*
  have *3*: ($\neg$ *is-NNil* (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) $\longrightarrow$
        $\neg$ *is-NNil nella* $\longrightarrow$
          nhd (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) =
          nhd nella $\wedge$
          ($\exists$ *nell*.
            ntl (*nellist-of-llist-a* (*nlast nella*) (*lbutlast* (*llist-of-nellist nella*))) =
            nellist-of-llist-a (*nlast nell*) (*lbutlast* (*llist-of-nellist nell*)) $\wedge$
            ntl nella = *nell* $\wedge$ $\neg$ *nfinite nell*))
    by (*auto simp add*: *llist.case-eq-if Eq-nellist*)
  from *1 2 3* **show** *?thesis* **by** *blast*
 qed
qed

lemma *llist-of-nellist-inverse-4*:
assumes *nfinite nell*
shows *nellist-of-llist-a* (*nlast nell*) (*lbutlast* (*llist-of-nellist nell*)) = *nell*
using *assms*
by (*induct rule*: *nfinite-induct*) (*simp-all add*: *llist-of-nellist.code*)

lemma *llist-of-nellist-inverse-a* [*simp*]:
shows *nellist-of-llist-a* (*nlast nell*) (*lbutlast* (*llist-of-nellist nell*)) = *nell*
using *llist-of-nellist-inverse-3 llist-of-nellist-inverse-4* **by** *fastforce*

lemma *nlast-llast*:
 assumes *nfinite nell*
 shows *nlast nell* = *llast*(*llist-of-nellist nell*)
using *assms*
by (*induct rule*: *nfinite-induct*)
 (*simp-all add*: *llist-of-nellist.code*)

lemma *llist-of-nellist-inverse-b* [*simp*]:
shows *nellist-of-llist* (*llist-of-nellist nell*) = *nell*
by (*metis lappend-inf lbutlast-snoc llist-of-nellist-inverse llist-of-nellist-inverse-a*
  nfinite-def snocn-inf nlast-llast)

lemma *nellist-of-llist-a-eq* [*simp*]:
 nellist-of-llist-a b' *ll* = *NNil* $b \longleftrightarrow b = b' \wedge$ *ll* = *LNil*
by(*cases ll*) *auto*

lemma *NNil-eq-nellist-of-llist-a* [*simp*]:
 NNil b = *nellist-of-llist-a* b' *ll* $\longleftrightarrow b = b' \wedge$ *ll* = *LNil*
by(*cases ll*) *auto*

lemma *nellist-of-llist-a-inject* [*simp*]:
 nellist-of-llist-a b llx = *nellist-of-llist-a c lly* $\longleftrightarrow$ *llx* = *lly* $\wedge$ (*lfinite lly* $\longrightarrow b = c$)
 (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof(*intro iffI conjI impI*)
 assume *?rhs*
 thus *?lhs* **by**(*auto intro*: *nellist-of-llist-a-cong*)

next
  assume *?lhs*
  thus *llx* = *lly*
    by(*coinduction arbitrary*: *llx lly*)(*auto simp add*: *lnull-def neq-LNil-conv*)
  assume *lfinite lly*
  thus *b* = *c* **using** ‹*?lhs*›
    unfolding ‹*llx* = *lly*› **by**(*induct*) *simp-all*
qed

lemma *nellist-of-llist-a-inverse-1*:
assumes ¬ *lfinite ll*
shows *llist-of-nellist* (*nellist-of-llist-a b ll*) = *lappend ll* (*LCons b LNil*)
using *assms*
by (*coinduction arbitrary*: *ll*) *auto*

lemma *nellist-of-llist-a-inverse-2*:
assumes *lfinite ll*
shows *llist-of-nellist* (*nellist-of-llist-a b ll*) = *lappend ll* (*LCons b LNil*)
using *assms*
proof (*induct ll rule*: *lfinite-induct*)
 case (*LNil xs*)
 then show *?case* **by** (*simp add*: *lnull-def*)
 next
 case (*LCons xs*)
 then show *?case*
   by (*metis nellist-of-llist-a.disc*(*2*) *lappend-code*(*2*) *lhd-LCons-ltl lhd-llist-of-nellist llist-of-nellist.code llist-of-nellist.simps*(*2*) *llist-of-nellist.simps*(*3*) *ltl-llist-of-nellist nhd-nellist-of-llist-a ntl-nellist-of-llist-a*)
qed

lemma *nellist-of-llist-a-inverse* [*simp*]:
shows *llist-of-nellist* (*nellist-of-llist-a b ll*) = *lappend ll* (*LCons b LNil*)
using *nellist-of-llist-a-inverse-1 nellist-of-llist-a-inverse-2* **by** *metis*

lemma *nellist-of-llist-inverse* [*simp*]:
assumes ¬ *lnull ll*
shows *llist-of-nellist* (*nellist-of-llist ll*) = *ll*
using *assms* **by** *simp*

lemma *nmap-nellist-of-llist-a*:
  nmap f (*nellist-of-llist-a b ll*) = *nellist-of-llist-a* (*f b*) (*lmap f ll*)
by (*coinduction arbitrary*: *ll*) (*auto simp add*: *nmap-is-NNil*)

lemma *nmap-nellist-of-llist*:
assumes ¬ *lnull ll*
shows *nmap f* (*nellist-of-llist ll*) = *nellist-of-llist* (*lmap f ll*)
using *assms*
by (*metis lappend-inf lbutlast-snoc lfinite-lmap llast-lmap lmap-lbutlast nellist-of-llist-a-cong nmap-nellist-of-llist-a*)

lemma *lmap-llist-of-nellist*:
lmap f (*llist-of-nellist nell*) = *llist-of-nellist* (*nmap f nell*)
by (*metis llist.map-disc-iff llist-of-nellist.disc-iff llist-of-nellist-inverse-b nellist-of-llist-inverse nmap-nellist-of-llist*)

definition *cr-nellist* :: *'a llist* ⇒ *'a nellist* ⇒ *bool*
where *cr-nellist* = (λ *ll nell*. *llist-of-nellist nell* = *ll*)

lemma *llist-of-nellist-not-lnull*:
¬ (*lnull* (*llist-of-nellist nell*))
by *simp*

lemma *not-lnull-eq-lappend-lbutlast-llast*:
¬(*lnull ll*) ⟷ *ll* = *lappend* (*lbutlast ll*) (*LCons* (*llast ll*) *LNil*)
using *llist.collapse*(*1*) **by** *fastforce*

lemma *Domainp-help*:
¬ *lnull ll* ⟹ ∃ *nell*. *llist-of-nellist nell* = *ll*
using *nellist-of-llist-inverse* **by** *blast*

lemma *not-lnull-conv-llist-of-nellist*:
¬ *lnull ll* ⟷ (∃ *nell*. *llist-of-nellist nell* = *ll*)
using *Domainp-help llist-of-nellist-not-lnull* **by** *blast*

lemma *Domainp-cr-nellist* [*transfer-domain-rule*]:
Domainp cr-nellist = (λ*ll*. ¬(*lnull ll*))
unfolding *cr-nellist-def Domainp-iff*[*abs-def*]
using *Domainp-help* **by** *fastforce*

lemma *bi-unique-cr-nellist-help*:
llist-of-nellist nelly = *llist-of-nellist nellz* ⟹ *nelly* = *nellz*
by (*coinduction arbitrary*: *nelly nellz*)
(*metis llist-of-nellist-inverse-b*)

lemma *quotient-help-2*:
(¬ *lnull* (*llist-of-nellist nell*) ∧ *nellist-of-llist* (*llist-of-nellist nell*) = *nell*)
by (*simp add*: *bi-unique-cr-nellist-help*)

lemma *quotient-help-nellist-1*:
cr-nellist ll nell ⟶ *nellist-of-llist ll* = *nell*
by (*metis cr-nellist-def llist-of-nellist-inverse-b*)

lemma *quotient-help-nellist-2*:
(*cr-nellist* (*llist-of-nellist nell*) *nell*)
by (*simp add*: *cr-nellist-def*)

lemma *quotient-help-3-nellist*:
((¬ *lnull llx* ∧ *llx* = *lly*) =

(*cr-nellist llx* (*nellist-of-llist llx*) ∧
cr-nellist lly (*nellist-of-llist lly*) ∧
nellist-of-llist llx =
nellist-of-llist lly))
by (*metis Domainp.DomainI Domainp-cr-nellist cr-nellist-def nellist-of-llist-inverse*)

lemma *Quotient-nellist*:
Quotient (λ *llx lly*. ¬ *lnull llx* ∧ *llx* = *lly*)
nellist-of-llist llist-of-nellist cr-nellist
unfolding *Quotient-alt-def*
using *quotient-help-3-nellist quotient-help-nellist-1 quotient-help-nellist-2* **by** *blast*

setup-lifting *Quotient-nellist*

context includes *lifting-syntax*
begin

lemma *bi-unique-cr-nellist* [*transfer-rule*]:
bi-unique cr-nellist
unfolding *cr-nellist-def bi-unique-def*
by (*auto simp add*: *bi-unique-cr-nellist-help*)

lemma *right-total-cr-nellist* [*transfer-rule*]:
right-total cr-nellist
unfolding *cr-nellist-def right-total-def*
by *simp*

lemma *NNil-transfer* [*transfer-rule*]:
(*A* ===> *pcr-nellist A*) (λ*b*. *LCons b LNil*) *NNil*
by (*auto simp add*: *pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *NCons-transfer* [*transfer-rule*]:
(*A* ===> *pcr-nellist A* ===> *pcr-nellist A*) *LCons NCons*
by (*auto simp add*: *pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *nmap-transfer* [*transfer-rule*]:
((=) ===> *pcr-nellist* (=) ===> *pcr-nellist* (=)) *lmap nmap*
by (*auto simp add*: *cr-nellist-def nellist.pcr-cr-eq rel-fun-def lmap-llist-of-nellist*)

lemma *is-NNil-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *is-lfirst is-NNil*
by (*simp add*: *cr-nellist-def nellist.pcr-cr-eq rel-fun-def*)
(*metis lbutlast-simps*(*2*) *llast-singleton llist.disc*(*2*) *llist-of-nellist-eq-LNil*
llist-of-nellist-inverse-a nellist-of-llist-inverse)

lemma *is-NNil-nellist-of-llist-conv-is-lfirst*:
assumes ¬ *lnull lx*
shows *is-NNil*(*nellist-of-llist lx*) ⟷ *is-lfirst lx*

using *assms*
by (*cases lx*)
(*simp*, *metis lbutlast.ctr*(*1*) *lbutlast.disc-iff*(*2*) *lbutlast-eq-LNil-conv llist.disc*(*1*)
nellist-of-llist-a.disc-iff(*2*))

lemma *nfirst-transfer-a* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *lhd nfirst*
by (*simp add*: *cr-nellist-def nellist.pcr-cr-eq nfirst-def rel-fun-def llist-of-nellist.simps*(*2*))

lemma *nhd-transfer-a1* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) (λ *ll*. *if is-lfirst ll then lhd LNil else lhd ll*) *nhd*
by (*simp add*: *cr-nellist-def nellist.pcr-cr-eq rel-fun-def OO-def*)
(*metis lbutlast-simps*(*2*) *lhd-llist-of-nellist llist-of-nellist-eq-LNil nhd-nellist-of-llist-a*
quotient-help-2)

lemma *ntl-transfer* [*transfer-rule*]:
(*pcr-nellist A* ===> *pcr-nellist A*) (λ *ll*. *if is-lfirst ll then ll else ltl ll*) *ntl*
proof (*auto simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def intro*!: *llist-all2-ltlI*
lfinite-LConsI dest: *llist-all2-lnullD*)
show $\bigwedge$*b y*. *llist-all2 A* (*LCons b LNil*) (*llist-of-nellist y*) $\Longrightarrow$
llist-all2 A (*LCons b LNil*) (*llist-of-nellist* (*ntl y*))
using *llist-all2-ltlI*
by (*metis eq-LConsD is-NNil-ntl llist-all2-LNil1 llist-of-nellist.code llist-of-nellist.simps*(*3*)
llist-of-nellist-eq-LNil ltl-llist-of-nellist nlast-ntl)
next
show $\bigwedge$*x y*. $\forall$ *b*. *x* $\neq$ *LCons b LNil* $\Longrightarrow$ *llist-all2 A x* (*llist-of-nellist y*) $\Longrightarrow$
llist-all2 A (*ltl x*) (*llist-of-nellist* (*ntl y*))
by (*metis lhd-LCons-ltl llist-all2-LNil2 llist-all2-lnullD llist-all2-ltlI*
llist-of-nellist.disc-iff ltl-llist-of-nellist ltl-llist-of-nellist1)
qed

lemma *nfinite-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *lfinite nfinite*
by (*auto simp add*: *nellist.pcr-cr-eq cr-nellist-def nfinite-def rel-fun-def*)

lemma *llist-of-nellist-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *id llist-of-nellist*
by (*simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)

lemma *nellist-of-llist-a-transfer* [*transfer-rule*]:
((=) ===> (=) ===> *pcr-nellist* (=)) (λ*b ll*. *lappend ll* (*LCons b LNil*)) *nellist-of-llist-a*
by (*auto simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def*)

lemma *nlast-nellist-of-llist-a-lfinite* [*simp*]:
lfinite ll $\Longrightarrow$ *nlast* (*nellist-of-llist-a b ll*) = *b*
by(*induct rule*: *lfinite.induct*) *simp-all*

lemma *snocn-transfer* [*transfer-rule*]:
(*pcr-nellist* (*A*) ===> (*A*) ===> *pcr-nellist* (*A*)) (λ *ll a*. *lappend ll* (*LCons a LNil*)) *snocn*
unfolding *rel-fun-def*

by (*auto simp add*: *pcr-nellist-def nellist.pcr-cr-eq cr-nellist-def OO-def*)
(*metis LNil-transfer llist.rel-intros*(*2*) *llist-all2-lappendI llist-of-nellist-inverse*
nellist-of-llist-a-inverse)

lemma *nellist-all2-help-a*:
llist-all2 P (*llist-of-nellist nella*) (*llist-of-nellist nellb*) $\Longrightarrow$ *nellist-all2 P nella nellb*
by (*coinduction arbitrary*: *nella nellb*)
(*metis lhd-llist-of-nellist llist.disc*(*1*) *llist-all2-LCons-LCons llist-all2-LNil1*
llist-all2-LNil2 llist-all2-lhdD llist-all2-ltlI llist-of-nellist.disc-iff
llist-of-nellist-eq-LNil ltl-llist-of-nellist ltl-simps(*2*))

lemma *nellist-all2-help-b*:
nellist-all2 P nella nellb $\Longrightarrow$ *llist-all2 P* (*llist-of-nellist nella*) (*llist-of-nellist nellb*)
proof (*coinduction arbitrary*: *nella nellb*)
case *LNil*
then show *?case*
by *simp*
next
case *LCons*
then show *?case*
by *auto*
(*metis llist-of-nellist.simps*(*2*) *nellist.case-eq-if nellist.rel-sel*,
metis llist-all2-LNil1 ltl-llist-of-nellist ltl-llist-of-nellist1 nellist.rel-sel)
qed

lemma *nellist-all2-transfer* [*transfer-rule*]:
((=) ===> *pcr-nellist* (=) ===> *pcr-nellist* (=) ===> (=)) *llist-all2 nellist-all2*
unfolding *nellist.pcr-cr-eq cr-nellist-def*
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*

lift-definition *nappend* :: *'a nellist* $\Rightarrow$ *'a nellist* $\Rightarrow$ *'a nellist*
is (λ *llx lly*. *lappend llx lly*)
by *auto*

lemma *llist-all2-llist-of-nellist-1*:
assumes $\neg$ *lnull y*
llist-all2 (λx *z*. *llist-of-nellist z* = *x*) *llist1 y*
llist-all2 (λx *z*. *llist-of-nellist z* = *x*) *llist2 y*
shows *llist1* = *llist2*
proof –
have *1*: *llist-all2* (λx *z*. *llist-of-nellist z* = *x*) *llist1 y* =
llist-all2 (=) *llist1* (*lmap llist-of-nellist y*)
using *llist-all2-lmap2*[*of* (=) *llist1 llist-of-nellist y*]
using *llist-all2-mono* **by** *fastforce*
have *2*: *llist-all2* (λx *z*. *llist-of-nellist z* = *x*) *llist2 y* =
llist-all2 (=) *llist2* (*lmap llist-of-nellist y*)
using *llist-all2-lmap2*[*of* (=) *llist2 llist-of-nellist y*]
using *llist-all2-mono* **by** *fastforce*
show *?thesis* **using** *assms*

by (*metis* (*full-types*) *1 2 llist.rel-eq*)
qed

lemma *llist-all2-llist-of-nellist-2*:
assumes ¬ *lnull y*
llist-all2 (λ*z x*. *llist-of-nellist z* = *x*) *y llist1*
llist-all2 (λ*z x*. *llist-of-nellist z* = *x*) *y llist2*
shows *llist1* = *llist2*
proof −
have *1*: *llist-all2* (λ*z x*. *llist-of-nellist z* = *x*) *y llist1* =
llist-all2 (=) (*lmap llist-of-nellist y*) *llist1*
using *llist-all2-lmap1*[*of* (=)]
using *llist-all2-mono* **by** *fastforce*
have *2*: *llist-all2* (λ*z x*. *llist-of-nellist z* = *x*) *y llist2* =
llist-all2 (=) (*lmap llist-of-nellist y*) *llist2*
using *llist-all2-lmap1*[*of* (=)]
using *llist-all2-mono* **by** *fastforce*
show *?thesis* **using** *assms*
by (*metis* (*full-types*) *1 2 llist.rel-eq*)
qed

lift-definition *nconcat* :: ′*a nellist nellist* ⇒ ′*a nellist*
is (λ *lxs*. *lconcat lxs*)
apply (*simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)
using *llist.rel-cases mem-Collect-eq llist-all2-llist-of-nellist-1* **by** *fastforce*

lift-definition *nfilter* :: (′*a* ⇒ *bool*) ⇒ ′*a nellist* ⇒ ′*a nellist*
is λ *P ll*. (*if lnull*(*lfilter P ll*) *then ll else lfilter P ll*)
by *auto*

lift-definition *lappendn* :: ′*a llist* ⇒ ′*a nellist* ⇒ ′*a nellist*
is *lappend*
by *auto*

lift-definition *nzip* :: ′*a nellist* ⇒ ′*b nellist* ⇒ (′*a* × ′*b*) *nellist*
is (λ *llx lly*. *lzip llx lly*)
by *auto*

lift-definition *niterates* :: (′*a* ⇒ ′*a*) ⇒ ′*a* ⇒ ′*a nellist*
is (λ *f a* . *iterates f a*)
by *auto*

lift-definition *ndistinct* :: ′*a nellist* ⇒ *bool*
is *ldistinct*
by *auto*

lift-definition *nnth* :: ′*a nellist* ⇒ *nat* ⇒ ′*a*
is λ *ll n*. *lnth ll* (*the-enat* (*min* (*enat n*) ((*epred* (*llength ll*)))))

by *blast*

lift-definition *nlength* :: *'a nellist* ⇒ *enat*
is *λ ll. epred*(*llength ll*)
by *auto*

lift-definition *ndropn* :: *nat* ⇒ *'a nellist* ⇒ *'a nellist*
is *λ n ll. ldropn* (*the-enat* (*min* (*enat n*) ((*epred*(*llength ll*))))) *ll*
by *auto*
(*metis co.enat.exhaust-sel enat-the-enat iless-Suc-eq infinity-ileE leD llength-eq-0 min.cobounded1 min.cobounded2*)

lift-definition *ntake* :: *enat* ⇒ *'a nellist* ⇒ *'a nellist*
is *λ n ll. ltake* (*eSuc n*) *ll*
by *auto*

lift-definition *ntaken* :: *nat* ⇒ *'a nellist* ⇒ *'a nellist*
is *λ n ll. ltake* (*Suc n*) *ll*
using *enat-0-iff*(*2*) **by** *auto*

2.4 The nlast element *nlast*

lemma *nlast-not-nfinite*:
assumes ¬ *nfinite nell*
shows *nlast nell* = *undefined*
unfolding *nlast0-nlast*[*symmetric*]
using *assms*
by (*rule contrapos-np*)
(*induct nell rule*: *nlast0.raw-induct*[*rotated 1*, *OF refl*, *consumes 1*],
auto split: *nellist.split-asm*)

lemma *nlast-nellist-of-llist-a*:
nlast (*nellist-of-llist-a y ll*) = (*if lfinite ll then y else undefined*)
by (*simp add*: *nfinite-def nlast-not-nfinite*)

lemma *nlast-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) (*λll. if lfinite ll then llast ll else undefined*) *nlast*
by (*auto simp add*: *cr-nellist-def pcr-nellist-def nlast-nellist-of-llist-a OO-def dest*: *llist-all2-lfiniteD*)
(*simp add*: *llist.rel-eq nfinite-def nlast-llast nlast-not-nfinite rel-funI*)

lemma *nlast-nmap* [*simp*]:
nfinite nell ⟹ *nlast* (*nmap f nell*) = *f* (*nlast nell*)
by (*induct rule*: *nfinite-induct*)
(*auto simp add*: *nellist.map-sel*(*1*))

lemma *nset-nlast*:
nfinite nell ⟹ *nlast nell* ∈ *nset nell*
by (*induct rule*: *nfinite-induct*)
(*simp-all add*: *nellist.set-sel*(*3*))

2.5 *nset*

lemma *lset-llist-of-nellist-1*:
assumes *nfinite nell*
shows *lset* (*lbutlast* (*llist-of-nellist nell*)) ∪ {*nlast nell*}= *nset nell* (**is** *?lhs* = *?rhs*)
proof(*intro set-eqI iffI*)
  fix *x*
  assume *x* ∈ *?lhs*
  thus *x* ∈ *?rhs*
    proof –
     have *1*: *nlast nell* = *x* ⟹ *x* ∈ *nset nell*
       using *assms nset-nlast* **by** *blast*
     have *2*: *nlast nell* = *x* ⟹ *x* ∈ *nset nell*
       unfolding *nlast0-nlast*[*symmetric*] **by** (*simp add*: *1*)
     have *3*: *x* ∈ *lset* (*lbutlast*(*llist-of-nellist nell*)) ⟹ *x* ∈ *nset nell*
       proof (*induct lbutlast* (*llist-of-nellist nell*) *arbitrary*: *nell rule*: *llist-set-induct*)
       case *find*
       then show *?case*
          by (*metis lbutlast.disc*(*1*) *llist.disc*(*1*) *llist-of-nellist-inverse-b ltl-llist-of-nellist1*
             nellist.set-sel(*2*) *nhd-nellist-of-llist-a*)
       next
       case (*step y*)
       then show *?case*
         by (*metis lbutlast.disc*(*1*) *lbutlast-ltl llist.disc*(*1*) *ltl-llist-of-nellist ltl-llist-of-nellist1*
            nellist.disc(*1*) *nellist.exhaust-sel nellist.set-intros*(*3*))
       qed
     show *?thesis*
       using *2 3* ‹*x* ∈ *lset* (*lbutlast* (*llist-of-nellist nell*)) ∪ {*nlast nell*}› **by** *blast*
  qed
next
  fix *x*
  assume *x* ∈ *?rhs*
  thus *x* ∈ *?lhs*
  proof(*induct rule*: *nellist-set-induct*)
    case (*findnil nell*)
    then show *?case*
     by (*cases nell*) *auto*
    next
    case (*find nell*)
    thus *?case*
     by (*metis UnI1 lbutlast.disc-iff*(*2*) *llist.set-sel*(*1*) *ltl-llist-of-nellist*
        nhd-nellist-of-llist-a quotient-help-2)
    next
    case *step*
    thus *?case*
     by (*metis Un-iff in-lset-ltlD lbutlast-ltl ltl-llist-of-nellist nlast-ntl*)
  qed
qed

lemma *lset-llist-of-nellist-2*:
assumes ¬*nfinite nell*

shows *lset* (*lbutlast* (*llist-of-nellist nell*)) = *nset nell* (**is** *?lhs* = *?rhs*)
proof(*intro set-eqI iffI*)
  fix *x*
  assume *x* ∈ *?lhs*
  thus *x* ∈ *?rhs*
    proof −
     have *3*: *x* ∈ *lset* (*lbutlast* (*llist-of-nellist nell*)) ⟹ *x* ∈ *nset nell*
       proof (*induct lbutlast* (*llist-of-nellist nell*) *arbitrary*: *nell rule*: *llist-set-induct*)
       case *find*
       then show *?case*
        by (*metis lbutlast.disc*(*1*) *llist.disc*(*1*) *llist-of-nellist-inverse-a ltl-llist-of-nellist1*
          nellist.set-sel(*2*) *nhd-nellist-of-llist-a*)
       next
       case (*step y*)
       then show *?case*
         by (*metis lbutlast.disc*(*1*) *lbutlast-ltl llist.disc*(*1*) *ltl-llist-of-nellist*
           ltl-llist-of-nellist1 nellist.collapse(*2*) *nellist.set-intros*(*3*))
       qed
     show *?thesis*
       using *3* ‹*x* ∈ *lset* (*lbutlast* (*llist-of-nellist nell*))› **by** *blast*
  qed
next
  fix *x*
  assume *x* ∈ *?rhs*
  thus *x* ∈ *?lhs*
  using *assms*
  proof(*induct rule*: *nellist-set-induct*)
    case (*findnil nell*)
    then show *?case*
     by (*cases nell*) *auto*
    next
    case (*find nell*)
    thus *?case*
      by (*metis lbutlast-not-lfinite lhd-llist-of-nellist llist.set-sel*(*1*) *llist-of-nellist-not-lnull*
        nfinite-def)
    next
    case *step*
    thus *?case*
     by (*metis in-lset-ltlD lbutlast-ltl ltl-llist-of-nellist nfinite-ntl*)
  qed
qed

lemma *lset-llist-of-nellist* [*simp*]:
 (*if nfinite nell then lset* (*lbutlast*(*llist-of-nellist nell*)) ∪ {*nlast nell*}
   else lset (*lbutlast* (*llist-of-nellist nell*))) = *nset nell*
using *lset-llist-of-nellist-1 lset-llist-of-nellist-2* **by** *auto*

lemma *lset-llist-of-nellist-a* [*simp*]:
 lset(*llist-of-nellist nell*) = *nset nell*
proof (*cases nfinite nell*)

case *True*
then show *?thesis*
by (*metis lappend-lbutlast-llast-id-lfinite lbutlast-lfinite llist.simps*(*19*) *llist-of-nellist-not-lnull lset-LNil lset-lappend-lfinite lset-llist-of-nellist-1 nfinite-def nlast-llast*)
next
case *False*
then show *?thesis* **by** (*metis lbutlast-not-lfinite lset-llist-of-nellist-2 nfinite-def*)
qed

lemma *nset-nellist-of-llist-a* [*simp*]:
shows *nset* (*nellist-of-llist-a b ll*) = (*if lfinite ll then lset ll* ∪ {*b*} *else lset ll*)
proof (*cases lfinite ll*)
case *True*
then show *?thesis*
by (*metis llist.simps*(*19*) *lset-LNil lset-lappend-lfinite lset-llist-of-nellist-a nellist-of-llist-a-inverse*)
next
case *False*
then show *?thesis*
by (*metis lappend-inf lset-llist-of-nellist-a nellist-of-llist-a-inverse*)
qed

lemma *nset-transfer* [*transfer-rule*]:
(*pcr-nellist* (=) ===> (=)) *lset nset*
by(*auto simp add*: *cr-nellist-def nellist.pcr-cr-eq*)

end

2.6 *nmap*

lemma *nmap-eq-NCons-conv*:
nmap f nellx = *NCons y nelly* ⟷
(∃ *z nellz*. *nellx* = *NCons z nellz* ∧ *f z* = *y* ∧ *nmap f nellz* = *nelly*)
by(*cases nellx*) *simp-all*

lemma *NCons-eq-nmap-conv*:
NCons y nelly = *nmap f nellx* ⟷
(∃ *z nellz*. *nellx* = *NCons z nellz* ∧ *f z* = *y* ∧ *nmap f nellz* = *nelly*)
by(*cases nellx*) *auto*

2.7 Appending two nonempty lazy lists *nappend*

lemma *nappend-NNil* [*simp*, *code*, *nitpick-simp*]:
nappend (*NNil b*) *nell* = (*NCons b nell*)
by *transfer auto*

lemma *nappend-NCons* [*simp*, *code*, *nitpick-simp*]:
nappend (*NCons a nellx*) *nelly* = *NCons a* (*nappend nellx nelly*)
by *transfer auto*

lemma *nhd-nappend* [*simp*]:
nhd(*nappend nellx nelly*) = (*if is-NNil nellx then nlast nellx else nhd nellx*)
by (*cases nellx*) *auto*

lemma *ntl-nappend* [*simp*]:
ntl(*nappend nellx nelly*) = (*if is-NNil nellx then nelly else nappend* (*ntl nellx*) *nelly*)
by (*cases nellx*) *auto*

lemma *is-NNil-nappend*:
is-NNil(*nappend nellx nelly*) $\longleftrightarrow$ *False*
by (*cases nellx*) *auto*

lemma *nappend-assoc*:
nappend (*nappend nellx nelly*) *nellz* = *nappend nellx* (*nappend nelly nellz*)
by *transfer* (*auto simp add*: *split-beta lappend-assoc*)

lemma *nmap-nappend-distrib*:
nmap f (*nappend nellx nelly*) = *nappend* (*nmap f nellx*) (*nmap f nelly*)
by *transfer* (*auto simp add*: *split-beta lmap-lappend-distrib*)

lemma *nlast-nappend*:
nlast (*nappend nellx nelly*) = (*if nfinite nellx then nlast nelly else nlast nellx*)
by *transfer* (*auto simp add*: *llast-lappend*)

lemma *nfinite-nappend*:
nfinite (*nappend nellx nelly*) $\longleftrightarrow$ *nfinite nellx* $\wedge$ *nfinite nelly*
by *transfer auto*

lemma *nappend-inf*:
$\neg$ *nfinite nellx* $\Longrightarrow$ *nappend nellx nelly* = *nellx*
by *transfer* (*auto simp add*: *lappend-inf*)

lemma *nappend-snocn-inf*:
assumes $\neg$ *nfinite nell*
shows *nappend nell* (*NNil a*) = *snocn nell a*
using *assms*
by (*simp add*: *nappend-inf snocn-inf*)

lemma *nappend-snocn-finite*:
assumes *nfinite nell*
shows *nappend nell* (*NNil a*) = *snocn nell a*
using *assms*
by (*induct rule*: *nfinite-induct*) *simp-all*

lemma *nappend-snocn*:
nappend nell (*NNil a*) = *snocn nell a*
by (*meson nappend-snocn-finite nappend-snocn-inf*)

lemma *split-nellist-first*:

assumes *x* ∈ *nset nell*
shows *nell* = (*NNil x*) ∨ (∃ *ys*. *nell* = *nappend* (*NNil x*) *ys*) ∨
(∃*ys*. *nell* = *nappend ys* (*NNil x*) ∧ *nfinite ys* ∧ *x* ∉ *nset ys*) ∨
(∃*ys zs*. *nell* = *nappend ys* (*NCons x zs*) ∧ *nfinite ys* ∧ *x* ∉ *nset ys*)
using *assms*
by *transfer*
(*auto*,
metis eq-LConsD lhd-lappend llist.disc(*1*) *llist.expand split-llist-first*)

lemma *split-nellist*:
assumes *x* ∈ *nset nell*
shows *nell* = (*NNil x*) ∨ (∃ *ys*. *nell* = *nappend* (*NNil x*) *ys*) ∨
(∃ *ys zs*. *nell* = *nappend ys* (*NCons x zs*) ∧ *nfinite ys*) ∨
(∃ *ys*. *nell* = *nappend ys* (*NNil x*) ∧ *nfinite ys*)
using *assms*
by (*meson split-nellist-first*)

lemma *split-nellist-a*:
assumes *nell* = (*NNil x*) ∨ (∃ *ys*. *nell* = *nappend* (*NNil x*) *ys*) ∨
(∃ *ys zs*. *nell* = *nappend ys* (*NCons x zs*) ∧ *nfinite ys*) ∨
(∃ *ys*. *nell* = *nappend ys* (*NNil x*) ∧ *nfinite ys*)
shows *x* ∈ *nset nell*
proof −
have *1*: *nell* = (*NNil x*) ⟹ *x* ∈ *nset nell*
by *simp*
have *2*: (∃ *ys*. *nell* = *nappend* (*NNil x*) *ys*) ⟹ *x* ∈ *nset nell*
by *auto*
have *3*: (∃ *ys zs*. *nell* = *nappend ys* (*NCons x zs*) ∧ *nfinite ys*) ⟹ *x* ∈ *nset nell*
by *transfer auto*
have *4*: (∃ *ys*. *nell* = *nappend ys* (*NNil x*) ∧ *nfinite ys*) ⟹ *x* ∈ *nset nell*
by *transfer auto*
show *?thesis* **using** *1 2 3 4 assms* **by** *blast*
qed

2.8 Appending a nonempty lazy list to a lazy list *lappendn*

lemma *lappendn-LNil* [*simp*, *code*, *nitpick-simp*]:
lappendn LNil nell = *nell*
by *transfer auto*

lemma *lappendn-LCons* [*simp*, *code*, *nitpick-simp*]:
lappendn (*LCons x ll*) *nell* = *NCons x* (*lappendn ll nell*)
by *transfer auto*

lemma *nlast-lappendn-lfinite* [*simp*]:
lfinite ll ⟹ *nlast* (*lappendn ll nell*) = *nlast nell*
by *transfer*
(*auto simp add*: *llast-lappend*)

lemma *nset-lappendn-lfinite* [*simp*]:

lfinite ll $\Longrightarrow$ *nset* (*lappendn ll nell*) = *lset ll* $\cup$ *nset nell*
by *transfer auto*

lemma *nlength-nappend* [*simp*]:
nlength (*nappend nellx nelly*) = *nlength nellx* + *nlength nelly* +*1*
by *transfer*
(*auto*, *metis co.enat.exhaust-sel epred-iadd1 iadd-Suc-right llength-eq-0 plus-1-eSuc*(*2*))

lemma *nfinite-nlength-enat*:
assumes *nfinite nell*
shows $\exists$ *n*. *nlength nell* = *enat n*
using *assms*
by *transfer* (*metis epred-conv-minus idiff-enat-enat lfinite-llength-enat one-enat-def*)

lemma *nlength-eq-enat-nfiniteD*:
nlength nell = *enat n* $\Longrightarrow$ *nfinite nell*
by *transfer* (*metis epred-Infty llength-eq-enat-lfiniteD not-lfinite-llength*)

lemma *nfinite-conv-nlength-enat*:
nfinite nell $\longleftrightarrow$ ($\exists$ *n*. *nlength nell* = *enat n*)
using *nfinite-nlength-enat nlength-eq-enat-nfiniteD* **by** *blast*

2.9 The length of a nonempty lazy list *nlength*

lemma [*simp*, *nitpick-simp*]:
shows *nlength-NNil*: *nlength* (*NNil b*) = *0*
and *nlength-NCons*: *nlength* (*NCons x nell*) = *eSuc* (*nlength nell*)
by (*transfer*, *simp*) (*transfer*, *auto*)

lemma *llength-llist-of-nellist* [*simp*]:
epred(*llength* (*llist-of-nellist nell*)) = *nlength nell*
by *transfer auto*

lemma *nlength-nmap* [*simp*]:
nlength (*nmap f nell*) = *nlength nell*
by *transfer simp*

definition *gen-nlength* :: *nat* $\Rightarrow$ *'a nellist* $\Rightarrow$ *enat*
where *gen-nlength n nell* = *enat n* + *nlength nell*

lemma *gen-nlength-code* [*code*]:
gen-nlength n (*NNil b*) = *enat n*
gen-nlength n (*NCons x nell*) = *gen-nlength* (*n* + *1*) *nell*
by (*simp-all add*: *gen-nlength-def iadd-Suc eSuc-enat*[*symmetric*] *iadd-Suc-right*)

lemma *nlength-code* [*code*]:
nlength = *gen-nlength 0*
by(*simp add*: *gen-nlength-def fun-eq-iff zero-enat-def*)

2.10 The nth element of a nonempty lazy list *nnth*

lemma *nnth-NNil* [*nitpick-simp*]:
nnth (*NNil b*) *n* = *b*
by *transfer simp*

lemma *nnth-NCons*:
nnth (*NCons x nell*) *n* = (*case n of 0* ⇒ *x* | *Suc n′* ⇒ *nnth nell n′*)
by (*transfer fixing*: *n*)
(*auto simp add*: *lnth-LCons Nitpick.case-nat-unfold zero-enat-def min-enat1-conv-enat*,
metis enat-0-iff(*1*) *less-not-refl3 llength-eq-0 min-def min-enat1-conv-enat*,
metis enat-min-eq-0-iff min-enat1-conv-enat not-gr-zero the-enat.simps the-enat-0,
metis One-nat-def epred-enat epred-min min-enat1-conv-enat the-enat.simps)

lemma *nnth-code* [*simp*, *nitpick-simp*, *code*]:
shows *nnth-0*: *nnth* (*NCons x nell*) *0* = *x*
and *nnth-Suc-NCons*: *nnth* (*NCons x nell*) (*Suc n*) = *nnth nell n*
by(*simp-all add*: *nnth-NCons*)

lemma *lnth-llist-of-nellist* [*simp*]:
lnth (*llist-of-nellist nell*) (*the-enat* (*min* (*enat n*) ((*epred*(*llength* (*llist-of-nellist nell*))))))
= *nnth nell n*
by *transfer auto*

lemma *nnth-nmap* [*simp*]:
enat n ≤ *nlength nell* ⟹ *nnth* (*nmap f nell*) *n* = *f* (*nnth nell n*)
by *transfer*
(*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lmap lnth-lmap min.orderE*
the-enat.simps)

lemma *nhd-conv-nnth*:
¬ *is-NNil nell* ⟹ *nhd nell* = *nnth nell 0*
by (*metis nellist.collapse*(*2*) *nnth-0*)

lemmas *nnth-0-conv-nhd* = *nhd-conv-nnth*[*symmetric*]

lemma *nnth-ntl*:
nnth (*ntl nell*) *n* = *nnth nell* (*Suc n*)
by (*metis nellist.exhaust-sel nellist.sel*(*4*) *nnth-NNil nnth-Suc-NCons*)

lemma *in-nset-conv-nnth*:
x ∈ *nset nell* ⟷ (∃ *n*. *enat n* ≤ *nlength nell* ∧ *nnth nell n* = *x*)
by *transfer*
(*metis eSuc-epred iless-Suc-eq in-lset-conv-lnth llength-eq-0 min-absorb1 the-enat.simps*)

lemma *nnth-beyond*:
nlength nell < *enat n* ⟹ *nnth nell n* = *nlast nell*
by *transfer*
(*metis co.enat.exhaust-sel epred-llength less-enatE lfinite-ltl llast-conv-lnth llength-eq-0*
llength-eq-enat-lfiniteD min.absorb4 the-enat.simps)

lemma *exists-Pred-nnth-nset*:
($\exists$ $x \in$ *nset nell*. *P x*) = ($\exists$ *n*. *n*$\leq$ *nlength nell* $\wedge$ *P* (*nnth nell n*))
by (*metis in-nset-conv-nnth*)

lemma *nset-conv-nnth*:
nset nell = {*nnth nell n*| *n*. *enat n* $\leq$ *nlength nell*}
by (*auto simp add*: *in-nset-conv-nnth*)

lemma *nnth-nappend1*:
enat n $\leq$ *nlength nellx* $\Longrightarrow$ *nnth* (*nappend nellx nelly*) *n* = *nnth nellx n*
proof (*induct n arbitrary*: *nellx*)
case *0*
then show *?case*
by (*metis is-NNil-def is-NNil-nappend nellist.sel*(*1*) *nhd-nappend nnth-0-conv-nhd nnth-NNil*)
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis*
using *Suc.prems enat-0-iff*(*1*) **by** *auto*
next
case (*NCons x21 x22*)
then show *?thesis*
using *Suc.hyps Suc.prems Suc-ile-eq* **by** *auto*
qed
qed

lemma *nnth-nappend2*:
$[\![$*nlength nellx* = *enat k*; *k*<*n*$]\!]$ $\Longrightarrow$ *nnth* (*nappend nellx nelly*) *n* = *nnth nelly* (*n* −*Suc k*)
proof (*induct n arbitrary*: *nellx k*)
case *0*
then show *?case* **by** *blast*
next
case (*Suc n*)
then show *?case*
by (*cases nellx*)
(*auto simp add*: *eSuc-def zero-enat-def split*: *enat.split-asm*)
qed

lemma *nnth-nappend*:
nnth (*nappend nellx nelly*) *n* =
(*if enat n* $\leq$ *nlength nellx then nnth nellx n else nnth nelly* (*n*− *Suc*(*the-enat*(*nlength nellx*))))
by (*cases nlength nellx*)
(*auto simp add*: *nnth-nappend1 nnth-nappend2*)

lemma *nnth-nlast*:
nfinite nell $\Longrightarrow$ *nlast nell* = *nnth nell* (*the-enat* (*nlength nell*))
by *transfer*
(*simp*,

metis co.enat.exhaust-sel enat-the-enat ile-eSuc infinity-ileE lfinite-llength-enat llast-conv-lnth llength-eq-0 min.idem)

2.11 *ntake*

lemma *ntake-NNil* [*simp*, *code*, *nitpick-simp*]:
ntake n (*NNil b*) = (*NNil b*)
by *transfer auto*

lemma *ntake-0* [*simp*]:
ntake 0 nell = (*NNil* (*nfirst nell*))
by *transfer* (*auto simp add*: *ltake.ctr*(*2*))

lemma *ntake-Suc-NCons* [*simp*]:
ntake (*eSuc n*) (*NCons x nell*) = (*NCons x* (*ntake n nell*))
by *transfer auto*

lemma *ntake-Suc*:
ntake (*eSuc n*) *nell* =
(*case nell of* (*NNil b*) ⇒ (*NNil b*) | (*NCons x nell′*) ⇒ (*NCons x* (*ntake n nell′*)))
by (*cases nell*) *simp-all*

lemma *is-NNil-ntake* [*simp*]:
is-NNil(*ntake n nell*) ⟷ *is-NNil nell* ∨ *n=0*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x nell1*)
then show *?thesis*
proof (*cases n*)
case (*enat nat*)
then show *?thesis*
by (*metis NCons enat-coexhaust nellist.disc*(*1*) *nellist.disc*(*2*) *ntake-0 ntake-Suc-NCons*)
next
case *infinity*
then show *?thesis*
by (*metis NCons eSuc-infinity i0-ne-infinity nellist.disc*(*2*) *ntake-Suc-NCons*)
qed
qed
lemma *ntake-eq-NNil-iff* [*simp*]:
ntake n nell = (*NNil x*) ⟷ *nell* = (*NNil x*) ∨ (*n* = *0* ∧ *nfirst nell* = *x*)
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis*
using *ntake-0* **by** *fastforce*
next
case (*NCons x nell1*)
then show *?thesis*
proof (*cases n*)

case (*enat nat*)
then show *?thesis*
by (*metis NCons is-NNil-ntake nellist.disc*(*1*) *nellist.disc*(*2*) *nellist.inject*(*1*) *ntake-0*)
next
case *infinity*
then show *?thesis*
by (*metis NCons eSuc-infinity infinity-ne-i0 nellist.distinct*(*1*) *ntake-Suc-NCons*)
qed
qed

lemma *NNil-eq-ntake-iff* [*simp*]:
$(NNil\ x) = ntake\ n\ nell \longleftrightarrow nell = (NNil\ x) \vee (n = 0 \wedge nfirst\ nell = x)$
by (*metis ntake-eq-NNil-iff*)

lemma *ntake-NCons* [*code*, *nitpick-simp*]:
$ntake\ n\ (NCons\ x\ nell) = (case\ n\ of\ 0 \Rightarrow (NNil\ x) \mid (eSuc\ n') \Rightarrow (NCons\ x\ (ntake\ n'\ nell))\ )$
by (*simp add*: *co.enat.case-eq-if*)
(*metis eSuc-epred nellist.simps*(*6*) *nfirst-def ntake-Suc-NCons*)

lemma *nhd-ntake* [*simp*]:
$n \neq 0 \Longrightarrow nhd(ntake\ n\ nell) = nhd\ nell$
unfolding *nhd-def*
by (*simp add*: *nellist.case-eq-if*)
(*metis* (*no-types*, *lifting*) *co.enat.case-eq-if nellist.collapse*(*2*) *nellist.sel*(*3*) *ntake-NCons*)

lemma *ntl-ntake*:
$n \neq 0 \Longrightarrow ntl(ntake\ n\ nell) = ntake\ (epred\ n)\ (ntl\ nell)$
by (*cases nell*) (*simp*, *metis eSuc-epred nellist.sel*(*5*) *ntake-Suc-NCons*)

lemma *ntl-ntake-0*:
$ntl(ntake\ 0\ nell) = (NNil\ (nfirst\ nell))$
by *simp*

lemma *ntake-ntl*:
$ntake\ n\ (ntl\ nell) = ntl(ntake\ (Suc\ n)\ nell)$
by (*simp add*: *enat-0-iff*(*1*) *ntl-ntake*)

lemma *nlength-ntake* [*simp*]:
$nlength\ (ntake\ n\ nell) = min\ n\ (nlength\ nell)$
by *transfer simp*

lemma *ntake-nmap* [*simp*]:
$ntake\ n\ (nmap\ f\ nell) = nmap\ f\ (ntake\ n\ nell)$
by *transfer simp*

lemma *ntake-ntake* [*simp*]:
$ntake\ n\ (ntake\ m\ nell) = ntake\ (min\ n\ m)\ nell$
by *transfer simp*

lemma *nset-ntake*:

nset (*ntake n nell*) ⊆ *nset nell*
by *transfer* (*simp add*: *lset-ltake*)

lemma *ntake-all*:
nlength nell ≤ *m* ⟹ *ntake m nell* = *nell*
by *transfer* (*auto*, *metis eSuc-epred eSuc-ile-mono llength-eq-0 ltake-all*)

lemma *nfinite-ntake* [*simp*]:
nfinite (*ntake n nell*) ⟷ *nfinite nell* ∨ *n* < ∞
by *transfer* (*metis Extended-Nat.eSuc-mono eSuc-infinity lfinite-ltake*)

lemma *ntake-nappend1*:
n ≤ *nlength nellx* ⟹ *ntake n* (*nappend nellx nelly*) = *ntake n nellx*
by *transfer* (*auto*, *metis eSuc-epred eSuc-ile-mono llength-eq-0 ltake-lappend1*)

lemma *ntake-nappend2*:
assumes *nlength nellx* < *n*
shows *ntake n* (*nappend nellx nelly*) = *nappend nellx* (*ntake* (*n* − *nlength nellx* −*1*) *nelly*)
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis* **using** *assms*
by (*metis eSuc-le-iff eSuc-minus-1 idiff-0-right ileI1 nappend-NNil nlength-NNil ntake-Suc-NCons*)
next
case (*NCons x21 x22*)
then show *?thesis*
proof (*cases nfinite nellx*)
case *True*
then show *?thesis*
using *assms*
proof (*transfer*)
fix *llxa* :: *'a llist*
fix *na*
fix *llya* :: *'a llist*
assume *a0*: ¬ *lnull llxa* ∧ *llxa* = *llxa*
assume *a1*: *lfinite llxa*
assume *a2*: *epred* (*llength llxa*) < *na*
assume *a3*: ¬ *lnull llya* ∧ *llya* = *llya*
show ¬ *lnull* (*ltake* (*eSuc na*) (*lappend llxa llya*)) ∧
ltake (*eSuc na*) (*lappend llxa llya*) =
lappend llxa (*ltake* (*eSuc* (*na* − *epred* (*llength llxa*) − *1*)) *llya*)
proof −
have *1*: ¬ *lnull* (*ltake* (*eSuc na*) (*lappend llxa llya*))
using *a0* **by** *force*
have *2*: *ltake* (*eSuc na*) (*lappend llxa llya*) =
lappend (*ltake* (*eSuc na*) *llxa*) (*ltake* ((*eSuc na*) − *llength llxa*) *llya*)
by (*meson ltake-lappend*)
have *21*: *llength llxa* ≤ (*eSuc na*)
by (*metis a0 a2 co.enat.exhaust-sel eSuc-ile-mono leD le-cases llength-eq-0*)
have *3*: *lappend* (*ltake* (*eSuc na*) *llxa*) (*ltake* ((*eSuc na*) − *llength llxa*) *llya*) =
lappend llxa (*ltake* (*eSuc na* − *llength llxa*) *llya*)

using *ltake-lappend2*[*of llxa* (*eSuc na*) *llya*] *21 2* **by** *auto*
have *4*: (*eSuc na* − *llength llxa*) = (*eSuc* (*na* − *epred* (*llength llxa*) − *1*))
by (*metis a0 a1 a2 canonically-ordered-monoid-add-class.lessE co.enat.exhaust-sel eSuc-infinity eSuc-minus-1 eSuc-minus-eSuc enat-add-sub-same llength-eq-0 llength-eq-infty-conv-lfinite*)
have *5*: (*ltake* (*eSuc na* − *llength llxa*) *llya*) =
(*ltake* (*eSuc* (*na* − *epred* (*llength llxa*) − *1*)) *llya*)
using *4* **by** *auto*
show *?thesis* **using** *1 2 3 5* **by** *fastforce*
qed
qed
next
case *False*
then show *?thesis* **using** *assms*
by (*simp add*: *nappend-inf ntake-all*)
qed
qed

lemma *ntake-eq-ntake-antimono*:
$[\![$ *ntake n nellx* = *ntake n nelly*; $m \leq n$ $]\!] \Longrightarrow$ *ntake m nellx* = *ntake m nelly*
by (*metis min.orderE ntake-ntake*)

lemma *ntake-nnth*:
assumes *enat m* $\leq$ *n*
shows (*nnth* (*ntake n nell*) *m*) = (*nnth nell m*)
using *assms*
proof (*induct m arbitrary*: *nell n*)
case *0*
then show *?case*
proof (*cases n rule*: *enat-coexhaust*)
case *0*
then show *?thesis*
using *0.prems*
by (*metis nellist.case*(*2*) *nellist.collapse*(*1*) *nellist.exhaust-sel nfirst-def nnth-0-conv-nhd nnth-NNil ntake-eq-NNil-iff*)
next
case (*eSuc n′*)
then show *?thesis*
by (*simp add*: *nellist.case-eq-if nnth-0-conv-nhd ntake-Suc*)
qed
next
case (*Suc m*)
then show *?case*
proof (*cases n rule*: *enat-coexhaust*)
case *0*
then show *?thesis*
using *Suc.prems* **by** (*simp add*: *enat-0-iff*(*1*))
next
case (*eSuc n′*)
then show *?thesis*
proof (*cases nell*)

case (*NNil x1*)
then show *?thesis* by *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
using *Suc.hyps Suc.prems Suc-ile-eq eSuc* by *force*
qed
qed
qed

2.12 *ntaken*

lemma *ntaken-NNil* [*simp*, *code*, *nitpick-simp*]:
ntaken n (*NNil b*) = (*NNil b*)
by *transfer*
(*metis eSuc-enat llist.disc*(*2*) *ltake-LNil ltake-eSuc-LCons*)

lemma *ntaken-0* [*simp*]:
ntaken 0 nell = (*NNil* (*nfirst nell*))
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** (*metis ntake-0 ntake-NNil ntaken-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis*
by *transfer* (*simp*, (*metis One-nat-def ltake-0 ltake-eSuc-LCons one-eSuc one-enat-def zero-neq-one*))
qed

lemma *ntaken-Suc-NCons* [*simp*]:
ntaken (*Suc n*) (*NCons x nell*) = (*NCons x* (*ntaken n nell*))
by *transfer* (*auto simp add*: *zero-enat-def*, *metis eSuc-enat ltake-eSuc-LCons*)

lemma *ntaken-Suc*:
ntaken (*Suc n*) *nell* =
(*case nell of* (*NNil b*) ⇒ (*NNil b*) | (*NCons x nell'*) ⇒ (*NCons x* (*ntaken n nell'*)))
by (*cases nell*) *simp-all*

lemma *is-NNil-ntaken* [*simp*]:
is-NNil(*ntaken n nell*) ⟷ *is-NNil nell* ∨ *n*=*0*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis*
by *simp*
next
case (*NCons x nell1*)
then show *?thesis*
proof (*cases n*)
case *0*
then show *?thesis*
by *simp*

next
case (*Suc nat*)
then show *?thesis*
by (*simp add*: *NCons*)
qed
qed

lemma *ntaken-eq-NNil-iff* [*simp*]:
ntaken n nell = (*NNil x*) $\longleftrightarrow$ *nell* = (*NNil x*) $\vee$ (*n* = *0* $\wedge$ *nfirst nell* = *x*)
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis*
by (*metis ntake-0 ntake-NNil ntaken-NNil*)
next
case (*NCons x nell1*)
then show *?thesis*
proof (*cases n*)
case *0*
then show *?thesis*
by (*simp add*: *NCons*)
next
case (*Suc nat*)
then show *?thesis*
using *NCons* **by** *force*
qed
qed

lemma *NNil-eq-ntaken-iff* [*simp*]:
(*NNil x*) = *ntaken n nell* $\longleftrightarrow$ *nell* = (*NNil x*) $\vee$ (*n* = *0* $\wedge$ *nfirst nell* = *x*)
by (*metis ntaken-eq-NNil-iff*)

lemma *ntaken-NCons* [*code*, *nitpick-simp*]:
ntaken n (*NCons x nell*) = (*case n of 0* $\Rightarrow$ (*NNil x*) | (*Suc n'*) $\Rightarrow$ (*NCons x* (*ntaken n' nell*)))
by (*cases n*) (*auto simp add*: *nfirst-def*)

lemma *nhd-ntaken* [*simp*]:
n $\neq$ *0* $\Longrightarrow$ *nhd*(*ntaken n nell*) = *nhd nell*
by (*cases nell*)
(*simp-all add*: *Nitpick.case-nat-unfold ntaken-NCons*)

lemma *ntl-ntaken*:
n $\neq$ *0* $\Longrightarrow$ *ntl*(*ntaken n nell*) = *ntaken* (*n*−*1*) (*ntl nell*)
by *simp-all*
(*metis Suc-pred nellist.exhaust-sel nellist.sel*(*4*) *nellist.sel*(*5*) *ntaken-NNil ntaken-Suc-NCons*)

lemma *ntl-ntaken-0*:
ntl(*ntaken 0 nell*) = (*NNil* (*nfirst nell*))
by *simp*

lemma *ntaken-ntl*:

ntaken n (*ntl nell*) = *ntl*(*ntaken* (*Suc n*) *nell*)
by (*simp add*: *enat-0-iff*(*1*) *ntl-ntaken*)

lemma *ntaken-nlength* [*simp*]:
nlength (*ntaken n nell*) = *min n* (*nlength nell*)
by *transfer simp*

lemma *ntaken-nmap* [*simp*]:
ntaken n (*nmap f nell*) = *nmap f* (*ntaken n nell*)
using *enat-0-iff*(*2*) **by** *transfer auto*

lemma *ntaken-ntaken* [*simp*]:
ntaken n (*ntaken m nell*) = *ntaken* (*min n m*) *nell*
using *enat-0-iff*(*2*) **by** *transfer auto*

lemma *nset-ntaken*:
nset (*ntaken n nell*) ⊆ *nset nell*
by *transfer* (*simp add*: *lset-ltake*)

lemma *ntaken-all*:
nlength nell ≤ *m* ⟹ *ntaken m nell* = *nell*
by *transfer*
(*auto simp add*: *zero-enat-def*, *metis eSuc-enat eSuc-epred eSuc-ile-mono llength-eq-0 ltake-all*)

lemma *ntaken-nnth*:
shows (*nnth* (*ntaken m nell*) *k*) = (*nnth nell* (*min k m*))
apply *transfer*
by (*auto simp add*: *min-def lnth-ltake ltake-all*)
(*metis co.enat.sel*(*2*) *enat-eSuc-iff enat-ord-simps*(*1*) *epred-le-epredI order-trans*,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0,
meson dual-order.strict-iff-order enat-ord-simps(*2*) *linorder-not-less order-less-le-trans*,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0)

lemma *nfinite-ntaken* [*simp*]:
nfinite (*ntaken n nell*)
by *transfer simp*

lemma *ntaken-nlast*:
nlast (*ntaken n nell*) = *nnth nell n*
using *nnth-nlast*[*of ntaken n nell*] *ntaken-nlength*[*of n nell*]
by (*metis min.absorb3 min.idem min.orderI nfinite-ntaken nnth-beyond not-less-iff-gr-or-eq*
ntaken-all ntaken-nnth the-enat.simps)

lemma *ntaken-nfirst*:
nfirst (*ntaken n nell*) = *nfirst nell*
by *transfer* (*simp add*: *enat-0-iff*(*1*))

lemma *ntaken-nappend1*:

$n \leq$ *nlength nellx* $\Longrightarrow$ *ntaken n* (*nappend nellx nelly*) = *ntaken n nellx*
by *transfer*
(*simp add*: *zero-enat-def*, *metis eSuc-enat eSuc-epred eSuc-ile-mono llength-eq-0 ltake-lappend1*)

lemma *ntaken-nappend2*:
nlength nellx < (*enat n*) $\Longrightarrow$
ntaken n (*nappend nellx nelly*) = *nappend nellx* (*ntaken* (*n* − (*the-enat*(*nlength nellx*)) −*1*) *nelly*)
proof (*induct n arbitrary*: *nellx nelly*)
case *0*
then show *?case* **using** *zero-enat-def* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis*
by *simp*
next
case (*NCons x21 x22*)
then show *?thesis* **using** *Suc* **by** *simp*
(*metis Extended-Nat.eSuc-mono eSuc-enat enat-ord-code*(*4*) *order-less-imp-not-less the-enat-eSuc*)
qed
qed

lemma *ntaken-eq-ntaken-antimono*:
⟦ *ntaken n nellx* = *ntaken n nelly*; $m \leq n$ ⟧ $\Longrightarrow$ *ntaken m nellx* = *ntaken m nelly*
by (*metis min.orderE ntaken-ntaken*)

lemma *ntake-eq-ntaken*:
assumes (*enat k*) = *m*
shows *ntake m nell* = *ntaken k nell*
using *assms* **apply** *transfer*
using *eSuc-enat* **by** *auto*

2.13 Concatenating a nonempty lazy list of nonempty lazy lists *nconcat*

lemma *nconcat-NNil* [*simp*]:
nconcat (*NNil nell*) = *nell*
by *transfer auto*

lemma *nconcat-NCons* [*simp*]:
nconcat (*NCons nell nells*) = *nappend nell* (*nconcat nells*)
by *transfer auto*

lemma *nconcat-def2*:
nconcat = *nellist-of-llist* ∘ *lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
by (*simp add*: *map-fun-def nconcat-def*)

lemma *not-null-lconcat*:
¬*lnull*((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)

by (*simp add*: *llist-of-nellist.code*)

lemma *nconcat-def3*:
((*llist-of-nellist* ∘ *nconcat*) *nells*) =
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *llist-of-nellist* ∘ *nconcat* =
llist-of-nellist ∘ *nellist-of-llist* ∘ *lconcat* ∘ *?n2l*
by (*simp add*: *nconcat-def2 rewriteL-comp-comp*)
have *2*: ¬*lnull*((*lconcat* ∘ *?n2l*) *nells*)
using *not-null-lconcat* **by** *blast*
have *3*: (*llist-of-nellist* ∘ *nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells* =
(*lconcat* ∘ *?n2l*) *nells*
using *2 nellist-of-llist-inverse* **by** *simp*
show *?thesis*
by (*metis 1 3*)
qed

lemma *nmap-lmap-inverse*:
nmap nellist-of-llist (*nellist-of-llist* (*lmap llist-of-nellist* (*llist-of-nellist nells*))) = *nells*
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nmap nellist-of-llist* (*nellist-of-llist* (*?n2l nells*)) =
nmap nellist-of-llist (*nmap llist-of-nellist nells*)
by (*simp add*: *lmap-llist-of-nellist*)
have *2*: *nmap nellist-of-llist* (*nmap llist-of-nellist nells*) =
nmap (*nellist-of-llist*∘*llist-of-nellist*) *nells*
by (*simp add*: *nellist.map-comp*)
have *3*: (*nellist-of-llist*∘*llist-of-nellist*) = *id*
by *auto*
show *?thesis*
by (*metis 1 2 3 comp-apply nellist.map-id*)
qed

lemma *lmap-nmap-inverse*:
assumes ¬ *lnull nells*
∀ *nell*∈*lset nells*. ¬ *lnull nell*
shows (*lmap llist-of-nellist* (*llist-of-nellist* (*nmap nellist-of-llist* (*nellist-of-llist nells*)))) = *nells*
proof −
let *?l2n* = *nmap nellist-of-llist* ∘ *nellist-of-llist*
have *0*: (*nmap nellist-of-llist* (*nellist-of-llist nells*)) =
nellist-of-llist (*lmap nellist-of-llist nells*)
using *nmap-nellist-of-llist assms* **by** *simp*
have *00*: (*llist-of-nellist* (*nellist-of-llist* (*lmap nellist-of-llist nells*))) =
(*lmap nellist-of-llist nells*)
using *assms* **by** *simp*
have *1*: (*lmap llist-of-nellist* (*llist-of-nellist* (*?l2n nells*))) =
(*lmap llist-of-nellist* (((*lmap nellist-of-llist nells*))))
using *0 00* **by** *auto*

have *2*: (*lmap llist-of-nellist* (((*lmap nellist-of-llist nells*)))) =
(*lmap* (*llist-of-nellist*∘*nellist-of-llist*) *nells*)
using *llist.map-comp* **by** *blast*
have *3*: ∀ *nell* ∈ *lset nells*. (*llist-of-nellist*∘*nellist-of-llist*) *nell* = *nell*
using *assms* **by** *simp*
have *4*: (*lmap* (*llist-of-nellist*∘*nellist-of-llist*) *nells*) = *nells*
using *3* **by** *auto*
show *?thesis*
using *1 2 4 0 00* **by** *presburger*
qed

lemma *nconcat-expand*:
nconcat nells =
(*if is-NNil nells then nfirst nells else nappend* (*nfirst nells*) (*nconcat* (*ntl nells*)))
by (*metis nconcat-NCons nconcat-NNil nellist.case-eq-if nellist.collapse*(*1*) *nellist.collapse*(*2*)
nfirst-def)

lemma *lmap-llist-of-nellist-nmap*:
(*lmap* (*lmap f*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))) =
(*lmap llist-of-nellist* (*lmap* (*nmap f*) (*llist-of-nellist nells*)))
proof −
have *5*: (*lmap* (*lmap f*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))) =
(*lmap* ((*lmap f*) ∘ *llist-of-nellist*) (*llist-of-nellist nells*))
using *llist.map-comp* **by** *blast*
have *6*: (*lmap* ((*lmap f*) ∘ *llist-of-nellist*) (*llist-of-nellist nells*)) =
(*lmap* (*llist-of-nellist* ∘ (*nmap f*)) (*llist-of-nellist nells*))
by (*simp add*: *lmap-llist-of-nellist*)
have *7*: (*lmap* (*llist-of-nellist* ∘ (*nmap f*)) (*llist-of-nellist nells*)) =
(*lmap llist-of-nellist* (*lmap* (*nmap f*) (*llist-of-nellist nells*)))
using *llist.map-comp*[*of llist-of-nellist* (*nmap f*) (*llist-of-nellist nells*)]
by *presburger*
show *?thesis*
using *5 6 7* **by** *presburger*
qed

lemma *nmap-nconcat* :
nmap f (*nconcat nells*) = *nconcat* (*nmap* (*nmap f*) *nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nmap f* (*nconcat nells*) =
nmap f ((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)
by (*simp add*: *nconcat-def2*)
have *2*: *nmap f* ((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*) =
nellist-of-llist (*lmap f* ((*lconcat* ∘ *?n2l*) *nells*))
using *nmap-nellist-of-llist*
using *not-null-lconcat* **by** *fastforce*
have *3*: (*lmap f* ((*lconcat* ∘ *?n2l*) *nells*)) =
lconcat (*lmap* (*lmap f*) (*?n2l nells*))
by (*simp add*: *lmap-lconcat*)

have *4*: (*nconcat* (*nmap* (*nmap f*) *nells*)) =
(*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap* (*nmap f*) *nells*)
by (*simp add*: *nconcat-def2*)
have *8*: (*lmap* (*lmap f*) (*?n2l nells*)) =
(*lmap llist-of-nellist* (*lmap* (*nmap f*) (*llist-of-nellist nells*)))
using *lmap-llist-of-nellist-nmap* **by** *auto*
show *?thesis*
using *1 2 3 4 8*
by (*metis* (*no-types*, *opaque-lifting*) *comp-eq-dest-lhs llist.map-disc-iff llist-of-nellist-inverse-b llist-of-nellist-not-lnull lmap-lconcat lmap-llist-of-nellist-nmap nconcat-def3 nellist-of-llist-inverse nmap-nellist-of-llist*)
qed

lemma *nconcat-eq-NNil*:
nconcat nells = (*NNil x*) ⟷ *nells* = (*NNil* (*NNil x*))
by (*metis is-NNil-def is-NNil-nappend nconcat-NNil nconcat-expand*)

lemma *nhd-nconcat* [*simp*]:
⟦ ¬ *is-NNil nells*; ¬ *is-NNil* (*nhd nells*) ⟧ ⟹ *nhd* (*nconcat nells*) = *nhd* (*nhd nells*)
by (*metis nconcat-NCons nellist.collapse*(*2*) *nhd-nappend*)

lemma *ntl-nconcat* [*simp*]:
⟦ ¬ *is-NNil nells*; ¬ *is-NNil* (*nhd nells*) ⟧ ⟹
ntl (*nconcat nells*) = *nappend* (*ntl* (*nhd nells*)) (*nconcat* (*ntl nells*))
by (*metis nconcat-NCons nellist.collapse*(*2*) *ntl-nappend*)

lemma *nconcat-nappend* [*simp*]:
assumes *nfinite nells*
shows *nconcat* (*nappend nells nells1*) = *nappend* (*nconcat nells*) (*nconcat nells1*)
using *assms*
by (*induct rule*:*nfinite-induct*) (*simp-all add*: *nappend-assoc*)

lemma *nconcat-eq-NCons-conv*:
nconcat nells = *NCons x nell* ⟷
nells = (*NNil* (*NCons x nell*)) ∨
(∃ *nells'*. *nells* = (*NCons* (*NNil x*) *nells'*) ∧ *nell* = *nconcat nells'*) ∨
(∃ *nell' nells''*. *nells* = (*NCons* (*NCons x nell'*) *nells''*) ∧ *nell* = *nappend nell'* (*nconcat nells''*))
proof (*cases nells*)
case (*NNil nell1*)
then show *?thesis* **by** *simp*
next
case (*NCons nell nell2*)
then show *?thesis*
proof (*cases is-NNil nell*)
case *True*
then show *?thesis*
using *NCons* **by** *simp*
(*metis nappend-NCons nappend-NNil nellist.collapse*(*1*) *nellist.sel*(*3*) *nellist.sel*(*5*))
next

case *False*
then show *?thesis*
using *NCons* **by** *simp*
(*metis nappend-NCons nellist.collapse*(*2*) *nellist.distinct*(*1*) *nellist.sel*(*3*) *nellist.sel*(*5*))
qed
qed

lemma *nlength-nconcat*:
shows *nlength* (*nconcat nells*) =
(*case nells of* (*NNil nell*) ⇒ *nlength nell* |
(*NCons nell nells1*) ⇒ *eSuc*(*nlength nell*) + *nlength* (*nconcat nells1*))
proof (*cases nells*)
case (*NNil nell1*)
then show *?thesis* **by** *simp*
next
case (*NCons nell nell2*)
then show *?thesis* **by** (*simp add*: *eSuc-plus plus-1-eSuc*(*2*))
qed

lemma *nlength-nconcat-nfinite-conv-sum*:
assumes *nfinite nells*
shows *nlength* (*nconcat nells*) =
nlength nells + ($\sum$ *i* = *0*.. (*the-enat* (*nlength nells*)). *nlength* (*nnth nells i*))
using *assms*
proof(*induct rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*simp add*: *zero-enat-def*) (*simp add*: *enat-0-iff*(*2*) *nnth-NNil*)
next
case (*NCons nell nells*)
then show *?case*
proof −
have *1*: *nlength* (*nconcat* (*NCons nell nells*)) = *1*+ *nlength nell* + *nlength* (*nconcat nells*)
by *simp*
have *2*: *nlength* (*nconcat nells*) =
nlength nells + ($\sum$ *i* = *0*.. (*the-enat* (*nlength nells*)). *nlength* (*nnth nells i*))
using *NCons.hyps*(*2*) **by** *blast*
have *3*: *nlength* (*NCons x nells*) = *1*+ *nlength nells*
by (*simp add*: *plus-1-eSuc*(*1*))
have *4*: ($\sum$ *i* = *0*..*the-enat* (*nlength* (*NCons nell nells*)). *nlength* (*nnth* (*NCons nell nells*) *i*)) =
nlength (*nnth* (*NCons nell nells*) *0*) +
($\sum$ *i* = *1*..*the-enat* (*nlength* (*NCons nell nells*)). *nlength* (*nnth* (*NCons nell nells*) *i*))
by (*simp add*: *sum.atLeast-Suc-atMost*)
have *5*: ($\sum$ *i* = *1*..*the-enat* (*nlength* (*NCons nell nells*)). *nlength* (*nnth* (*NCons nell nells*) *i*)) =
($\sum$ *i* = *1*..(*Suc* (*the-enat* (*nlength* (*nells*)))). *nlength* (*nnth* (*NCons nell nells*) *i*))
using *NCons.hyps*(*1*) *eSuc-enat nfinite-nlength-enat* **by** *fastforce*
have *6*: ($\sum$ *i* = *1*..(*Suc* (*the-enat* (*nlength* (*nells*)))). *nlength* (*nnth* (*NCons nell nells*) *i*)) =
($\sum$ *i* = *0*..((*the-enat* (*nlength* (*nells*)))). *nlength* (*nnth* (*NCons nell nells*) (*Suc i*)))
using *sum.shift-bounds-cl-nat-ivl*[*of* λ *i*. *nlength* (*nnth* (*NCons nell nells*) *i*) *0 1*
((*the-enat* (*nlength* (*nells*))))]

by *simp*
have *7*: $(\sum i = 0..((\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) (\textit{Suc } i))) =$
$(\sum i = 0..((\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{nlength} (\textit{nnth} (\textit{nells}) (i)))$
by *auto*
show *?thesis*
using *2 3 4 5 6* by *force*
qed
qed

lemma *nlength-nconcat-nfinite-conv-sum-alt*:
assumes *nfinite nells*
shows $\textit{nlength nells} + (\sum i = 0.. (\textit{the-enat} (\textit{nlength nells})). \textit{nlength} (\textit{nnth nells } i)) =$
$\textit{epred}(\sum i = 0.. (\textit{the-enat} (\textit{nlength nells})). \textit{eSuc} (\textit{nlength} (\textit{nnth nells } i)))$
using *assms*
proof(*induct rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** *simp*
next
case (*NCons nell nells*)
then show *?case*
proof −
have *1*: $(\sum i = 0..\textit{the-enat} (\textit{nlength} (\textit{NCons nell nells})). \textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i)) =$
$\textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) 0) +$
$(\sum i = 1..\textit{the-enat} (\textit{nlength} (\textit{NCons nell nells})). \textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i))$
by (*simp add*: *sum.atLeast-Suc-atMost*)
have *2*: $(\sum i = 1..\textit{the-enat} (\textit{nlength} (\textit{NCons nell nells})). \textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i)) =$
$(\sum i = 1..(\textit{Suc} (\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i))$
using *NCons.hyps*(*1*) *eSuc-enat nfinite-nlength-enat* **by** *fastforce*
have *3*: $(\sum i = 1..(\textit{Suc} (\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i)) =$
$(\sum i = 0..((\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) (\textit{Suc } i)))$
using *sum.shift-bounds-cl-nat-ivl*[*of* λ *i*. *nlength* (*nnth* (*NCons nell nells*) *i*) *0 1*
((*the-enat* (*nlength* (*nells*))))]
by *simp*
have *4*: $(\sum i = 0..((\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) (\textit{Suc } i))) =$
$(\sum i = 0..((\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{nlength} (\textit{nnth} (\textit{nells}) (i)))$
by *auto*
have *5*: $(\sum i = 0.. (\textit{the-enat} (\textit{nlength} (\textit{NCons nell nells}))). \textit{eSuc} (\textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i)))$
$=$
$\textit{eSuc} (\textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) 0)) +$
$(\sum i = 1.. (\textit{the-enat} (\textit{nlength} (\textit{NCons nell nells}))). \textit{eSuc} (\textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i)))$
by (*simp add*: *sum.atLeast-Suc-atMost*)
have *6*: $(\sum i = 1.. (\textit{the-enat} (\textit{nlength} (\textit{NCons nell nells}))). \textit{eSuc} (\textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i)))$
$=$
$(\sum i = 1.. (\textit{Suc} (\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{eSuc} (\textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i)))$
using *NCons.hyps*(*1*) *eSuc-enat nfinite-nlength-enat* **by** *fastforce*
have *7*: $(\sum i = 1.. (\textit{Suc} (\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{eSuc} (\textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) i))) =$
$(\sum i = 0.. ((\textit{the-enat} (\textit{nlength} (\textit{nells})))). \textit{eSuc} (\textit{nlength} (\textit{nnth} (\textit{NCons nell nells}) (\textit{Suc } i))))$
using *sum.shift-bounds-cl-nat-ivl*[*of* λ *i*. *eSuc*(*nlength* (*nnth* (*NCons nell nells*) *i*)) *0 1*
((*the-enat* (*nlength* (*nells*))))]
by *simp*

have *8*: (∑ *i* = *0*.. ((*the-enat* (*nlength* (*nells*)))). *eSuc* (*nlength* (*nnth* (*NCons nell nells*) (*Suc i*)))) =
(∑ *i* = *0*.. ((*the-enat* (*nlength* (*nells*)))). *eSuc* (*nlength* (*nnth* (*nells*) (*i*))))
by *auto*
have *9*: (∑ *i* = *0*.. (*the-enat* (*nlength* (*NCons nell nells*))). *eSuc* (*nlength* (*nnth* (*NCons nell nells*) *i*)))
=
(*eSuc*(*nlength nell*) +
(∑ *i* = *0*.. ((*the-enat* (*nlength* (*nells*)))). *eSuc* (*nlength* (*nnth* (*nells*) (*i*)))))
by (*metis 5 6 7 8 nnth-0*)
have *10*: *epred*(∑ *i* = *0*.. (*the-enat* (*nlength* (*NCons nell nells*))). *eSuc* (*nlength* (*nnth* (*NCons nell nells*) *i*))) =
epred(*eSuc*(*nlength nell*) +
(∑ *i* = *0*.. ((*the-enat* (*nlength* (*nells*)))). *eSuc* (*nlength* (*nnth* (*nells*) (*i*)))))
using *9* **by** *presburger*
have *11*: *epred*(*eSuc*(*nlength nell*) +
(∑ *i* = *0*.. ((*the-enat* (*nlength* (*nells*)))). *eSuc* (*nlength* (*nnth* (*nells*) (*i*))))) =
eSuc(*nlength nell*) +
epred(∑ *i* = *0*.. ((*the-enat* (*nlength* (*nells*)))). *eSuc* (*nlength* (*nnth* (*nells*) (*i*))))
by (*metis add.commute eSuc-ne-0 epred-iadd1 le-add1 le-add-same-cancel1 sum.last-plus zero-eq-add-iff-both-eq-0*)
show *?thesis*
using *1 11 2 3 9 NCons.hyps*(*2*) *eSuc-plus* **by** *fastforce*
qed
qed

lemma *nset-nconcat-nfinite*:
assumes ∀ *xs* ∈ *nset nells*. *nfinite xs*
shows *nset* (*nconcat nells*) = (⋃ *xs*∈*nset nells*. *nset xs*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nset* (*nconcat nells*) =
nset (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))
by (*simp add*: *nconcat-def2*)
have *2*: *nset* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)) =
lset (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)))
by (*metis lset-llist-of-nellist-a*)
have *3*: (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
((((*lconcat* ∘ *?n2l*) *nells*)))
using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *4*: *lset* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
lset ((((*lconcat* ∘ *?n2l*) *nells*)))
using *3* **by** *presburger*
have *5*: ∀ *nell* ∈ *lset* (*?n2l nells*) . *lfinite nell*
using *assms nfinite-def* **by** *fastforce*
have *6*: *lset* ((((*lconcat* ∘ *?n2l*) *nells*))) = (⋃ *nell*∈ *lset* (*?n2l nells*). *lset nell*)
by (*simp add*: *5 lset-lconcat-lfinite*)
have *7*: (⋃ *nell*∈ *lset* (*?n2l nells*). *lset nell*) = ⋃ (*nset* ‘ *nset nells*)
by *simp*
show *?thesis*
by (*metis 6 7 lset-llist-of-nellist-a nconcat-def3 o-apply*)

qed

lemma *nconcat-ntake*:
shows *nconcat* (*ntake* (*enat n*) *nells*) =
ntake (*n* +($\sum$ *i*=*0*..*n*. *nlength* (*nnth nells i*))) (*nconcat nells*)
proof(*induct n arbitrary*: *nells*)
case *0* **thus** *?case*
proof (*cases nells*)
case (*NNil nell*)
then show *?thesis* **by** (*simp add*: *zero-enat-def*[*symmetric*] *nnth-NNil ntake-all*)
next
case (*NCons nell nell2*)
then show *?thesis* **by** (*simp add*: *zero-enat-def*[*symmetric*])
(*metis dual-order.refl nlast-NNil nnth-0 ntake-all ntake-nappend1 ntaken-0 ntaken-nlast*)
qed
next
case (*Suc n*)
show *?case*
proof (*cases nells*)
case (*NNil nell*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *dual-order.trans enat-le-plus-same*(*1*) *enat-le-plus-same*(*2*) *nconcat-NNil nfinite-NNil nlast-NNil nlength-NNil nnth-nlast ntake-NNil ntake-all sum.atLeast0-atMost-Suc-shift the-enat-0*)
next
case (*NCons nell nells1*)
then show *?thesis*
proof −
have *1*: *nconcat* (*ntake* (*enat* (*Suc n*)) *nells*) =
nappend nell (*nconcat* (*ntake* (*enat n*) *nells1*))
by (*metis NCons eSuc-enat nconcat-NCons ntake-Suc-NCons*)
have *2*: (*nconcat* (*ntake* (*enat n*) *nells1*)) =
ntake (*enat n* + ($\sum$ *i* = *0*..*n*. *nlength* (*nnth nells1 i*))) (*nconcat nells1*)
using *NCons Suc.hyps Suc.prems Suc-ile-eq* **by** *auto*
have *3*: *nlength* (*nappend nell* (*nconcat nells1*)) =
nlength nell + *1* + *nlength* (*nconcat nells1*)
by *simp*
have *4*: *nappend nell* (*ntake* (*enat n* + ($\sum$ *i* = *0*..*n*. *nlength* (*nnth nells1 i*))) (*nconcat nells1*)) =
ntake (*nlength nell* + *1* + (*enat n* + ($\sum$ *i* = *0*..*n*. *nlength* (*nnth nells1 i*)))) (*nappend nell* (*nconcat nells1*))
proof (*cases nlength nell* = ∞)
case *True*
then show *?thesis* **by** (*metis enat-le-plus-same*(*2*) *ntake-all ntake-nappend1 plus-enat-simps*(*2*))
next
case *False*
then show *?thesis* **by** (*simp add*: *ab-semigroup-add-class.add-ac*(*1*) *enat-0-iff*(*1*) *ntake-nappend2*)
qed
have *5*: ($\sum$ *i* = *0*..*Suc n*. *nlength* (*nnth nells i*)) =
nlength (*nnth nells 0*) + ($\sum$ *i* = *1*..*Suc n*. *nlength* (*nnth nells i*))

by (*simp add*: *sum.atLeast-Suc-atMost*)
have *6*: *nlength* (*nnth nells 0*) = *nlength nell*
by (*simp add*: *NCons*)
have *7*: ($\sum$ *i* = *1..Suc n. nlength* (*nnth nells i*)) =
($\sum$ *i* = *0..n. nlength* (*nnth nells* (*Suc i*)))
using *sum.shift-bounds-cl-nat-ivl*[*of* λ*i . nlength* (*nnth nells i*) *0 1 n*]
by *simp*
have *8*: ($\sum$ *i* = *0..n. nlength* (*nnth nells* (*Suc i*))) =
($\sum$ *i* = *0..n. nlength* (*nnth nells1* (*i*)))
using *NCons* **by** *auto*
have *9*: *nlength nell* + *1* + (*enat n* + ($\sum$ *i* = *0..n. nlength* (*nnth nells1 i*))) =
(*enat* (*Suc n*) + ($\sum$ *i* = *0..Suc n. nlength* (*nnth nells i*)))
by (*metis* (*no-types*, *lifting*) *5 6 7 8 ab-semigroup-add-class.add-ac*(*1*)
add.left-commute eSuc-enat plus-1-eSuc(*2*))
show *?thesis*
by (*metis 1 2 4 9 NCons nconcat-NCons*)
qed
qed
qed

lemma *nnth-nconcat-conv*:
assumes *enat n* $\leq$ *nlength* (*nconcat nells*)
shows $\exists$ *m n'*. *nnth* (*nconcat nells*) *n* = *nnth* (*nnth nells m*) *n'* $\wedge$ *enat n'* $\leq$ *nlength* (*nnth nells m*) $\wedge$
enat m $\leq$ *nlength nells* $\wedge$
enat n = ($\sum$ *i*<*m . eSuc*(*nlength* (*nnth nells i*))) + *enat n'*
proof −
let *?n2l* = (*lmap llist-of-nellist* $\circ$ *llist-of-nellist*)
have *1*: $\bigwedge$*nell. nlength nell* = *epred* (*llength* (*llist-of-nellist nell*))
unfolding *nlength-def* **by** *auto*
have *2*: $\bigwedge$*nell j. j*$\leq$ *nlength nell* $\longrightarrow$ *nnth nell j* = *lnth* (*llist-of-nellist nell*) *j*
unfolding *nnth-def* **by** *auto*
have *3*: *nlength* (*nconcat nells*) =
nlength (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*))
by (*simp add*: *nconcat-def2*)
have *4*: *nlength* (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*)) =
epred (*llength* (*llist-of-nellist* (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*))))
using *nlength.rep-eq* **by** *blast*
have *5*: (*llist-of-nellist* (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*))) =
((((*lconcat* $\circ$ *?n2l*) *nells*)))
using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *6*: *epred* (*llength* (*llist-of-nellist* (((*nellist-of-llist* $\circ$ *lconcat* $\circ$ *?n2l*) *nells*)))) =
epred (*llength* (((*lconcat* $\circ$ *?n2l*) *nells*)))
using *5* **by** *presburger*
have *7*: *enat n* < (*llength* (((*lconcat* $\circ$ *?n2l*) *nells*)))
by (*metis* (*no-types*, *lifting*) *3 4 5 assms co.enat.collapse iless-Suc-eq llength-eq-0*
llist-of-nellist-not-lnull)
have *8*: (((*lconcat* $\circ$ *?n2l*) *nells*)) =
lconcat (*?n2l nells*)
by *simp*
have *9*: $\exists$ *m n'*.

$lnth$ ($lconcat$ ($?n2l$ $nells$)) n = $lnth$ ($lnth$ ($?n2l$ $nells$) m) n' ∧
$enat$ n' < $llength$ ($lnth$ ($?n2l$ $nells$) m) ∧ $enat$ m < $llength$ ($?n2l$ $nells$) ∧
$enat$ n = (∑ i<m. $llength$ ($lnth$ ($?n2l$ $nells$) i)) + $enat$ n'
using *7 lnth-lconcat-conv*[*of n* (*?n2l nells*)]
by *fastforce*
obtain *m n'* **where** *10*: *lnth* (*lconcat* (*?n2l nells*)) *n* = *lnth* (*lnth* (*?n2l nells*) *m*) *n'* ∧
enat n' < *llength* (*lnth* (*?n2l nells*) *m*) ∧ *enat m* < *llength* (*?n2l nells*) ∧
enat n = (∑ *i*<*m*. *llength* (*lnth* (*?n2l nells*) *i*)) + *enat n'*
using *9* **by** *blast*
have *11*: *lnth* (*lconcat* (*?n2l nells*)) *n* = *nnth* (*nconcat nells*) *n*
by (*metis 2 5 8 assms nconcat-def2*)
have *12*: *lnth* (*lnth* (*?n2l nells*) *m*) *n'* = *nnth* (*nnth nells m*) *n'*
by (*metis 10 2 comp-apply co.enat.collapse iless-Suc-eq llength-eq-0 llength-lmap llist-of-nellist-not-lnull lnth-lmap nlength.rep-eq*)
have *13*: *llength* (*lnth* (*?n2l nells*) *m*) = *eSuc* (*nlength* (*nnth nells m*))
by (*metis 10 2 comp-apply co.enat.collapse iless-Suc-eq llength-eq-0 llength-lmap llist-of-nellist-not-lnull lnth-lmap nlength.rep-eq*)
have *14*: *llength* (*?n2l nells*) = *eSuc*(*nlength nells*)
by (*metis comp-apply co.enat.exhaust-sel llength-eq-0 llength-llist-of-nellist llength-lmap llist-of-nellist-not-lnull*)
have *15*: ⋀*i*. *i*<*m* ⟹ *llength* (*lnth* (*?n2l nells*) *i*) = *eSuc*(*nlength*(*nnth nells i*))
proof −
fix *i*
assume *a*: *i*<*m*
show *llength* (*lnth* (*?n2l nells*) *i*) = *eSuc*(*nlength*(*nnth nells i*))
proof −
have *151*: (*lnth* (*?n2l nells*) *i*) = *llist-of-nellist* (*nnth nells i*)
using *a 10 14 2*
by (*metis comp-apply enat-ord-simps*(*2*) *iless-Suc-eq llength-lmap lnth-lmap order-less-trans*)
have *152*: *llength* (*llist-of-nellist* (*nnth nells i*)) = *eSuc*(*nlength*(*nnth nells i*))
using *epred-inject* **by** *force*
show *?thesis* **using** *151 152* **by** *presburger*
qed
qed
have *16*: (∑ *i*<*m*. *llength* (*lnth* (*?n2l nells*) *i*)) = (∑ *i*<*m*. *eSuc*(*nlength*(*nnth nells i*)))
by (*meson 15 lessThan-iff sum.cong*)
show *?thesis*
using *10 11 12 13 14 16* **by** (*metis iless-Suc-eq*)
qed

lemma *nnth-nconcat-ntake*:
assumes *enat w* ≤ *nlength* (*nconcat* (*ntake* (*enat n*) *nells*))
shows *nnth* (*nconcat* (*ntake* (*enat n*) *nells*)) *w* = *nnth* (*nconcat nells*) *w*
using *assms* **by** (*simp add*: *nconcat-ntake ntake-nnth*)

lemma *nfinite-nconcat* [*simp*]:
nfinite (*nconcat nells*) ⟷ *nfinite nells* ∧ (∀ *nell* ∈ *nset nells*. *nfinite nell*)
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs*

```
  thus ?rhs (is ?concl nells)
  proof(induct nconcat nells arbitrary: nells rule: nfinite-induct)
  case (NNil y)
  then show ?case
  by (metis is-NNil-imp-nfinite nconcat-eq-NNil nellist.discI(1) nellist.simps(20) singleton-iff)
  next
  case (NCons x nell)
  then show ?case
    proof (cases nells)
    case (NNil nell1)
    then show ?thesis using NCons.hyps by auto
    next
    case (NCons nell1 nells1)
    then show ?thesis using NCons.hyps by simp
     (metis nconcat-NCons nellist.sel(5) nellist.set-intros(3) nfinite-NCons nfinite-nappend ntl-nappend)
    qed
  qed
next
  assume ?rhs
  then obtain nfinite ( nells)
    and ∀nell∈nset nells. nfinite nell ..
  thus ?lhs
  proof(induct nells rule: nfinite-induct)
  case (NNil nell)
  then show ?case
  by simp
  next
  case (NCons nell nells)
  then show ?case
  by (simp add: nfinite-nappend)
  qed
qed

lemma nfilter-nconcat-nfinite-help:
assumes (∀nell ∈ nset nells. (∃ x ∈ nset nell. P x))
shows  (∃nell ∈ nset (nconcat nells). P nell)
proof (cases nells)
case (NNil nell)
then show ?thesis using assms by simp
next
case (NCons nell nells1)
then show ?thesis using assms
  by simp (metis dual-order.trans enat-le-plus-same(1) in-nset-conv-nnth nlength-nappend nnth-nappend1)
qed

lemma nfilter-nconcat-nfinite:
assumes  ∀nell∈nset nells. nfinite nell
         ∀nell ∈ nset nells. (∃ x ∈ nset nell. P x)
 shows   nfilter P (nconcat nells) = nconcat (nmap (nfilter P) nells)
proof –
```

let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *0*: ∃ *nell* ∈ *nset* (*nconcat nells*). *P nell*
  using *assms nfilter-nconcat-nfinite-help* **by** *blast*
have *1*: *nset nells* = *lset*(*llist-of-nellist nells*)
  by *simp*
have *2*: ⋀*nell*. *nell* ∈ *lset*(*llist-of-nellist nells*)⟶ *lfinite* (*llist-of-nellist nell*)
 using *1 assms nfinite-def* **by** *blast*
have *3*: ⋀*nell Q*. (∃ *x*∈*nset nell*. *Q x*) ⟶
    nfilter Q nell = *nellist-of-llist* (*lfilter Q* (*llist-of-nellist nell*))
    unfolding *nfilter-def* **by** *simp*
have *4*: *nfilter P* (*nconcat nells*) =
    nfilter P (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))
    by (*simp add*: *nconcat-def2*)
have *5*: *nfilter P* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)) =
    nellist-of-llist (*lfilter P* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))))
   by (*metis 3 0 nconcat-def2*)
have *6*: (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
    ((((*lconcat* ∘ *?n2l*) *nells*)))
  using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *7*: (*lfilter P* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)))) =
    (*lfilter P* ((((*lconcat* ∘ *?n2l*) *nells*))))
  using *6* **by** *presburger*
have *8*: (((*lconcat* ∘ *?n2l*) *nells*)) = *lconcat* (*?n2l nells*)
    by *simp*
have *9*: ∀ *nell*∈*lset* (*?n2l nells*). *lfinite nell*
   by (*simp add*: *2*)
have *10*: (*lfilter P* ((((*lconcat* ∘ *?n2l*) *nells*)))) = *lconcat* (*lmap* (*lfilter P*) (*?n2l nells*))
  by (*simp add*: *9 lfilter-lconcat-lfinite*)
have *11*: *nconcat* (*nmap* (*nfilter P*) *nells*) =
    (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap* (*nfilter P*) *nells*)))
    by (*simp add*: *nconcat-def2*)
have *12*: ⋀*nell f*. *nmap f nell* = *nellist-of-llist* (*lmap f* (*llist-of-nellist nell*))
  by (*metis llist-of-nellist-inverse-b llist-of-nellist-not-lnull nmap-nellist-of-llist*)
have *13*: (*nmap* (*nfilter P*) *nells*) = *nellist-of-llist* (*lmap* (*nfilter P*) (*llist-of-nellist nells*))
  using *12* **by** *blast*
have *14*: *nellist-of-llist* (*lmap* (*nfilter P*) (*llist-of-nellist nells*)) =
    nellist-of-llist (*lmap* (λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*))
 by (*metis* (*mono-tags*, *lifting*) *1 3 assms*(*2*) *llist.map-cong*)
have *15*: *lmap* (*lfilter P*) (*lmap llist-of-nellist* (*llist-of-nellist nells*)) =
    lmap ((*lfilter P*)∘ *llist-of-nellist*) (*llist-of-nellist nells*)
  using *llist.map-comp* **by** *blast*
have *16*: ((*?n2l* (*nmap* (*nfilter P*) *nells*))) =
    ((*?n2l* (*nellist-of-llist* (*lmap* (*nfilter P*) (*llist-of-nellist nells*)))))
   using *13* **by** *presburger*
have *17*: ((*?n2l* (*nellist-of-llist* (*lmap* (*nfilter P*) (*llist-of-nellist nells*))))) =
    ((*?n2l* (*nellist-of-llist*
      (*lmap* (λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*)))))
   using *14* **by** *presburger*
have *18*: ((*?n2l* (*nellist-of-llist*
      (*lmap* (λ*ys*. *nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*))))) =

((*lmap llist-of-nellist*
(((*lmap* (*λys. nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*))))))
by *simp*
have *19*: ((*lmap llist-of-nellist*
(((*lmap* (*λys. nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*))) (*llist-of-nellist nells*)))))) =
((*lmap* (*llist-of-nellist*∘(*λys. nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*)))) (*llist-of-nellist nells*)))
using *llist.map-comp* **by** *blast*
have *20*: ⋀ *nell. nell* ∈ *lset*(*llist-of-nellist nells*) ⟶ ¬*lnull* (*lfilter P* (*llist-of-nellist nell*))
by (*simp add*: *assms*(*2*))
have *21*: ((*lmap* (*llist-of-nellist*∘(*λys. nellist-of-llist* (*lfilter P* (*llist-of-nellist ys*)))) (*llist-of-nellist nells*)))
=
((*lmap* ((*λys.* (*lfilter P* (*llist-of-nellist ys*)))) (*llist-of-nellist nells*)))
by (*metis* (*no-types, lifting*) *20 comp-def llist.map-cong0 nellist-of-llist-inverse*)
have *22*: ((*lmap llist-of-nellist* (*llist-of-nellist* (*nmap* (*nfilter P*) *nells*)))) =
(*lmap* (*lfilter P*) (((((*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*))))
using *13 14 15 19 21* **by** *force*
have *23*: *nellist-of-llist* (*lconcat* (*lmap* (*lfilter P*) (*?n2l nells*))) =
nconcat (*nmap* (*nfilter P*) *nells*)
by (*simp add*: *22 nconcat-def2*)
show *?thesis*
by (*metis 10 23 5 6 nconcat-def2*)
qed

lemma *nconcat-nmap-singleton* [*simp*]:
nconcat (*nmap* (*λx. NNil* (*f x*)) *nell*) = *nmap f nell*
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nconcat* (*nmap* (*λx. NNil* (*f x*)) *nell*) =
(((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap* (*λx. NNil* (*f x*)) *nell*)))
by (*simp add*: *nconcat-def2*)
have *2*: (*nmap* (*λx. NNil* (*f x*)) *nell*) =
nellist-of-llist (*lmap* (*λx. NNil* (*f x*)) (*llist-of-nellist nell*))
by (*simp add*: *lmap-llist-of-nellist*)
have *3*: *nellist-of-llist* (*lmap* (*λx. NNil* (*f x*)) (*llist-of-nellist nell*)) =
nellist-of-llist (*lmap* (*λx. nellist-of-llist* (*LCons* (*f x*) *LNil*)) (*llist-of-nellist nell*))
by *force*
have *4*: *nmap f nell* = *nellist-of-llist* (*lmap f* (*llist-of-nellist nell*))
by (*simp add*: *lmap-llist-of-nellist*)
have *5*: *lconcat* (*?n2l* (*nellist-of-llist*
(*lmap* (*λx. nellist-of-llist* (*LCons* (*f x*) *LNil*)) (*llist-of-nellist nell*)))) =
lconcat (*lmap llist-of-nellist* (*lmap* (*λz. NNil* (*f z*)) (*llist-of-nellist nell*)))
by *simp*
have *6*: (*lmap llist-of-nellist* (*lmap* (*λz. NNil* (*f z*)) (*llist-of-nellist nell*))) =
(*lmap* (*llist-of-nellist*∘(*λz. NNil* (*f z*))) (*llist-of-nellist nell*))
using *llist.map-comp* **by** *metis*
have *7*: (*lmap* (*llist-of-nellist*∘(*λz. NNil* (*f z*))) (*llist-of-nellist nell*)) =
(*lmap* (*λz. LCons* (*f z*) *LNil*) (*llist-of-nellist nell*))
by *auto*
have *8*: *lconcat* (*?n2l* (*nellist-of-llist*
(*lmap* (*λx. nellist-of-llist* (*LCons* (*f x*) *LNil*)) (*llist-of-nellist nell*))))

= (*lmap f* (*llist-of-nellist nell*))
  using *6* **by** *force*
have *9*: (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) (*nmap* (*λx. NNil* (*f x*)) *nell*))) =
  nellist-of-llist (*lmap f* (*llist-of-nellist nell*))
  using *2 8* **by** *auto*
show *?thesis*
using *1 4 9* **by** *presburger*
qed

lemma *nset-nconcat-subset*:
nset (*nconcat nells*) ⊆ (⋃*nell*∈*nset nells. nset nell*)
proof –
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nset* (*nconcat nells*) =
  nset (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))
  by (*simp add*: *nconcat-def2*)
have *2*: *nset* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)) =
  lset (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)))
  by (*metis lset-llist-of-nellist-a*)
have *3*: (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
  ((((*lconcat* ∘ *?n2l*) *nells*)))
  using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *4*: *lset* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
  lset ((((*lconcat* ∘ *?n2l*) *nells*)))
  using *3* **by** *presburger*
have *5*: *lset* ((((*lconcat* ∘ *?n2l*) *nells*))) ⊆
  (⋃*nell*∈*lset* (*?n2l nells*). *lset nell*)
  using *lset-lconcat-subset* **by** *fastforce*
have *6*: (⋃*nell*∈*lset* (*?n2l nells*). *lset nell*) =
  ⋃ (*nset* ‘ *nset nells*)
  by *simp*
show *?thesis*
by (*metis 1 2 3 5 6*)
qed

lemma *ndistinct-nconcat*:
assumes *ndistinct nells*
  ⋀*nell. nell* ∈ *nset nells* ⟹ *ndistinct nell*
  ⋀*nell nell1.* ⟦ *nell* ∈ *nset nells*; *nell1* ∈ *nset nells*; *nell* ≠ *nell1* ⟧ ⟹ *nset nell* ∩ *nset nell1* = {}
shows *ndistinct* (*nconcat nells*)
proof –
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *ldistinct* (*llist-of-nellist nells*)
  using *assms*(*1*) *ndistinct.rep-eq* **by** *auto*
have *2*: ⋀*nell. nell* ∈ *lset* (*llist-of-nellist nells*) ⟹ *ldistinct* (*llist-of-nellist nell*)
  using *assms*(*2*) *ndistinct.rep-eq* **by** *auto*
have *3*: ⋀*nell nell1.* ⟦ *nell* ∈ *lset* (*llist-of-nellist nells*); *nell1* ∈ *lset* (*llist-of-nellist nells*); *nell* ≠ *nell1* ⟧
  ⟹ *lset* (*llist-of-nellist nell*) ∩ *lset* (*llist-of-nellist nell1*) = {}
  using *assms*(*3*) **by** *simp*
have *4*: *ndistinct* (*nconcat nells*) =

ndistinct (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))
by (*simp add*: *nconcat-def2*)
have *5*: *ndistinct* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)) =
ldistinct (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*)))
using *ndistinct.rep-eq* **by** *blast*
have *6*: (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
((((*lconcat* ∘ *?n2l*) *nells*)))
using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have *7*: *ldistinct* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lconcat* ∘ *?n2l*) *nells*))) =
ldistinct ((((*lconcat* ∘ *?n2l*) *nells*)))
using *6* **by** *presburger*
have *8*: *ldistinct* (*?n2l nells*)
by (*metis 1 bi-unique-cr-nellist-help comp-eq-dest-lhs inj-on-def ldistinct-lmap*)
have *9*: $\bigwedge$*nell*. *nell* ∈ *lset* (*?n2l nells*) ⟹ *ldistinct nell*
using *2* **by** *auto*
have *10*: $\bigwedge$*nell nell1*. ⟦ *nell* ∈ *lset* (*?n2l nells*);
nell1 ∈ *lset* (*?n2l nells*); *nell* ≠ *nell1*⟧ ⟹
lset nell ∩ *lset nell1* = {}
by (*metis* (*no-types*, *lifting*) *assms*(*3*) *comp-def imageE lset-llist-of-nellist-a lset-lmap*)
have *11*: *ldistinct* (*lconcat* (*?n2l nells*))
using *10 8 9 ldistinct-lconcat* **by** *blast*
show *?thesis*
using *11 4 5 6* **by** *fastforce*
qed

2.14 *nellist-all2*

lemmas *nellist-all2-NNil* = *nellist.rel-inject*(*1*)
lemmas *nellist-all2-NCons* = *nellist.rel-inject*(*2*)

lemma *nellist-all2-NNil1*:
nellist-all2 Q (*NNil b*) *nell* ⟷ (∃ *b'*. *nell* = *NNil b'* ∧ *Q b b'*)
using *nellist.rel-cases* **by** *fastforce*

lemma *nellist-all2-NNil2*:
nellist-all2 Q nell (*NNil b'*) ⟷ (∃ *b*. *nell* = *NNil b* ∧ *Q b b'*)
using *nellist.rel-sel*
by (*metis is-NNil-def nellist-all2-NNil*)

lemma *nellist-all2-NCons1*:
nellist-all2 P (*NCons x nell*) *nell'* ⟷
(∃ *x' nell''*. *nell'* = *NCons x' nell''* ∧ *P x x'* ∧ *nellist-all2 P nell nell''*)
using *nellist.rel-sel*
by (*metis nellist.collapse*(*2*) *nellist.disc*(*2*) *nellist.sel*(*3*) *nellist.sel*(*5*))

lemma *nellist-all2-NCons2*:
nellist-all2 P nell' (*NCons x nell*) ⟷
(∃ *x' nell''*. *nell'* = *NCons x' nell''* ∧ *P x' x* ∧ *nellist-all2 P nell'' nell*)
by (*metis nellist.collapse*(*2*) *nellist.disc*(*2*) *nellist.rel-sel nellist.sel*(*3*) *nellist.sel*(*5*))

lemma *nellist-all2-coinduct* [*consumes 1*, *case-names ilist-all2*,
case-conclusion nellist-all2 is-NNil NNil NCons,
coinduct pred: *nellist-all2*]:
assumes *X nellx nelly*
and $\bigwedge$*nellix nelliy.*
X nellix nelliy $\Longrightarrow$
(*is-NNil nellix* = *is-NNil nelliy*) $\wedge$
(*is-NNil nellix* $\longrightarrow$ *is-NNil nelliy* $\longrightarrow$ *P* (*nlast nellix*) (*nlast nelliy*)) $\wedge$
($\neg$ *is-NNil nellix* $\longrightarrow$ $\neg$ *is-NNil nelliy* $\longrightarrow$ *P* (*nhd nellix*) (*nhd nelliy*) $\wedge$
(*X* (*ntl nellix*) (*ntl nelliy*) $\vee$ *nellist-all2 P* (*ntl nellix*) (*ntl nelliy*)))
shows *nellist-all2 P nellx nelly*
using *assms*
nellist.rel-coinduct[*of* (λ *nelx nely. X nelx nely* $\vee$ *nellist-all2 P nelx nely*) *nellx nelly P*]
by (*metis nellist.rel-sel*)

lemma *nellist-all2-cases*[*consumes 1*, *case-names NNil NCons*, *cases pred*]:
assumes *nellist-all2 P nellx nelly*
obtains (*NNil*) *b b'* **where** *nellx* = *NNil b nelly* = *NNil b' P b b'*
| (*NCons*) *x nellx' y nelly'*
where *nellx* = *NCons x nellx'* **and** *nelly* = *NCons y nelly'*
and *P x y* **and** *nellist-all2 P nellx' nelly'*
using *assms*
using *nellist.rel-cases* **by** *blast*

lemma *nellist-all2-nmap*:
nellist-all2 P (*nmap f nellx*) *nelly* $\longleftrightarrow$ *nellist-all2* (λx *y. P* (*f x*) *y*) *nellx nelly*
using *nellist.rel-map*(*1*) **by** *blast*

lemma *nellist-all2-nmap2*:
nellist-all2 P nellx (*nmap f nelly*) $\longleftrightarrow$ *nellist-all2* (λx *y. P x* (*f y*)) *nellx nelly*
using *nellist.rel-map*(*2*) **by** *blast*

lemma *nellist-all2-mono*:
$\llbracket$ *nellist-all2 P nellx nelly*; $\bigwedge$*x y. P x y* $\Longrightarrow$ *P' x y* $\rrbracket$
$\Longrightarrow$ *nellist-all2 P' nellx nelly*
using *nellist.rel-mono-strong* **by** *blast*

lemma *nellist-all2-nlengthD*:
nellist-all2 P nellx nelly $\Longrightarrow$ *nlength nellx* = *nlength nelly*
by(*transfer*)(*auto dest*: *llist-all2-llengthD*)

lemma *nellist-all2-nfiniteD*:
nellist-all2 P nellx nelly $\Longrightarrow$ *nfinite nellx* = *nfinite nelly*
by *transfer*
(*auto dest*: *llist-all2-lfiniteD*)

lemma *nellist-all2-nfinite1-nlastD*:
$\llbracket$ *nellist-all2 P nellx nelly*; *nfinite nellx* $\rrbracket$ $\Longrightarrow$ *P* (*nlast nellx*) (*nlast nelly*)
by (*frule nellist-all2-nfiniteD*)
(*transfer*,

auto simp add: *llist-all2-conv-all-lnth*,
metis Suc-ile-eq eSuc-enat lfinite.simps lfinite-llength-enat llast-conv-lnth llength-LCons llist.discI(*1*) *order-refl*)

lemma *nellist-all2-nfinite2-nlastD*:
$[\![$ *nellist-all2 P nellx nelly*; *nfinite nelly* $]\!] \Longrightarrow$ *P* (*nlast nellx*) (*nlast nelly*)
by(*metis nellist-all2-nfinite1-nlastD nellist-all2-nfiniteD*)

lemma *nellist-all2D-llist-all2-llist-of-nellist*:
nellist-all2 P nellx nelly $\Longrightarrow$ *llist-all2 P* (*llist-of-nellist nellx*) (*llist-of-nellist nelly*)
by *transfer*
(*simp add*: *nellist-all2-help-b*)

lemma *nellist-all2-is-NNilD*:
nellist-all2 P nellx nelly $\Longrightarrow$ *is-NNil nellx* $\longleftrightarrow$ *is-NNil nelly*
by (*cases nellx*) (*auto simp add*: *nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-nhdD*:
$[\![$ *nellist-all2 P nellx nelly*; $\neg$ *is-NNil nellx* $\vee$ $\neg$ *is-NNil nelly* $]\!] \Longrightarrow$ *P* (*nhd nellx*) (*nhd nelly*)
by (*cases nellx*) (*auto simp add*: *nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-ntlI*:
$[\![$ *nellist-all2 P nellx nelly*; $\neg$ *is-NNil nellx* $\vee$ $\neg$ *is-NNil nelly* $]\!] \Longrightarrow$
nellist-all2 P (*ntl nellx*) (*ntl nelly*)
by (*cases nellx*) (*auto simp add*: *nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-refl*:
nellist-all2 P nell nell $\longleftrightarrow$
($\forall$ *x* $\in$ *nset nell*. *P x x*) $\wedge$ (*nfinite nell* $\longrightarrow$ *P* (*nlast nell*) (*nlast nell*))
by *transfer*
(*auto*, *metis in-lset-lappend-iff lappend-lbutlast-llast-id-lfinite lfinite-lappend llist.set-intros*(*1*))

lemma *nellist-all2-reflI*:
$[\![$ $\bigwedge$*x*. *x* $\in$ *nset nell* $\Longrightarrow$ *P x x*; *nfinite nell* $\Longrightarrow$ *P* (*nlast nell*) (*nlast nell*) $]\!]$
$\Longrightarrow$ *nellist-all2 P nell nell*
by(*simp add*: *nellist-all2-refl*)

lemma *nellist-all2-conv-all-nnth-help1*:
$\neg$ *lnull nellx* $\Longrightarrow$ $\neg$ *lnull nelly* $\Longrightarrow$ *lfinite nelly* $\Longrightarrow$ *llist-all2 P nellx nelly* $\Longrightarrow$
P (*llast nellx*) (*llast nelly*)
proof −
assume *a1*: *lfinite nelly*
assume *a2*: *llist-all2 P nellx nelly*
assume *a3*: $\neg$ *lnull nellx*
assume *a4*: $\neg$ *lnull nelly*
have *f5*: *lfinite* (*lappend* (*lbutlast nellx*) (*LCons* (*llast nellx*) *LNil*))
using *a3 a2 a1* **by** (*simp add*: *llist-all2-lfiniteD*)
have *f6*: *llength* (*ltl nellx*) = *epred* (*llength nelly*)
using *a2* **by** (*metis* (*no-types*) *epred-llength llist-all2-llengthD*)

have *f7*: *lappend* (*lbutlast nelly*) (*LCons* (*llast nelly*) *LNil*) = *nelly*
using *a4* **by** (*meson lappend-lbutlast-llast-id*)
have *llength* (*lbutlast nellx*) = *llength* (*ltl nellx*)
using *epred-llength* **by** *auto*
then show *?thesis*
using *f7 f6 f5 a3 a2 a1*
by (*metis* (*no-types*) *lappend-eq-lappend-conv lappend-lbutlast-llast-id lappend-ltake-ldrop lbutlast-conv-ltake lfinite-lappend lhd-LCons llist.disc*(*2*) *llist-all2-lappend1D*(*2*) *llist-all2-lhdD2*)
qed

lemma *nellist-all2-conv-all-nnth*:
nellist-all2 P nellx nelly $\longleftrightarrow$
nlength nellx = *nlength nelly* $\wedge$
($\forall$ *n*. *enat n* $\leq$ *nlength nellx* $\longrightarrow$ *P* (*nnth nellx n*) (*nnth nelly n*))
by *transfer*
(*auto simp add*: *llist-all2-llengthD*,
metis eSuc-epred iless-Suc-eq llength-eq-0 llist-all2-lnthD2,
metis eSuc-epred iless-Suc-eq llength-eq-0 llist-all2-conv-all-lnth)

lemma *nellist-all2-True* [*simp*]:
nellist-all2 (λ- -. *True*) *nellx nelly* $\longleftrightarrow$ *nlength nellx* = *nlength nelly*
by(*simp add*: *nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nnthD*:
$[\![$ *nellist-all2 P nellx nelly*; *enat n* $\leq$ *nlength nellx* $]\!] \Longrightarrow$ *P* (*nnth nellx n*) (*nnth nelly n*)
by (*simp add*: *nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nnthD2*:
$[\![$ *nellist-all2 P nellx nelly*; *enat n* $\leq$ *nlength nelly* $]\!] \Longrightarrow$ *P* (*nnth nellx n*) (*nnth nelly n*)
by (*simp add*: *nellist-all2-conv-all-nnth*)

lemmas *nellist-all2-eq* = *nellist.rel-eq*

lemma *nmap-eq-nmap-conv-nellist-all2*:
nmap f nellx = *nmap f′ nelly* $\longleftrightarrow$
nellist-all2 (λ*x y*. *f x* = *f′ y*) *nellx nelly*
by *transfer*
(*clarsimp simp add*: *lmap-eq-lmap-conv-llist-all2*)

lemma *nellist-all2-trans*:
$[\![$ *nellist-all2 P nellx nelly*; *nellist-all2 P nelly nellz*; *transp P* $]\!]$
$\Longrightarrow$ *nellist-all2 P nellx nellz*
by *transfer* (*auto elim*: *llist-all2-trans dest*: *llist-all2-lfiniteD transpD*)

lemma *nellist-all2-nappendI*:
$[\![$ *nellist-all2 P nellx nelly*;
$[\![$ *nfinite nellx*; *nfinite nelly*; *P* (*nlast nellx*) (*nlast nelly*) $]\!]$
$\Longrightarrow$ *nellist-all2 P nellx′ nelly′* $]\!]$

⟹ *nellist-all2 P* (*nappend nellx nellx′*) (*nappend nelly nelly′*)
by *transfer*
(*auto simp add*: *lnull-def lappend-eq-lappend-conv nellist-all2-conv-all-nnth-help1*
intro: *llist-all2-lappendI*)

lemma *nested-nellist-all2-nested-llist-all2*:
nellist-all2 (*nellist-all2 A*) *nells nells1* =
llist-all2 (*llist-all2 A*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))
(*lmap llist-of-nellist* (*llist-of-nellist nells1*))
proof −
have *1*: *nellist-all2* (*nellist-all2 A*) *nells nells1* =
llist-all2 (*nellist-all2 A*) (*llist-of-nellist nells*) (*llist-of-nellist nells1*)
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*
have *2*: (λ *xx yy*. *nellist-all2 A xx yy*) =
(λ *xx yy*. *llist-all2 A* (*llist-of-nellist xx*) (*llist-of-nellist yy*))
by (*meson nellist-all2D-llist-all2-llist-of-nellist nellist-all2-help-a*)
have *3*: *llist-all2* (*nellist-all2 A*) (*llist-of-nellist nells*) (*llist-of-nellist nells1*) =
llist-all2 (λ *xx yy*. *llist-all2 A* (*llist-of-nellist xx*) (*llist-of-nellist yy*))
(*llist-of-nellist nells*) (*llist-of-nellist nells1*)
using *2* **by** *auto*
have *6*: *llist-all2* (λ *xx yy*. *llist-all2 A* (*llist-of-nellist xx*) (*llist-of-nellist yy*))
(*llist-of-nellist nells*) (*llist-of-nellist nells1*) =
llist-all2 (*llist-all2 A*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))
(*lmap llist-of-nellist* (*llist-of-nellist nells1*))
using *llist-all2-lmap1*[*of* (*llist-all2 A*) *llist-of-nellist* (*llist-of-nellist nells*)
(*lmap llist-of-nellist* (*llist-of-nellist nells1*))]
llist-all2-lmap2[*of* (*llist-all2 A*) - *llist-of-nellist* (*llist-of-nellist nells1*)]
by (*simp add*: *llist-all2-conv-all-lnth*)
show *?thesis*
using *1 3 6* **by** *blast*
qed

lemma *nellist-all2-nconcatI*:
assumes *nellist-all2* (*nellist-all2 A*) *nells nells1*
shows *nellist-all2 A* (*nconcat nells*) (*nconcat nells1*)
proof −
have *3*: *nellist-all2 A* (*nconcat nells*) (*nconcat nells1*) =
llist-all2 A (*llist-of-nellist* (*nconcat nells*)) (*llist-of-nellist* (*nconcat nells1*))
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*
have *4*: (*llist-of-nellist* (*nconcat nells*)) =
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
using *nconcat-def3* **by** *auto*
have *5*: (*llist-of-nellist* (*nconcat nells1*)) =
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells1*)
using *nconcat-def3* **by** *auto*
have *7*: *llist-all2* (*llist-all2 A*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))
(*lmap llist-of-nellist* (*llist-of-nellist nells1*)) ⟹
llist-all2 A ((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells1*)
using *llist-all2-lconcatI*[*of A* (*lmap llist-of-nellist* (*llist-of-nellist nells*))

(*lmap llist-of-nellist* (*llist-of-nellist nells1*))]
by *simp*
have *8*:
llist-all2 A ((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells*)
((*lconcat* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)) *nells1*)
= *nellist-all2 A* (*nconcat nells*) (*nconcat nells1*)
using *3 4 5* **by** *presburger*
have *9*: *nellist-all2* (*nellist-all2 A*) *nells nells1* =
llist-all2 (*llist-all2 A*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))
(*lmap llist-of-nellist* (*llist-of-nellist nells1*))
using *nested-nellist-all2-nested-llist-all2* **by** *blast*
show *?thesis* **using** *assms 7 8 9*
by *blast*
qed

lemma *nlength-nconcat-eqI*:
fixes *nells* :: *'a nellist nellist* **and** *nells1* :: *'b nellist nellist*
assumes *nellist-all2* (*λxs ys. nlength xs* = *nlength ys*) *nells nells1*
shows *nlength* (*nconcat nells*) = *nlength* (*nconcat nells1*)
proof –
have *nellist-all2* (*nellist-all2* (*λa b. True*)) *nells nells1*
using *assms nellist.rel-mono-strong nellist-all2-True* **by** *blast*
then show *?thesis* **using** *nellist-all2-nconcatI nellist-all2-nlengthD* **by** *blast*
qed

lemma *llist-all2-nellist-of-llistI*:
nellist-all2 A nellx nelly ⟹
llist-all2 A (*lbutlast*(*llist-of-nellist nellx*)) (*lbutlast*(*llist-of-nellist nelly*))
proof (*coinduction arbitrary*: *nellx nelly*)
case *LNil*
then show *?case*
by (*metis lbutlast.disc-iff*(*1*) *llist.disc*(*1*) *llist-of-nellist-inverse-a ltl-llist-of-nellist1*
nellist-all2-is-NNilD nellist-of-llist-a.disc(*1*))
next
case (*LCons nell1 nell2*)
then show *?case*
proof –
assume *a0*: *nellist-all2 A nell1 nell2*
assume *a1*: ¬ *lnull* (*lbutlast* (*llist-of-nellist nell1*))
assume *a2*: ¬ *lnull* (*lbutlast* (*llist-of-nellist nell2*))
have *1*: *A* (*lhd* (*lbutlast* (*llist-of-nellist nell1*))) (*lhd* (*lbutlast* (*llist-of-nellist nell2*)))
using *a0 a1 a2*
by (*metis lbutlast.disc*(*1*) *llist.disc*(*1*) *llist-of-nellist-inverse-a ltl-llist-of-nellist1*
nellist-all2-nhdD nhd-nellist-of-llist-a)
have *2*: ((∃ *nellx nelly*.
ltl (*lbutlast* (*llist-of-nellist nell1*)) = *lbutlast* (*llist-of-nellist nellx*) ∧
ltl (*lbutlast* (*llist-of-nellist nell2*)) = *lbutlast* (*llist-of-nellist nelly*) ∧
nellist-all2 A nellx nelly) ∨

llist-all2 A (*ltl* (*lbutlast* (*llist-of-nellist nell1*))) (*ltl* (*lbutlast* (*llist-of-nellist nell2*))))
by (*metis a0 a1 lbutlast.ctr*(*1*) *lbutlast-ltl llist.discI*(*1*) *ltl-llist-of-nellist ltl-llist-of-nellist1 nellist.rel-sel*)
show *?thesis* **using** *1 2* **by** *blast*
qed
qed

lemma *nellist-all2-nellist-of-llist-a* [*simp*]:
nellist-all2 A (*nellist-of-llist-a b llx*) (*nellist-of-llist-a c lly*) ⟷
llist-all2 A llx lly ∧ (*lfinite llx* ⟶ *A b c*)
proof (*cases lfinite llx*)
case *True*
then show *?thesis*
using *llist-all2-nellist-of-llistI*
by (*auto simp add*: *llist-all2-lappendI nellist-all2-help-a*, *fastforce*,
metis lbutlast-lfinite lbutlast-snoc llist-all2-lfiniteD nellist-all2-nfinite1-nlastD nellist-of-llist-a-inverse nfinite-def nlast-nellist-of-llist-a-lfinite)
next
case *False*
then show *?thesis*
by (*metis lbutlast-snoc llist-all2-lappendI llist-all2-nellist-of-llistI nellist-all2-help-a nellist-of-llist-a-inverse*)
qed

2.15 From a nonempty lazy list to a lazy list *llist-of-nellist*

lemma *llist-of-nellist-nmap* [*simp*]:
llist-of-nellist (*nmap f nell*) = *lmap f* (*llist-of-nellist nell*)
by (*simp add*: *lmap-llist-of-nellist*)

lemma *llist-of-nellist-nappend*:
llist-of-nellist (*nappend nellx nelly*) = *lappend* (*llist-of-nellist nellx*) (*llist-of-nellist nelly*)
by (*transfer*) *auto*

lemma *llist-of-nellist-lappendn* [*simp*]:
llist-of-nellist (*lappendn ll nell*) = *lappend ll* (*llist-of-nellist nell*)
by *transfer auto*

lemma *llist-of-nellist-nconcat* [*simp*]:
llist-of-nellist (*nconcat nell*) = *lconcat* ((*lmap llist-of-nellist* ∘ *llist-of-nellist*) *nell*)
using *nconcat-def3* **by** *fastforce*

lemma *llist-of-nellist-nfilter* [*simp*]:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows *llist-of-nellist* (*nfilter P nell*) = *lfilter P* (*llist-of-nellist nell*)
using *assms*
by *transfer auto*

2.16 *ndropn*

lemma *ndropn-0* [*simp*, *code*, *nitpick-simp*]:
ndropn 0 nell = nell
using *zero-enat-def* **by** *transfer auto*

lemma *ndropn-NNil* [*simp*, *code*]:
ndropn n (*NNil b*) = (*NNil b*)
by *transfer auto*

lemma *ndropn-Suc-NCons* [*simp*, *code*]:
ndropn (*Suc n*) (*NCons x nell*) = *ndropn n nell*
proof (*cases nfinite nell*)
case *True*
then show *?thesis*
by *transfer*
(*auto simp add*: *min-def not-lnull-conv Suc-ile-eq llength-eq-infty-conv-lfinite the-enat-eSuc* ,
metis Extended-Nat.eSuc-mono eSuc-enat iless-Suc-eq leD,
metis antisym eSuc-enat enat-the-enat ile-eSuc llength-eq-infty-conv-lfinite n-not-Suc-n
the-enat.simps)
next
case *False*
then show *?thesis*
by *transfer*
(*simp*,
metis co.enat.sel(*2*) *eSuc-infinity infinity-ileE ldropn-Suc-LCons llength-eq-infty-conv-lfinite*
min.cobounded1 min-def the-enat.simps)
qed

lemma *ndropn-Suc* [*nitpick-simp*]:
ndropn (*Suc n*) *nell* = (*case nell of NNil b* $\Rightarrow$ *NNil b* | *NCons x nell'* $\Rightarrow$ *ndropn n nell'*)
by(*cases nell*) *simp-all*

lemma *ltl-power-NNil-help*:
((λ*ll. if* $\exists$ *b. ll* = *LCons b LNil then ll else ltl ll*) $\frown\frown$ *n*) (*LCons b LNil*) = *LCons b LNil*
by (*induction n*) *simp-all*

lemma *ntl-power-NNil*:
(*ntl* $\frown\frown$ *n*) (*NNil b*) = (*NNil b*)
by *transfer* (*auto simp add*: *lnull-def ltl-power-NNil-help*)

lemma *ntl-power-NCons*:
ntl ((*ntl* $\frown\frown$ *n*) (*NCons x nell*)) = (*ntl* $\frown\frown$ *n*) *nell*
by (*induction n*) (*transfer*, *auto*)

lemma *ntl-power-Suc* [*simp*]:
(*ntl* $\frown\frown$ (*Suc n*)) *nell* = (*case nell of NNil b* $\Rightarrow$ *NNil b* | *NCons x nell'* $\Rightarrow$ (*ntl* $\frown\frown$ *n*) *nell'*)
by (*cases nell*)
(*simp-all add*: *ntl-power-NNil ntl-power-NCons*)

lemma *llist-of-nellist-ndropn* [*simp*]:

llist-of-nellist (*ndropn n nell*) =
ldropn (*the-enat* (*min* (*enat n*) ((*epred*(*llength*((*llist-of-nellist nell*))))))))
(*llist-of-nellist nell*)
by *transfer auto*

lemma *ndropn-Suc-conv-ndropn*:
enat n < *nlength nell* ⟹ *NCons* (*nnth nell n*) (*ndropn* (*Suc n*) *nell*) = *ndropn n nell*
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **using** *0.prems* **by** *auto*
next
case (*NCons x nell1*)
then show *?thesis* **by** *simp*
qed
next
case (*Suc n*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **using** *Suc.prems* **by** *auto*
next
case (*NCons x nell1*)
then show *?thesis* **using** *Suc*
by (*metis One-nat-def add.commute add-left-mono gen-nlength-code*(*2*) *gen-nlength-def leD ndropn-Suc-NCons nlength-code nnth-Suc-NCons not-le-imp-less of-nat-Suc of-nat-eq-enat one-enat-def plus-1-eq-Suc*)
qed
qed

lemma *ndropn-nlength* [*simp*]:
nlength (*ndropn n nell*) = *nlength nell* − *enat n*
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x nell1*)
then show *?thesis* **using** *Suc*
by (*metis eSuc-enat eSuc-minus-eSuc ndropn-Suc-NCons nlength-NCons*)
qed
qed

lemma *ndropn-nnth* [*simp*]:
  nnth (*ndropn n nell*) *m* = *nnth nell* (*n*+*m*)
proof (*induct n arbitrary*: *m nell*)
 case *0*
 then show *?case* **by** *simp*
 next
 case (*Suc n*)
 then show *?case*
 proof (*cases nell*)
  case (*NNil x1*)
  then show *?thesis* **by** (*simp add*: *nnth-NNil*)
  next
  case (*NCons x nell1*)
  then show *?thesis* **by** (*simp add*: *Suc.hyps*)
 qed
qed

lemma *ndropn-nnth-a*:
assumes *nlength nell* ≤ *enat*(*n*+*m*)
shows *nnth* (*ndropn n nell*) *m* = (*nlast nell*)
 proof −
 have *1*: *nfinite nell*
 using *assms enat-ile nfinite-conv-nlength-enat* **by** *auto*
 have *2*: *nfinite nell* ⟹ *nlength nell* ≤ *enat*(*n*+*m*) ⟹ *nnth* (*ndropn n nell*) *m* = (*nlast nell*)
 proof (*induct arbitrary*: *n m rule*: *nfinite-induct*)
 case (*NNil y*)
 then show *?case* **by** (*simp add*: *nnth-NNil*)
 next
 case (*NCons x nell*)
 then show *?case*
 proof (*cases n*)
 case *0*
 then show *?thesis*
 by (*metis NCons*(*2*) *NCons*(*3*) *Suc-ile-eq Suc-pred add-cancel-right-left enat-le-plus-same*(*1*) *gen-nlength-def iless-Suc-eq leD le-add1 ndropn-0 nellist.sel*(*2*) *nlast0-nlast nlength-NCons nlength-code nnth-Suc-NCons not-le-imp-less order.not-eq-order-implies-strict*)
 next
 case (*Suc nat*)
 then show *?thesis*
 by (*metis NCons*(*2*) *NCons*(*3*) *add-Suc eSuc-enat epred-eSuc epred-le-epredI ndropn-Suc-NCons nlast-NCons nlength-NCons*)
 qed
 qed
 show *?thesis* **using** *1 2 assms* **by** *auto*
 qed

lemma *ndropn-ntl* :
 ndropn n nell = (*ntl* ^^ *n*) *nell*
proof (*induction n arbitrary*: *nell*)
case *0*

then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
  proof (*cases nell*)
  case (*NNil x1*)
  then show *?thesis*
    by (*simp add*: *ntl-power-NNil*)
  next
  case (*NCons x nell*)
  then show *?thesis*
  by (*metis Suc.IH funpow-Suc-right ndropn-Suc-NCons nellist.sel*(*5*) *o-apply*)
  qed
qed

lemma *ndropn-is-NNil*:
  is-NNil nell $\Longrightarrow$ *ndropn n nell* = *nell*
proof (*induct n arbitrary*: *nell*)
 case *0*
 then show *?case* **by** *auto*
 next
 case (*Suc n*)
 then show *?case* **by** (*simp add*: *ndropn-Suc nellist.case-eq-if*)
qed

lemma *is-NNil-ndropn*:
  is-NNil(*ndropn n nell*) $\longleftrightarrow$ *nlength nell* $\leq$ (*enat n*)
proof (*induct n arbitrary*: *nell*)
 case *0*
 then show *?case*
  proof (*cases nell*)
  case (*NNil x1*)
  then show *?thesis* **by** *simp*
  next
  case (*NCons x nell*)
  then show *?thesis* **using** *zero-enat-def* **by** *auto*
  qed
 next
 case (*Suc n*)
 then show *?case*
  proof (*cases nell*)
  case (*NNil x1*)
  then show *?thesis* **by** *simp*
  next
  case (*NCons x nell*)
  then show *?thesis*
  by (*metis One-nat-def Suc.hyps add.commute add-left-mono co.enat.sel*(*2*) *eSuc-enat epred-le-epredI gen-nlength-code*(*2*) *gen-nlength-def ndropn-Suc-NCons nlength-NCons nlength-code of-nat-Suc of-nat-eq-enat one-enat-def plus-1-eq-Suc*)
  qed

qed

lemma *ndropn-eq-NNil*:
ndropn n nell = (*NNil b*) ⟷ *nnth nell* (*the-enat*(*nlength nell*)) = *b* ∧ *nlength nell* ≤ (*enat n*)
proof –
have *1*: *nfinite nell* ⟹ *ndropn n nell* = *NNil b* ⟹ *nnth nell* (*the-enat* (*nlength nell*)) = *b*
by (*metis add-cancel-left-right enat-le-plus-same*(*2*) *gen-nlength-def is-NNil-ndropn ndropn-NNil ndropn-nnth ndropn-nnth-a nfinite-nlength-enat nlast-NNil nlength-NNil nlength-code plus-enat-simps*(*1*) *the-enat.simps zero-enat-def*)
have *2*: *nfinite nell* ⟹ *ndropn n nell* = *NNil b* ⟹ *nlength nell* ≤ *enat n*
by (*metis is-NNil-ndropn nellist.disc*(*1*))
have *3*: *nfinite nell* ⟹ *nlength nell* ≤ *enat n* ⟹ *b* = *nnth nell* (*the-enat* (*nlength nell*)) ⟹
ndropn n nell = *NNil* (*nnth nell* (*the-enat* (*nlength nell*)))
by (*metis add.right-neutral is-NNil-ndropn ndropn-nnth-a nellist.collapse*(*1*) *nfinite-NNil nlength-NNil nnth-nlast the-enat-0*)
have *4*: ¬*nfinite nell* ⟹
ndropn n nell = (*NNil b*) ⟷
nnth nell (*the-enat*(*nlength nell*)) = *b* ∧ *nlength nell* ≤ (*enat n*)
by (*metis enat-ile is-NNil-ndropn nellist.disc*(*1*) *nfinite-conv-nlength-enat*)
show *?thesis*
using *1 2 3 4* **by** *fastforce*
qed

lemma *ntl-ndropn*:
ntl(*ndropn n nell*) = *ndropn n* (*ntl nell*)
by (*simp add*: *funpow-swap1 ndropn-ntl*)

lemma *nfinite-ndropn-a*:
assumes *nfinite nell*
shows *nfinite*(*ndropn n nell*)
using *assms*
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **by** (*simp add*: *ndropn-ntl*)
qed

lemma *nfinite-ndropn-b*:
assumes *nfinite*(*ndropn n nell*)
shows *nfinite nell*
using *assms*
proof (*induct ys*≡*ndropn n nell arbitrary*: *n nell rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*metis enat-ile ndropn-eq-NNil nfinite-conv-nlength-enat*)
next
case (*NCons x nell*)
then show *?case* **by** (*metis nellist.sel*(*5*) *nfinite-ntl ntl-ndropn*)
qed

lemma *nfinite-ndropn*[*simp*]:
nfinite(*ndropn n nell*) = *nfinite nell*
using *nfinite-ndropn-a nfinite-ndropn-b* **by** *blast*

lemma *ndropn-ndropn*:
ndropn m (*ndropn n nell*) = *ndropn* (*n*+*m*) *nell*
proof (*induct n arbitrary*: *nell*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
by (*metis add-Suc ndropn-NNil ndropn-Suc nellist.case-eq-if*)
qed

lemma *ndropn-nlast*:
nfinite nell $\Longrightarrow$ *ndropn* (*the-enat*(*nlength nell*)) *nell* = (*NNil* (*nlast nell*))
by (*metis add.left-neutral enat.simps*(*3*) *enat-the-enat ndropn-eq-NNil ndropn-nnth ndropn-nnth-a nfinite-conv-nlength-enat order-refl*)

lemma *ndropn-nfirst*:
nfirst (*ndropn n nell*) = (*nnth nell n*)
by *transfer*
(*metis lhd-ldropn llist-of-nellist-ndropn lnull-ldropn not-le-imp-less not-lnull-conv-llist-of-nellist*)

lemma *ndropn-all*:
nlength nell $\leq$ *enat n* $\Longrightarrow$ *ndropn n nell* = (*NNil* (*nlast nell*))
by (*metis enat-ile ndropn-eq-NNil ndropn-nlast nlength-eq-enat-nfiniteD*)

lemma *ndropn-nappend1*:
nfinite nellx $\Longrightarrow$ *n* $\leq$ *nlength nelly* $\Longrightarrow$
ndropn (*Suc*(*the-enat* (*nlength nellx*) + *n*)) (*nappend nellx nelly*) = *ndropn n nelly*
proof (*induct arbitrary*: *nelly n rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** *simp*
next
case (*NCons x nell*)
then show *?case*
by (*metis ab-semigroup-add-class.add-ac*(*1*) *eSuc-enat nappend-NCons ndropn-Suc-NCons nfinite-nlength-enat nlength-NCons plus-1-eq-Suc the-enat.simps*)
qed

lemma *ndropn-nappend2*:
enat n $\leq$ (*nlength nellx*) $\Longrightarrow$ *ndropn n* (*nappend nellx nelly*) = *nappend* (*ndropn n nellx*) *nelly*
proof (*induct n arbitrary*: *nellx nelly*)
case *0*
then show *?case* **by** *simp*
next

case (*Suc n*)
then show *?case*
  proof (*cases nellx*)
  case (*NNil x1*)
  then show *?thesis*
  using *Suc.prems enat-0-iff*(*1*) **by** *auto*
  next
  case (*NCons x21 x22*)
  then show *?thesis*
 by (*metis Suc.hyps Suc.prems eSuc-enat eSuc-ile-mono nappend-NCons ndropn-Suc-NCons nlength-NCons*)
  qed
qed

lemma *ndropn-nappend3*:
 nlength nellx $<$ *enat n* $\Longrightarrow$
  ndropn n (*nappend nellx nelly*) $=$ *ndropn* (*n* $-$ (*the-enat* (*eSuc*(*nlength nellx*)))) *nelly*
proof (*induct n arbitrary*: *nellx nelly*)
case *0*
then show *?case* **using** *zero-enat-def* **by** *auto*
next
case (*Suc n*)
then show *?case*
 proof (*cases nellx*)
 case (*NNil x1*)
 then show *?thesis*
 by (*metis add-diff-cancel-left′ nappend-NNil ndropn-Suc-NCons nlength-NNil one-eSuc one-enat-def plus-1-eq-Suc the-enat.simps*)
 next
 case (*NCons x21 x22*)
 then show *?thesis*
 by (*metis Extended-Nat.eSuc-mono Suc.hyps Suc.prems add-diff-cancel-left′ diff-Suc-eq-diff-pred eSuc-enat enat-ord-code*(*4*) *nappend-NCons ndropn-Suc-NCons nlength-NCons order-less-imp-not-less plus-1-eq-Suc the-enat-eSuc*)
 qed
qed

lemma *nset-ndropn*:
  nset (*ndropn n nell*) $\subseteq$ *nset nell*
by *transfer* (*simp add*: *lset-ldropn-subset*)

lemma *ndropn-nmap*:
 ndropn n (*nmap f nell*) $=$ *nmap f* (*ndropn n nell*)
by *transfer*
   (*auto*,
    metis eSuc-epred enat-the-enat iless-Suc-eq infinity-ileE leD llength-eq-0 min.cobounded1 min.cobounded2)

lemma *nappend-ntaken-ndropn*:
 assumes (*Suc k*) $\leq$ *nlength nell*

shows *nappend* (*ntaken k nell*) (*ndropn* (*Suc k*) *nell*) = *nell*
using *assms*
by *transfer* (*simp add*: *min-absorb1*)

lemma *nfirst-eq-nnth-zero*:
nfirst nell = *nnth nell 0*
by (*metis ndropn-0 ndropn-nfirst*)

2.17 *nzip*

lemma *nzip-nhd*:
¬*is-NNil nellx* ∧ ¬*is-NNil nelly* ⟹ *nhd* (*nzip nellx nelly*) = (*nhd nellx*, *nhd nelly*)
by *transfer*
(*auto simp add*: *lzip-eq-LCons-conv*,
metis lzip-eq-LNil-conv)

lemma *nzip-ntl*:
¬*is-NNil nellx* ∧ ¬*is-NNil nelly* ⟹ *ntl*(*nzip nellx nelly*) = *nzip* (*ntl nellx*) (*ntl nelly*)
by *transfer*
(*auto*,
metis lhd-LCons-ltl ltl-lzip ltl-simps(*2*) *lzip-eq-LNil-conv*,
metis (*full-types*) *lhd-LCons-ltl lnull-def*,
metis lhd-LCons-ltl llist.collapse(*1*))

lemma *nzip-simps* [*simp*, *code*, *nitpick-simp*]:
nzip (*NNil b*) *nelly* = (*NNil* (*b*, (*nnth nelly 0*)))
nzip nellx (*NNil b*) = (*NNil* ((*nnth nellx 0*), *b*))
nzip (*NCons x nellx*) (*NCons y nelly*) = *NCons* (*x*, *y*) (*nzip nellx nelly*)
apply *transfer*
apply (*auto simp add*: *not-lnull-conv*)
apply (*metis lnth-0 min-enat-simps*(*3*) *the-enat-0 zero-enat-def*)
apply *transfer*
apply (*auto simp add*: *not-lnull-conv*)
apply (*metis lnth-0 min-enat-simps*(*3*) *the-enat-0 zero-enat-def*)
apply *transfer*
by (*auto simp add*: *not-lnull-conv*)

lemma *is-NNil-nzip* [*simp*]:
is-NNil (*nzip nellx nelly*) ⟷ (*is-NNil nellx*) ∨ (*is-NNil nelly*)
by *transfer*
(*auto simp add*: *lzip-eq-LCons-conv not-lnull-conv*,
metis lzip-eq-LNil-conv)

lemma *nzip-eq-NNil-conv*:
nzip nellx nelly = (*NNil* (*x*, *y*)) ⟷
((*is-NNil nellx*) ∨ (*is-NNil nelly*)) ∧ (*nnth nellx 0*) = *x* ∧ (*nnth nelly 0*) = *y*
by *auto*
(*metis is-NNil-nzip nellist.disc*(*1*),
metis Pair-inject is-NNil-nzip nellist.collapse(*1*) *nellist.disc*(*1*) *nlast-NNil nzip-simps*(*1*)

nzip-simps(*2*),
metis Pair-inject is-NNil-def is-NNil-nzip nellist.inject(*1*) *nzip-simps*(*1*) *nzip-simps*(*2*),
metis nellist.collapse(*1*) *nnth-NNil nzip-simps*(*1*),
metis nellist.collapse(*1*) *nnth-NNil nzip-simps*(*2*))

lemma *nzip-eq-NCons-conv*:
$nzip\ nellx\ nelly = (NCons\ z\ zs) \longleftrightarrow$
$(\exists\ x\ nellx'\ y\ nelly'.\ nellx = (NCons\ x\ nellx') \wedge nelly = (NCons\ y\ nelly') \wedge$
$z = (x,\ y) \wedge zs = (nzip\ nellx'\ nelly')\ )$
by (*cases nellx nelly rule*: *nellist.exhaust*[*case-product nellist.exhaust*])
auto

lemma *nzip-nappend*:
$nlength\ nellx = nlength\ nellu$
$\Longrightarrow nzip\ (nappend\ nellx\ nelly)\ (nappend\ nellu\ nellv) =$
$nappend\ (nzip\ nellx\ nellu)\ (nzip\ nelly\ nellv)$
by *transfer*
(*simp*,
meson co.enat.expand llength-eq-0 lzip-lappend)

lemma *nlength-nzip* [*simp*]:
$nlength\ (nzip\ nellx\ nelly) = (min\ (nlength\ nellx)\ (nlength\ nelly))$
by *transfer simp*

lemma *ntake-nzip*:
$ntake\ n\ (nzip\ nellx\ nelly) = nzip\ (ntake\ n\ nellx)\ (ntake\ n\ nelly)$
by *transfer*
(*simp add*: *ltake-lzip*)

lemma *ntaken-nzip*:
$ntaken\ n\ (nzip\ nellx\ nelly) = nzip\ (ntaken\ n\ nellx)\ (ntaken\ n\ nelly)$
by *transfer*
(*simp add*: *enat-0-iff*(*1*) *ltake-lzip*)

lemma *ndropn-nzip* [*simp*]:
$n \le nlength\ nellx \wedge n \le nlength\ nelly \Longrightarrow$
$ndropn\ n\ (nzip\ nellx\ nelly) = nzip\ (ndropn\ n\ nellx)\ (ndropn\ n\ nelly)$
by *transfer*
(*auto simp add*: *min-def*,
metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0,
metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0)

lemma *nzip-niterates*:
$nzip\ (niterates\ f\ x)\ (niterates\ g\ y) = niterates\ (\lambda(x,y).\ (f\ x,\ g\ y))\ (x,\ y)$
by *transfer*
(*simp add*: *lzip-iterates*)

lemma *nnth-nzip*:
assumes $n \le nlength\ nellx$
$n \le nlength\ nelly$

shows *nnth* (*nzip nellx nelly*) *n* = (*nnth nellx n*, *nnth nelly n*)
using *assms*
by *transfer*
(*auto simp add*: *min-def*,
metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-lzip)

lemma *nset-nzip*:
nset (*nzip nellx nelly*) =
{ (*nnth nellx n*, *nnth nelly n*) | *n*. *n* ≤ *min* (*nlength nellx*) (*nlength nelly*)}
by *transfer*
(*auto simp add*: *in-lset-conv-lnth*,
metis Pair-inject co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-lzip min.orderE the-enat.simps,
metis eSuc-epred iless-Suc-eq llength-eq-0 lnth-lzip)

lemma *nset-nzipD1*:
(x, y) ∈ *nset* (*nzip nellx nelly*) ⟹ x ∈ *nset nellx*
by *transfer*
(*meson lset-lzipD1*)

lemma *nset-nzipD2*:
(x, y) ∈ *nset* (*nzip nellx nelly*) ⟹ y ∈ *nset nelly*
by *transfer*
(*meson lset-lzipD2*)

lemma *nset-nzip-same* [*simp*]:
nset (*nzip nellx nellx*) = (λ *x*. (*x*, *x*)) ‘ *nset nellx*
by *transfer*
simp

lemma *nfinite-nzip* [*simp*]:
nfinite (*nzip nellx nelly*) ⟷ *nfinite nellx* ∨ *nfinite nelly*
by *transfer*
simp

lemma *nzip-eq-nappend-conv*:
assumes *eq*: *nzip nellx nelly* = *nappend nellu nellv*
shows ∃ *nellx′ nellx″ nelly′ nelly″*.
nellx = *nappend nellx′ nellx″* ∧ *nelly* = *nappend nelly′ nelly″* ∧
nlength nellx′ = *nlength nelly′* ∧
nellu = *nzip nellx′ nelly′* ∧ *nellv* = *nzip nellx″ nelly″*
using *assms*
apply *transfer*
using *lzip-eq-lappend-conv*
by (*auto simp add*: *lzip-eq-lappend-conv*)
fastforce

lemma *nzip-nmap* [*simp*]:
nzip (*nmap f nellx*) (*nmap g nelly*) = *nmap* (λ(*x*, *y*). (*f x*, *g y*)) (*nzip nellx nelly*)
by *transfer auto*

lemma *nzip-nmap1*:
nzip (*nmap f nellx*) *nelly* = *nmap* (λ(*x*, *y*). (*f x*, *y*)) (*nzip nellx nelly*)
by *transfer*
(*simp add*: *lzip-lmap1*)

lemma *nzip-nmap2*:
nzip nellx (*nmap f nelly*) = *nmap* (λ(*x*, *y*). (*x*, *f y*)) (*nzip nellx nelly*)
by *transfer*
(*simp add*: *lzip-lmap2*)

lemma *nmap-fst-nzip-conv-ntake*:
nmap fst (*nzip nellx nelly*) = *ntake* (*min* (*nlength nellx*) (*nlength nelly*)) *nellx*
by *transfer*
(*auto*,
metis co.enat.exhaust-sel llength-eq-0 lmap-fst-lzip-conv-ltake min-eSuc-eSuc)

lemma *nmap-snd-nzip-conv-ntake*:
nmap snd (*nzip nellx nelly*) = *ntake* (*min* (*nlength nellx*) (*nlength nelly*)) *nelly*
by *transfer*
(*auto*,
metis co.enat.exhaust-sel llength-eq-0 lmap-snd-lzip-conv-ltake min-eSuc-eSuc)

lemma *nzip-conv-nzip-ntake-min-nlength*:
nzip nellx nelly =
nzip (*ntake* (*min* (*nlength nellx*) (*nlength nelly*)) *nellx*)
(*ntake* (*min* (*nlength nellx*) (*nlength nelly*)) *nelly*)
by *transfer*
(*auto*,
metis co.enat.exhaust-sel epred-min i0-lb llength-lzip ltake-all ltake-lzip order-refl)

lemma *nellist-all2-conv-nzip*:
nellist-all2 P nellx nelly ⟷
nlength nellx = *nlength nelly* ∧ (∀ (*x*, *y*) ∈ *nset*(*nzip nellx nelly*). *P x y*)
using *nset-nzip*[*of nellx nelly*]
by (*auto simp add*: *nellist-all2-conv-all-nnth*)
blast

lemma *nellist-all2-all-nnthI*:
assumes *nlength nellx* = *nlength nelly*
⋀*n*. *enat n* ≤ *nlength nellx* ⟹ *P* (*nnth nellx n*) (*nnth nelly n*)
shows *nellist-all2 P nellx nelly*
using *assms* **by** (*simp add*: *nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nsetD1*:
assumes *nellist-all2 P nellx nelly*
x ∈ *nset nellx*
shows ∃ *y* ∈ *nset nelly*. *P x y*
using *assms*
by (*metis in-nset-conv-nnth nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nsetD2*:
assumes *nellist-all2 P nellx nelly*
y ∈ *nset nelly*
shows ∃ *x* ∈ *nset nellx*. *P x y*
using *assms*
by (*metis in-nset-conv-nnth nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nzipI*:
assumes *nellist-all2 P nellx nelly*
nellist-all2 P′ nellx′ nelly′
shows *nellist-all2* (*rel-prod P P′*) (*nzip nellx nellx′*) (*nzip nelly nelly′*)
using *assms*
proof (*coinduction arbitrary*: *nellx nellx′ nelly nelly′*)
case (*ilist-all2 nx nx′ ny ny′*)
then show *?case*
proof −
have *1*: *is-NNil* (*nzip nx nx′*) = *is-NNil* (*nzip ny ny′*)
by (*metis ilist-all2*(*1*) *ilist-all2*(*2*) *is-NNil-nzip nellist-all2-is-NNilD*)
have *2*: (*is-NNil* (*nzip nx nx′*) ⟹
is-NNil (*nzip ny ny′*) ⟹
rel-prod P P′ (*nlast* (*nzip nx nx′*)) (*nlast* (*nzip ny ny′*)))
by (*metis* (*no-types*, *lifting*) *enat-min-eq-0-iff ilist-all2*(*1*) *ilist-all2*(*2*)
is-NNil-nzip min-def-raw nellist.sel(*1*) *nellist-all2-conv-all-nnth*
nzip-eq-NNil-conv order-refl rel-prod-inject zero-enat-def)
have *3*: (¬ *is-NNil* (*nzip nx nx′*) ⟹
¬ *is-NNil* (*nzip ny ny′*) ⟹
((∃ *nellx nellx′ nelly nelly′*.
ntl (*nzip nx nx′*) = *nzip nellx nellx′* ∧
ntl (*nzip ny ny′*) = *nzip nelly nelly′* ∧
nellist-all2 P nellx nelly ∧ *nellist-all2 P′ nellx′ nelly′*) ∨
nellist-all2 (*rel-prod P P′*) (*ntl* (*nzip nx nx′*)) (*ntl* (*nzip ny ny′*))))
by (*auto simp add*: *nzip-eq-NCons-conv dest*: *nellist-all2-nhdD intro*: *nellist-all2-ntlI*)
(*meson ilist-all2*(*1*) *ilist-all2*(*2*) *nellist-all2-ntlI nzip-ntl*)
have *4*: ¬ *is-NNil* (*nzip nx nx′*) ⟶
¬ *is-NNil* (*nzip ny ny′*) ⟶
rel-prod P P′ (*nhd* (*nzip nx nx′*)) (*nhd* (*nzip ny ny′*))
by (*simp add*: *ilist-all2*(*1*) *ilist-all2*(*2*) *nellist-all2-nhdD nzip-nhd*)
have *5*: (¬ *is-NNil* (*nzip nx nx′*) ⟶
¬ *is-NNil* (*nzip ny ny′*) ⟶
rel-prod P P′ (*nhd* (*nzip nx nx′*)) (*nhd* (*nzip ny ny′*)) ∧
((∃ *nellx nellx′ nelly nelly′*.
ntl (*nzip nx nx′*) = *nzip nellx nellx′* ∧
ntl (*nzip ny ny′*) = *nzip nelly nelly′* ∧
nellist-all2 P nellx nelly ∧ *nellist-all2 P′ nellx′ nelly′*) ∨
nellist-all2 (*rel-prod P P′*) (*ntl* (*nzip nx nx′*)) (*ntl* (*nzip ny ny′*))))
using *3 4* **by** *blast*
show *?thesis*
using *1 2 3 4* **by** *presburger*
qed

qed

lemma *ndistinct-nzipI1*:
ndistinct nellx $\Longrightarrow$ *ndistinct* (*nzip nellx nelly*)
by *transfer*
(*simp add*: *ldistinct-lzipI1*)

lemma *ndistinct-nzipI2*:
ndistinct nelly $\Longrightarrow$ *ndistinct* (*nzip nellx nelly*)
by *transfer*
(*simp add*: *ldistinct-lzipI2*)

2.18 *niterates*

lemma *niterates-not-is-NNil* [*nitpick-simp*, *simp*]:
$\neg$ *is-NNil* (*niterates f x*)
by *transfer*
(*metis lfinite-LConsI lfinite-code*(*1*) *lfinite-iterates*)

lemma *nhd-niterates* [*code*, *simp*, *nitpick-simp*]:
nhd(*niterates f x*) = *x*
by *transfer*
(*metis lfinite-LConsI lfinite-LNil lfinite-iterates lhd-iterates*)

lemma *ntl-niterates* [*code*, *simp*, *nitpick-simp*]:
ntl(*niterates f x*) = *niterates f* (*f x*)
by *transfer*
(*simp*,
metis lfinite-LConsI lfinite-LNil lfinite-iterates)

lemma *nfinite-niterates* [*iff*]:
$\neg$ *nfinite* (*niterates f x*)
by *transfer simp*

lemma *niterates-nmap*:
niterates f x = *NCons x* (*nmap f* (*niterates f x*))
by *transfer*
(*meson iterates.disc-iff iterates-lmap*)

lemma [*simp*]:
fixes *f* :: $'a \Rightarrow 'a$
shows *is-NNil-funpow-nmap*: *is-NNil* ((*nmap f* $\widehat{}\widehat{}$ *n*) *nellx*) $\longleftrightarrow$ *is-NNil nellx*
and *nhd-funpow-nmap*: $\neg$ *is-NNil nellx* $\Longrightarrow$ *nhd* ((*nmap f* $\widehat{}\widehat{}$ *n*) *nellx*) = (*f* $\widehat{}\widehat{}$ *n*) (*nhd nellx*)
and *ntl-funpow-nmap*: $\neg$ *is-NNil nellx* $\Longrightarrow$ *ntl* ((*nmap f* $\widehat{}\widehat{}$ *n*) *nellx*) = (*nmap f* $\widehat{}\widehat{}$ *n*) (*ntl nellx*)
by (*induct n*) *simp-all*

lemma *niterates-equality*:
assumes *h*: $\bigwedge x.\ h\ x = NCons\ x\ (nmap\ f\ (h\ x))$
shows *h* = *niterates f*
proof –

{ **fix** x
  have ¬ *is-NNil* (h x) *nhd* (h x) = x *ntl* (h x) = *nmap* f (h x)
   by (*subst* h, *simp*)+ }
note [*simp*] = *this*
{ **fix** x
  define n :: *nat* **where** n = *0*
  have (*nmap* f ^^ n) (h x) = (*nmap* f ^^ n) (*niterates* f x)
    proof (*coinduction arbitrary*: n)
    case (*Eq-nellist nn*)
    then show *?case*
      proof −
       have *1*: *is-NNil* ((*nmap* f ^^ *nn*) (h x)) = *is-NNil* ((*nmap* f ^^ *nn*) (*niterates* f x))
         by *auto*
       have *2*: (*is-NNil* ((*nmap* f ^^ *nn*) (h x)) ⟶
                is-NNil ((*nmap* f ^^ *nn*) (*niterates* f x)) ⟶
                nlast ((*nmap* f ^^ *nn*) (h x)) = *nlast* ((*nmap* f ^^ *nn*) (*niterates* f x)))
         by *simp*
       have *3*: (¬ *is-NNil* ((*nmap* f ^^ *nn*) (h x)) ⟶
                ¬ *is-NNil* ((*nmap* f ^^ *nn*) (*niterates* f x)) ⟶
                nhd ((*nmap* f ^^ *nn*) (h x)) = *nhd* ((*nmap* f ^^ *nn*) (*niterates* f x)))
         by *simp*
       have *4*: (¬ *is-NNil* ((*nmap* f ^^ *nn*) (h x)) ⟶
                ¬ *is-NNil* ((*nmap* f ^^ *nn*) (*niterates* f x)) ⟶
                (*ntl* ((*nmap* f ^^ *nn*) (h x)) = (*nmap* f ^^ (*Suc nn*)) (h x) ∧
                    ntl ((*nmap* f ^^ *nn*) (*niterates* f x)) = (*nmap* f ^^ (*Suc nn*)) (*niterates* f x)))
         by (*metis* ‹⋀x. ¬ *is-NNil* (h x)› ‹⋀x. *ntl* (h x) = *nmap* f (h x)› *funpow-simps-right*(*2*)
           nellist.sel(*5*) *niterates-nmap niterates-not-is-NNil ntl-funpow-nmap o-apply*)
       show *?thesis* **using** *1 2 3 4* **by** *blast*
      qed
    qed
   }
  thus *?thesis* **by** *auto*
qed

lemma *nlength-niterates* [*simp*]:
nlength (*niterates* f x) = ∞
by *transfer auto*

lemma *ndropn-niterates*:
 ndropn n (*niterates* f x) = *niterates* f ((f ^^ n) x)
by *transfer*
  (*simp add*: *ldropn-iterates*)

lemma *nnth-niterates* [*simp*]:
 nnth (*niterates* f x) n = (f ^^ n) x
by *transfer auto*

lemma *nset-niterates*:
 nset (*niterates* f x) = { (f ^^ n) x|n. *True*}
by *transfer*

(*metis lset-iterates*)

lemma *nnth-niterates-Suc*:
nnth (*niterates Suc 0*) $i = i$
proof (*induct i*)
case *0*
then show *?case*
by *force*
next
case (*Suc i*)
then show *?case* **by** *simp*
qed

2.19 Filtering non-empty lazy lists *nfilter*

lemma *nfilter-NNil* [*simp*]:
shows *nfilter P* (*NNil b*) = *NNil b*
by *transfer auto*

lemma *nfilter-True* [*simp*]:
shows *nfilter* (λx. *True*) *nell* = *nell*
by *transfer auto*

lemma *nfilter-False-finite*:
assumes *nfinite nell*
shows *nfilter* (λ *x*. *False*) *nell* = *nell*
using *assms* **by** *transfer auto*

lemma *nfilter-NCons* [*simp*]:
assumes ($\exists$ $x \in$ *nset nell*. *P x*)
shows *nfilter P* (*NCons x nell*) = (*if P x then NCons x* (*nfilter P nell*) *else nfilter P nell*)
using *assms* **by** *transfer auto*

lemma *nfilter-NCons-a* [*simp*]:
assumes $\neg$($\exists$ $x \in$ *nset nell*. *P x*)
P x
shows *nfilter P* (*NCons x nell*) = (*NNil x*)
using *assms* **by** *transfer auto*

lemma *nfilter-expand*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
shows *nfilter P nell* =
(*if is-NNil nell then nell*
else
(*if* ($\exists x \in$ *nset*(*ntl nell*). *P x*) *then*
(*if P* (*nhd nell*) *then* (*NCons* (*nhd nell*) (*nfilter P* (*ntl nell*)))
else (*nfilter P* (*ntl nell*)))
else (*NNil* (*nhd nell*))))
using *assms* **by** (*cases nell*) *auto*

lemma *nset-nfilter*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows $nset\ (nfilter\ P\ nell) = nset\ nell \cap \{xa.\ P\ xa\}$
using *assms*
by *transfer auto*

lemma *exist-conj*:
assumes $\exists\ x \in nset\ (nfilter\ Q\ nell).\ P\ x$
$\exists\ x \in nset\ nell.\ Q\ x$
shows $\exists\ x \in nset\ nell.\ P\ x \wedge Q\ x$
using *assms*
by *transfer* (*auto split*: *if-split-asm*)

lemma *nfilter-nfilter* [*simp*]:
assumes $\exists\ x \in nset\ (nfilter\ Q\ nell).\ P\ x$
$\exists\ x \in nset\ nell.\ Q\ x$
shows $nfilter\ P\ (nfilter\ Q\ nell) = nfilter\ (\lambda x.\ P\ x \wedge Q\ x)\ nell$
using *assms*
proof (*transfer fixing*: *P Q*)
fix *xsa* :: *'a llist*
assume $\neg\ lnull\ xsa \wedge xsa = xsa$
assume *a1*: $Bex\ (lset\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa))\ P$
assume $Bex\ (lset\ xsa)\ Q$
then have $\neg\ lnull\ (lfilter\ Q\ xsa)$
by *simp*
then have $lfilter\ P\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa) = lfilter\ P\ (lfilter\ Q\ xsa) \wedge$
$\neg\ lnull\ (lfilter\ P\ (lfilter\ Q\ xsa)) \wedge$
$\neg\ lnull\ (if\ lnull\ (lfilter\ P\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa))$
$then\ if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa$
$else\ lfilter\ P\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa))$
using *a1* **by** (*auto split*: *if-split-asm*)
then show $\neg\ lnull\ (if\ lnull\ (lfilter\ P\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa))$
$then\ if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa$
$else\ lfilter\ P\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa)) \wedge$
$(if\ lnull\ (lfilter\ P\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa))$
$then\ if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa$
$else\ lfilter\ P\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa)) =$
$(if\ lnull\ (lfilter\ (\lambda a.\ P\ a \wedge Q\ a)\ xsa)\ then\ xsa\ else\ lfilter\ (\lambda a.\ P\ a \wedge Q\ a)\ xsa)$
using *lfilter-lfilter* **by** *auto*
qed

lemma *length-nfilter-le* [*simp*]:
$nlength\ (nfilter\ P\ nell) \le nlength\ nell$
by *transfer* (*simp add*: *epred-le-epredI llength-lfilter-ile*)

lemma *nfilter-nnth*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$i\le\ nlength\ (nfilter\ P\ nell)$
shows $(\exists\ k \le nlength\ nell.\ nnth(nfilter\ P\ nell)\ i = nnth\ nell\ k)$
using *assms nset-nfilter*[*of nell P*]

using *in-nset-conv-nnth*
by (*metis Int-iff*)

lemma *nfilter-nappend1*:
assumes ∀ *x*∈*nset nell*. ¬ *P x*
nfinite nell
∃ *y* ∈ *nset nell1*. *P y*
shows *nfilter P* (*nappend nell nell1*) = *nfilter P nell1*
using *assms* **by** *transfer auto*

lemma *nfilter-nappend2*:
assumes ∀ *x*∈*nset nell*. ¬ *P x*
∃ *y*∈*nset nell1*. *P y*
shows *nfilter P* (*nappend nell1 nell*) = *nfilter P nell1*
using *assms*
by *transfer*
(*auto split*: *if-split-asm*,
meson in-lset-lappend-iff,
metis lappend-LNil2 lappend-inf lfilter-empty-conv lfilter-lappend-lfinite)

lemma *nfilter-nappend* [*simp*]:
assumes (∃ *x* ∈ *nset* (*nappend nell nell1*). *P x*)
(∃ *x* ∈ *nset nell*. *P x*)
(∃ *x* ∈ *nset nell1*. *P x*)
shows *nfilter P* (*nappend nell nell1*) =
(*if nfinite nell then nappend* (*nfilter P nell*) (*nfilter P nell1*)
else (*nfilter P nell*))
proof (*cases nfinite nell*)
case *True*
then show *?thesis* **using** *assms* **by** *transfer auto*
next
case *False*
then show *?thesis* **using** *assms* **by** *transfer* (*auto simp add*: *lappend-inf*)
qed

lemma *nfilter-nmap*:
shows *nfilter P* (*nmap f nell*) = *nmap f* (*nfilter* (*P*∘*f*) *nell*)
by *transfer* (*auto simp add*: *lfilter-lmap*)

lemma *nlength-nfilter-nmap*[*simp*]:
shows *nlength* (*nfilter P* (*nmap f nell*)) = *nlength*(*nfilter* (*P* ∘ *f*) *nell*)
by (*simp add*: *nfilter-nmap*)

lemma *nfilter-is-subset* [*simp*]:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows *nset* (*nfilter P nell*) ≤ *nset nell*
using *assms* **by** (*simp add*: *nset-nfilter*)

lemma *nfilter-cong*[*fundef-cong*]:
assumes *nell* = *nell1*

$(\bigwedge x.\ x \in nset\ nell1 \Longrightarrow P\ x = Q\ x)$
shows *nfilter P nell = nfilter Q nell1*
using *assms* **by** *transfer auto*

lemma *nset-nappend*:
$nset\ (nappend\ nell\ nell1) = (if\ nfinite\ nell\ then\ nset\ nell \cup nset\ nell1\ else\ nset\ nell)$
by *transfer* (*simp add*: *lappend-inf*)

lemma *split-nellist-nappend*:
assumes $\exists\ i \le nlength\ nell.\ nnth\ nell\ i = x \wedge P\ (nnth\ nell\ i) \wedge$
$(\forall\ j.\ j \neq i \wedge j \le nlength\ nell \longrightarrow \neg\ P(nnth\ nell\ j))$
shows $P\ x \wedge$
$(\ nell = (NNil\ x) \vee$
$(\exists\ vs.\ nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v)) \vee$
$(\exists\ us.\ nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg\ P\ u)) \vee$
$(\exists\ us\ vs\ .\ nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
$(\forall\ u \in nset\ us.\ \neg\ P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
$)$
proof –
obtain *i* **where** *1*: $i \le nlength\ nell \wedge nnth\ nell\ i = x \wedge P\ (nnth\ nell\ i) \wedge$
$(\forall\ j.\ j \neq i \wedge j \le nlength\ nell \longrightarrow \neg\ P(nnth\ nell\ j))$
using *assms* **by** *auto*
have *01*: $(\forall\ j.\ (j<i \vee i<j) \wedge j \le nlength\ nell \longrightarrow \neg\ P(nnth\ nell\ j))$
using *1*
by *auto*
have *2*: $i=0 \wedge nlength\ nell = 0 \Longrightarrow P\ x \wedge nell = (NNil\ x)$
by (*metis 1 ndropn-0 ndropn-eq-NNil the-enat-0 zero-enat-def*)
have *3*: $i=0 \wedge nlength\ nell > 0 \Longrightarrow P\ x \wedge\ nell = (NCons\ x\ (ntl\ nell))\ \wedge (\forall\ v \in nset\ (ntl\ nell).\ \neg\ P\ v)$
using *1 in-nset-conv-nnth*[*of - ntl nell*]
by (*metis Suc-ile-eq iless-Suc-eq nat.simps*(*3*) *nellist.disc*(*2*) *nellist.exhaust-sel nlength-NCons nlength-NNil nnth-0-conv-nhd nnth-ntl not-less-iff-gr-or-eq*)
have *4*: $i=0 \wedge nlength\ nell > 0 \Longrightarrow P\ x \wedge (\exists\ vs.\ nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
using *3* **by** *auto*
have *5*: $i > 0 \wedge i = nlength\ nell \wedge nfinite\ nell \Longrightarrow P\ x \wedge nell = nappend\ (ntaken\ (i-1)\ nell)\ (NNil\ x)$
using *1*
by *transfer*
(*auto*,
metis eSuc-epred lappend-lbutlast-llast-id-lfinite lbutlast-conv-ltake llast-conv-lnth llength-eq-0 the-enat.simps)
have *6*: $i > 0 \wedge i = nlength\ nell \wedge nfinite\ nell \Longrightarrow\ (\forall\ u \in nset\ (ntaken\ (i-1)\ nell).\ \neg\ P\ u)$
using *1*
by *transfer*
(*auto*,
metis in-lset-conv-lnth lbutlast-conv-ltake less-imp-le llength-lbutlast lnth-ltake min.strict-order-iff)
have *7*: $i > 0 \wedge i = nlength\ nell \wedge nfinite\ nell \Longrightarrow P\ x \wedge (\exists\ us.\ nell = nappend\ us\ (NNil\ x) \wedge$
$nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg\ P\ u))$
using *5 6* **by** *auto*
have *8*: $i > 0 \wedge i < nlength\ nell \Longrightarrow$

P x ∧ *nell* = *nappend* (*ntaken* (*i*−*1*) *nell*) (*NCons x* (*ndropn* (*i*+*1*) *nell*))
using *1*
by *transfer*
(*auto*,
metis co.enat.collapse eSuc-enat ileI1 iless-Suc-eq lappend-ltake-enat-ldropn ldropn-Suc-conv-ldropn llength-eq-0 min.absorb1 the-enat.simps)
have *9*: *i* >*0* ∧ *i* < *nlength nell* ⟹ *nfinite* (*ntaken* (*i*−*1*) *nell*)
using *enat-ord-code*(*4*) *nfinite-ntaken* **by** *blast*
have *10*: *i* >*0* ∧ *i* < *nlength nell* ⟹ (∀ *u* ∈ *nset* (*ntaken* (*i*−*1*) *nell*). ¬ *P u*)
using *01 in-nset-conv-nnth*[*of* - (*ntaken* (*i*−*1*) *nell*)] **by** *simp*
(*metis Suc-pred le-imp-less-Suc min.orderE ntaken-nnth*)
have *11*: *i* >*0* ∧ *i* < *nlength nell* ⟹ (∀ *v* ∈ *nset* (*ndropn* (*i*+*1*) *nell*). ¬ *P v*)
using *1 01 in-nset-conv-nnth*[*of* - (*ndropn* (*i*+*1*) *nell*)]
by *simp*
(*metis Suc-ile-eq iless-Suc-eq is-NNil-ndropn le-add1 le-imp-less-Suc ndropn-Suc-conv-ndropn ndropn-ndropn ndropn-nlength nlength-NCons not-less*)
have *12*: *i* >*0* ∧ *i* < *nlength nell* ⟹ (∃ *us vs* . *nell* = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*))
using *8 9 10 11* **by** *blast*
show *?thesis*
by (*metis 1 12 2 4 7 dual-order.order-iff-strict neq0-conv nlength-eq-enat-nfiniteD zero-enat-def*)
qed

lemma *nellist-split-2-first*:
assumes *0* < *nlength nell*
shows *nell* = (*NCons* (*nnth nell 0*) (*ntl nell*))
using *assms* **by** (*metis ndropn-0 ndropn-Suc-conv-ndropn nellist.sel*(*5*) *zero-enat-def*)

lemma *nellist-split-2-last*:
assumes *0*< *i*
i = *nlength nell*
nfinite nell
shows *nell* = *nappend* (*ntaken* (*i*−*1*) *nell*) (*NNil* (*nnth nell i*))
using *assms*
by *transfer*
(*simp*,
metis Suc-ile-eq co.enat.exhaust-sel eSuc-enat llength-eq-0 ltake-Suc-conv-snoc-lnth ltake-all order-refl the-enat.simps)

lemma *nellist-split-3*:
assumes *0* < *i*
i < *nlength nell*
shows *nell* = *nappend* (*ntaken* (*i*−*1*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*i*+*1*) *nell*))
using *assms* **by** *transfer*
(*auto*,
metis eSuc-enat eSuc-epred ileI1 iless-Suc-eq lappend-ltake-enat-ldropn ldropn-Suc-conv-ldropn less-imp-le llength-eq-0 min-def the-enat.simps)

lemma *NNil-eq-nfilterD*:

assumes $\exists\ x \in$ *nset nell*. *P x*
$(NNil\ x) = nfilter\ P\ nell$
shows $(\exists\ us\ vs\ .\ (nell = (NNil\ x) \vee$
$(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v)) \vee$
$(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg\ P\ u)) \vee$
$(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
$(\forall\ u \in nset\ us.\ \neg\ P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
$) \wedge P\ x)$
proof −
have *1*: $(\exists\ i \leq nlength\ nell.\ P\ (nnth\ nell\ i))$
by (*metis assms*(*1*) *in-nset-conv-nnth*)
obtain *i* **where** *2*: $i \leq nlength\ nell \wedge P\ (nnth\ nell\ i)$
using *1* **by** *auto*
have *3*: $i = 0 \wedge nlength\ nell = 0 \Longrightarrow P\ x \wedge nell = (NNil\ x)$
by (*metis 2 assms*(*2*) *ndropn-0 ndropn-eq-NNil nfilter-NNil the-enat-0 zero-enat-def*)
have *4*: $i{=}0 \wedge nlength\ nell > 0 \Longrightarrow P\ x \wedge\ nell = (NCons\ x\ (ntl\ nell)) \wedge (\forall\ v \in nset\ (ntl\ nell).\ \neg\ P\ v)$
using *2 assms nellist-split-2-first*[*of nell*] *nfilter-expand*[*of nell P*]
by (*metis nellist.distinct*(*1*) *nellist.sel*(*3*) *nlast-NNil*)
have *5*: $i{=}0 \wedge nlength\ nell > 0 \Longrightarrow P\ x \wedge (\exists\ vs.\ nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg\ P\ v))$
using *4* **by** *auto*
have *60*: $0 < i\ \wedge i = nlength\ nell \wedge nfinite\ nell \Longrightarrow x = (nnth\ nell\ i)$
using *2 assms nellist-split-2-last*[*of i nell*]
proof *simp*
assume *a1*: $nell = nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NNil\ (nnth\ nell\ i))$
assume *a2*: $P\ (nnth\ nell\ i)$
assume *a3*: $NNil\ x = nfilter\ P\ nell$
assume *a4*: $\exists x{\in}nset\ nell.\ P\ x$
have *f5*: $\forall as\ p\ asa.\ ((\exists a.\ (a{::}'a) \in nset\ as \wedge p\ a) \vee \neg\ nfinite\ as \vee (\forall a.\ a \notin nset\ asa \vee \neg\ p\ a))$
$\vee\ nfilter\ p\ (nappend\ as\ asa) = nfilter\ p\ asa$
by (*metis* (*full-types*) *nfilter-nappend1*)
have *f7*: $\exists a.\ a \in nset\ (NNil\ (nnth\ nell\ i)) \wedge P\ a$
using *a2* **by** *auto*
have *f8*: $nfilter\ P\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NNil\ (nnth\ nell\ i))) \neq$
$nappend\ (nfilter\ P\ (ntaken\ (i - Suc\ 0)\ nell))\ (nfilter\ P\ (NNil\ (nnth\ nell\ i)))$
using *a3 a1* **by** (*metis* (*full-types*) *is-NNil-nappend nellist.disc*(*1*))
have *f9*: $nfinite\ (ntaken\ (i - Suc\ 0)\ nell)$
by *simp*
obtain *aaa* :: *'a* **where**
f9: $aaa \in nset\ nell \wedge P\ aaa$
using *a4* **by** *blast*
then have $aaa \in nset\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NNil\ (nnth\ nell\ i)))$
using *a1* **by** *presburger*
then have *f10*: $\forall a.\ a \notin nset\ (ntaken\ (i - Suc\ 0)\ nell) \vee \neg\ P\ a$
using *f9 f8 f7* **by** (*meson nfilter-nappend nfinite-ntaken*)
then show *?thesis*
using *f9 f7 f5 a3 a1*
proof −
have $NNil\ (nnth\ nell\ i) = NNil\ x$
using *nfilter-NNil*[*of P* (*nnth nell i*)] **by** (*metis* (*no-types*) *f10 a1 a3 f5 f7 nfinite-ntaken*)
then show *?thesis*

by *blast*
qed
qed
have *6*: $0 < i \wedge i = nlength\ nell \wedge nfinite\ nell \Longrightarrow P\ x \wedge\ nell = nappend\ (ntaken\ (i-1)\ nell)\ (NNil\ x)$
using *2 60 assms nellist-split-2-last[of i nell]*
by *blast*
have *7*: $i > 0 \wedge i = nlength\ nell \wedge nfinite\ nell \Longrightarrow (\forall\ u \in nset\ (ntaken\ (i-1)\ nell).\ \neg\ P\ u)$
using *assms 6 nfilter-nappend[of (ntaken (i−1) nell) - P]*
by *(metis is-NNil-nappend nellist.disc(1) nfinite-ntaken split-nellist-a)*
have *8*: $i > 0 \wedge i = nlength\ nell \wedge nfinite\ nell \Longrightarrow P\ x \wedge (\exists\ us.\ nell = nappend\ us\ (NNil\ x) \wedge$
$nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg\ P\ u))$
using *6 7* **by** *auto*
have *9*: $i > 0 \wedge i < nlength\ nell \Longrightarrow$
$P\ x \wedge nell = nappend\ (ntaken\ (i-1)\ nell)\ (NCons\ x\ (ndropn\ (i+1)\ nell))$
using *2 assms nellist-split-3[of i nell]*
nfilter-nappend1[of (ntaken (i−1) nell) P (NCons x (ndropn (i+1) nell))]
nfilter-nappend[of (ntaken (i − 1) nell) - P]
proof *simp*
assume *a1*: $enat\ i \leq nlength\ nell \wedge P\ (nnth\ nell\ i)$
assume *a2*: $NNil\ x = nfilter\ P\ nell$
assume *a3*: $\exists\, x \in nset\ nell.\ P\ x$
assume *a4*: $nell = nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell))$
have *f5*: $P\ (nnth\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)))\ i)$
using *a1 a4* **by** *auto*
have *f6*: $NNil\ x = nfilter\ P\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)$
$nell)))$
using *a2 a4* **by** *auto*
have *f7*: $\forall\, as\ p\ asa.\ ((\exists\, a.\ (a::'a) \in nset\ as \wedge p\ a) \vee \neg\ nfinite\ as \vee (\forall\, a.\ a \notin nset\ asa \vee \neg\ p\ a)) \vee$
$nfilter\ p\ (nappend\ as\ asa) = nfilter\ p\ asa$
by *(metis (full-types) nfilter-nappend1)*
have $nfilter\ P\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell))) =$
$nappend\ (nfilter\ P\ (ntaken\ (i - Suc\ 0)\ nell))\ (nfilter\ P\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)$
$nell)))$
$\longrightarrow$
$nappend\ (nfilter\ P\ (ntaken\ (i - Suc\ 0)\ nell))\ (nfilter\ P\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)$
$nell))) =$
$NNil\ x$
using *f6* **by** *presburger*
then have $nfilter\ P\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)))$
$\neq$
$nappend\ (nfilter\ P\ (ntaken\ (i - Suc\ 0)\ nell))\ (nfilter\ P\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)$
$nell)))$
by *(metis (no-types) is-NNil-nappend nellist.disc(1))*
then have *f9*: $(\forall\, a.\ a \notin nset\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)) \vee \neg\ P\ a) \vee$
$nfilter\ P\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)))$
$=$
$nfilter\ P\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)) \vee$
$(\forall\, a.\ a \notin nset\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)$
$nell)))$
$\vee \neg\ P\ a)$

using *nfilter-nappend*[*of* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)) *P*]
nfinite-ntaken[*of* (*i* − *Suc 0*) *nell*]
by (*metis f7*)
obtain *aaa* :: ′*a* **where** *f10*: *aaa* ∈ *nset nell* ∧ *P aaa*
using *a3* **by** *blast*
then have *f11*: *aaa* ∈ *nset* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
using *a4* **by** *presburger*
have *f12*: *nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))) *i* ∈ *nset* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))
using *a4* **by** *fastforce*
then have *NNil x* = *nfilter P* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))
using *f11 f10 f9 f6 f5* **by** (*metis* (*no-types*))
then have *NNil x* = *nfilter P* (*NCons* (*nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*)
(*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
i) (*ndropn* (*Suc i*) *nell*))
using *a4* **by** *presburger*
then have *f13*: ∀ *a*. *a* ∉ *nset* (*ndropn* (*Suc i*) *nell*) ∨ ¬ *P a*
using *f5* **by** (*metis* (*full-types*) *nellist.distinct*(*1*) *nfilter-NCons*)
have *nfilter P* (*NCons* (*nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*)
(*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*)))
i) (*ndropn* (*Suc i*) *nell*)) = *NNil x*
using ‹*NNil x* = *nfilter P* (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))› *a4* **by** *presburger*
then have *x* = *nnth* (*nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons* (*nnth nell i*) (*ndropn* (*Suc i*) *nell*))) *i*
using *f13 f5* **by** *simp*
then have *f14*: *x* = *nnth nell i*
using *a4* **by** *presburger*
then have *nell* = *nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons x* (*ndropn* (*Suc i*) *nell*))
using *a4* **by** *blast*
then show *P x* ∧ *nell* = *nappend* (*ntaken* (*i* − *Suc 0*) *nell*) (*NCons x* (*ndropn* (*Suc i*) *nell*))
using *f14 a1* **by** *blast*
qed
have *10*: *i* >*0* ∧ *i* < *nlength nell* ⟹ (∀ *u* ∈ *nset* (*ntaken* (*i*−*1*) *nell*). ¬ *P u*)
using *9 assms nfilter-nappend*[*of* (*ntaken* (*i*−*1*) *nell*) - *P*]
by (*metis is-NNil-nappend nellist.disc*(*1*) *nellist.set-intros*(*2*) *nfinite-ntaken*)
have *11*: *i* >*0* ∧ *i* < *nlength nell* ⟹ (∀ *v* ∈ *nset* (*ndropn* (*i*+*1*) *nell*). ¬ *P v*)
using *9 10 assms nfilter-nappend*[*of* (*ntaken* (*i*−*1*) *nell*) - *P*]
nfilter-nappend1[*of* (*ntaken* (*i*−*1*) *nell*) *P* (*NCons x* (*ndropn* (*i*+*1*) *nell*))]
by (*metis nellist.distinct*(*1*) *nellist.set-intros*(*2*) *nfilter-NCons nfinite-ntaken*)
have *12*: *i* >*0* ∧ *i* < *nlength nell* ⟹
P x ∧ (∃ *us vs* . *nell* = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧
(∀ *u* ∈ *nset us*. ¬ *P u*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*))
using *9 10 11* **using** *assms*(*1*) **by** *fastforce*
show *?thesis*
by (*metis 12 2 3 5 8 dual-order.order-iff-strict neq0-conv nlength-eq-enat-nfiniteD zero-enat-def*)
qed

lemma *nfilter-eq-NNilD*:
assumes ∃ *x* ∈ *nset nell*. *P x*

nfilter P nell = (*NNil x*)
shows (∃ *us vs* . (*nell* = (*NNil x*) ∨
(*nell* = (*NCons x vs*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*)) ∨
(*nell* = *nappend us* (*NNil x*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∨
(*nell* = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧
(∀ *u* ∈ *nset us*. ¬ *P u*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*))
) ∧ *P x*)
using *assms NNil-eq-nfilterD*[*of nell P x*] **by** *simp*

lemma *nfilter-eq-NNil-iff*:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows (*nfilter P nell* = (*NNil x*)) ⟷
(∃ *us vs* . (*nell* = (*NNil x*) ∨
(*nell* = (*NCons x vs*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*)) ∨
(*nell* = *nappend us* (*NNil x*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∨
(*nell* = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧
(∀ *u* ∈ *nset us*. ¬ *P u*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*))
) ∧ *P x*)
proof −
have *1*: (*nfilter P nell* = (*NNil x*)) ⟹
(∃ *us vs* . (*nell* = (*NNil x*) ∨
(*nell* = (*NCons x vs*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*)) ∨
(*nell* = *nappend us* (*NNil x*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∨
(*nell* = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧
(∀ *u* ∈ *nset us*. ¬ *P u*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*))
) ∧ *P x*)
using *assms nfilter-eq-NNilD*[*of nell P x*] **by** *simp*
have *2*: (∃ *us vs* . (*nell* = (*NNil x*) ∨
(*nell* = (*NCons x vs*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*)) ∨
(*nell* = *nappend us* (*NNil x*) ∧ *nfinite us* ∧ (∀ *u* ∈ *nset us*. ¬ *P u*)) ∨
(*nell* = *nappend us* (*NCons x vs*) ∧ *nfinite us* ∧
(∀ *u* ∈ *nset us*. ¬ *P u*) ∧ (∀ *v* ∈ *nset vs*. ¬ *P v*))
) ∧ *P x*) ⟹ (*nfilter P nell* = (*NNil x*))
using *nfilter-NCons-a nfilter-NNil*[*of P x*]
by (*auto simp add*: *nfilter-nappend1*)
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *NCons-eq-nfilterD-help*:
assumes ¬ *lnull ys*
¬ *lnull xs*
LCons x xs = *lfilter P ys*
xa ∈ *lset ys*
P xa
shows ∃ *us*. ¬ *lnull us* ∧
(∃ *vs*. (∃ *x*∈*lset vs*. *P x*) ∧
((∃ *x*∈*lset vs*. *P x*) ⟶
¬ *lnull vs* ∧ (*ys* = *LCons x vs* ∨ *ys* = *lappend us* (*LCons x vs*) ∧ *lfinite us* ∧
(∀ *u*∈*lset us*. ¬ *P u*)) ∧ *P x* ∧ *xs* = *lfilter P vs*))

unfolding *lnull-def* **using** *assms lfilter-eq-LConsD*[*of P ys x xs*]
by (*metis diverge-lfilter-LNil lappend-code*(*1*) *lfilter-LNil llist.disc*(*1*))

lemma *NCons-eq-nfilterD*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
(*NCons x nell1*) = *nfilter P nell*
shows ($\exists$ *us vs*.
(*nell* = (*NCons x vs*) $\vee$
nell = *nappend us* (*NCons x vs*) $\wedge$ *nfinite us* $\wedge$ ($\forall$ $u \in$ *nset us*. $\neg$ *P u*)) $\wedge$
($\exists$ $x \in$ *nset vs*. *P x*) $\wedge$ *P x* $\wedge$ *nell1* = *nfilter P vs*)
using *assms* **by** *transfer* (*auto split*: *if-split-asm simp add*: *NCons-eq-nfilterD-help*)

lemma *nfilter-eq-NConsD*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
nfilter P nell = (*NCons x nell1*)
shows ($\exists$ *us vs*.
(*nell* = (*NCons x vs*) $\vee$
nell = *nappend us* (*NCons x vs*) $\wedge$ *nfinite us* $\wedge$ ($\forall$ $u \in$ *nset us*. $\neg$ *P u*)) $\wedge$
($\exists$ $x \in$ *nset vs*. *P x*) $\wedge$ *P x* $\wedge$ *nell1* = *nfilter P vs*)
using *assms NCons-eq-nfilterD*[*of nell P x nell1*] **by** *simp*

lemma *nfilter-eq-NCons-iff*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
shows *nfilter P nell* = (*NCons x nell1*) $\longleftrightarrow$
($\exists$ *us vs*.
(*nell* = (*NCons x vs*) $\vee$
nell = *nappend us* (*NCons x vs*) $\wedge$ *nfinite us* $\wedge$ ($\forall$ $u \in$ *nset us*. $\neg$ *P u*)) $\wedge$
($\exists$ $x \in$ *nset vs*. *P x*) $\wedge$ *P x* $\wedge$ *nell1* = *nfilter P vs*)
proof –
have *1*: *nfilter P nell* = (*NCons x nell1*) $\Longrightarrow$
($\exists$ *us vs*.
(*nell* = (*NCons x vs*) $\vee$
nell = *nappend us* (*NCons x vs*) $\wedge$ *nfinite us* $\wedge$ ($\forall$ $u \in$ *nset us*. $\neg$ *P u*)) $\wedge$
($\exists$ $x \in$ *nset vs*. *P x*) $\wedge$ *P x* $\wedge$ *nell1* = *nfilter P vs*)
using *assms nfilter-eq-NConsD*[*of nell P x nell1*] **by** *simp*
have *2*: ($\exists$ *us vs*.
(*nell* = (*NCons x vs*) $\vee$
nell = *nappend us* (*NCons x vs*) $\wedge$ *nfinite us* $\wedge$ ($\forall$ $u \in$ *nset us*. $\neg$ *P u*)) $\wedge$
($\exists$ $x \in$ *nset vs*. *P x*) $\wedge$ *P x* $\wedge$ *nell1* = *nfilter P vs*) $\Longrightarrow$ *nfilter P nell* = (*NCons x nell1*)
using *nfilter-nappend1*[*of - P*] *nfilter-NCons*[*of - P x*]
by (*auto*, *meson nfilter-NCons*, *metis*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *nfilter-id-conv*:
assumes ($\exists$ $x \in$ *nset nell*. *P x*)
shows (*nfilter P nell* = *nell*) = ($\forall$ $x\in$*nset nell*. *P x*) (**is** *?lhs* = *?rhs*)
using *assms* **by** *transfer* (*auto simp add*: *lfilter-id-conv*)

lemma *nfilter-idem*:

assumes (∃ *x* ∈ *nset nell*. *P x*)
shows *nfilter P* (*nfilter P nell*) = *nfilter P nell*
using *assms* **by** *transfer auto*

lemma *nfilter-ndropn-nlength*:
assumes *k* ≤ *nlength nell*
∃ *x* ∈ *nset* ((*ndropn k nell*)). *P x*
shows *nlength*(*nfilter P* ((*ndropn k nell*))) ≤ *nlength* (*nfilter P* (*nell*))
using *assms*
by *transfer*
(*auto simp add*: *min-def dest*: *in-lset-ldropnD*,
metis co.enat.exhaust-sel epred-le-epredI iless-Suc-eq lfilter-ldropn-llength llength-eq-0)

2.20 Setup for Lifting/Transfer

2.20.1 Relator and predicator properties

abbreviation *nellist-all* == *pred-nellist*

2.20.2 Transfer rules for the Transfer package

context includes *lifting-syntax*
begin

lemma *set1-pre-nellist-transfer* [*transfer-rule*]:
(*rel-pre-nellist A C* ===> *rel-set A*) *set1-pre-nellist set1-pre-nellist*
by(*auto simp add*: *rel-pre-nellist-def vimage2p-def rel-fun-def set1-pre-nellist-def rel-set-def collect-def sum-set-defs prod-set-defs elim*: *rel-sum.cases split*: *sum.split-asm*)

lemma *set2-pre-nellist-transfer* [*transfer-rule*]:
(*rel-pre-nellist A C* ===> *rel-set C*) *set2-pre-nellist set2-pre-nellist*
by (*auto simp add*: *rel-pre-nellist-def vimage2p-def rel-fun-def set2-pre-nellist-def rel-set-def collect-def sum-set-defs prod-set-defs elim*: *rel-sum.cases split*: *sum.split-asm*)

lemma *NCons-transfer2* [*transfer-rule*]:
(*A* ===> *nellist-all2 A* ===> *nellist-all2 A*) *NCons NCons*
unfolding *rel-fun-def* **by** *simp*
declare *NCons-transfer* [*transfer-rule*]

lemma *case-nellist-transfer* [*transfer-rule*]:
((*A* ===> *C*) ===> (*A* ===> *nellist-all2 A* ===> *C*) ===> *nellist-all2 A* ===> *C*)
case-nellist case-nellist
unfolding *rel-fun-def*
by (*simp add*: *nellist-all2-NNil1 nellist-all2-NNil2 split*: *nellist.split*)

lemma *unfold-nellist-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> (*A* ===> *C*) ===> (*A* ===> *C*) ===> (*A* ===> *A*) ===> *A* ===> *nellist-all2 C*)
unfold-nellist unfold-nellist
proof(*rule rel-funI*)+
fix *IS-NNIL1* :: *'a* ⇒ *bool* **and** *IS-NNIL2*

NLAST1 NLAST2 NHD1 NHD2 NTL1 NTL2 x y
assume *rel*: (*A* ===> (=)) *IS-NNIL1 IS-NNIL2* (*A* ===> *C*) *NLAST1 NLAST2*
(*A* ===> *C*) *NHD1 NHD2* (*A* ===> *A*) *NTL1 NTL2*
and *A x y*
show *nellist-all2 C* (*unfold-nellist IS-NNIL1 NLAST1 NHD1 NTL1 x*)
(*unfold-nellist IS-NNIL2 NLAST2 NHD2 NTL2 y*)
using ‹*A x y*› **using** *rel* **by** (*coinduction arbitrary*: *x y*) (*auto 4 4 elim*: *rel-funE*)
qed

lemma *corec-nellist-transfer* [*transfer-rule*]:
((*A* ===> (=)) ===> (*A* ===> *C*) ===> (*A* ===> *C*) ===> (*A* ===> (=)) ===> (*A* ===> *nellist-all2 C*)
===> (*A* ===> *A*) ===> *A* ===> *nellist-all2 C*) *corec-nellist corec-nellist*
by (*simp add*: *nellist.corec-transfer*)

lemma *ntl-transfer2* [*transfer-rule*]:
(*nellist-all2 A* ===> *nellist-all2 A*) *ntl ntl*
unfolding *ntl-def*[*abs-def*] **by** *transfer-prover*
declare *ntl-transfer* [*transfer-rule*]

lemma *nset-transfer2* [*transfer-rule*]:
(*nellist-all2 A* ===> *rel-set A*) *nset nset*
by (*simp add*: *nellist.set-transfer*)

lemma *nmap-transfer4* [*transfer-rule*]:
((*A* ===> *B*) ===> *nellist-all2 A* ===> *nellist-all2 B*) *nmap nmap*
by (*simp add*: *nellist.map-transfer*)
declare *nmap-transfer* [*transfer-rule*]

lemma *is-NNil-transfer2* [*transfer-rule*]:
(*nellist-all2 A* ===> (=)) *is-NNil is-NNil*
by(*auto dest*: *nellist-all2-is-NNilD*)
declare *is-NNil-transfer* [*transfer-rule*]

lemma *snocn-transfer2* [*transfer-rule*]:
(*nellist-all2 A* ===> *A* ===> *nellist-all2 A*) *snocn snocn*
unfolding *rel-fun-def*
by (*metis nappend-snocn nellist-all2-NNil nellist-all2-nappendI*)
declare *snocn-transfer*[*transfer-rule*]

lemma *nappend-transfer* [*transfer-rule*]:
(*nellist-all2 A* ===> (*nellist-all2 A*) ===> *nellist-all2 A*) *nappend nappend*
by(*auto intro*: *nellist-all2-nappendI elim*: *rel-funE*)
declare *nappend.transfer* [*transfer-rule*]

lemma *lappendn-transfer* [*transfer-rule*]:
(*llist-all2 A* ===> *nellist-all2 A* ===> *nellist-all2 A*) *lappendn lappendn*
unfolding *rel-fun-def*
by *transfer*(*auto intro*: *llist-all2-lappendI*)
declare *lappendn.transfer* [*transfer-rule*]

lemma *nellist-of-llist-a-transfer2* [*transfer-rule*]:
  (*A* ===> *llist-all2 A* ===> *nellist-all2 A*) *nellist-of-llist-a nellist-of-llist-a*
by (*simp add*: *rel-funI*)
declare *nellist-of-llist-a-transfer* [*transfer-rule*]

lemma *nlength-transfer* [*transfer-rule*]:
  (*nellist-all2 A* ===> (=)) *nlength nlength*
by(*auto dest*: *nellist-all2-nlengthD*)
declare *nlength.transfer* [*transfer-rule*]

lemma *ndropn-transfer* [*transfer-rule*]:
  ((=) ===> *nellist-all2 A* ===> *nellist-all2 A*) *ndropn ndropn*
unfolding *rel-fun-def*
by *transfer* (*auto intro*: *llist-all2-ldropnI simp add*: *llist-all2-ldropnI llist-all2-llengthD*)
declare *ndropn.transfer* [*transfer-rule*]

lemma *ntake-transfer* [*transfer-rule*]:
  ((=) ===> *nellist-all2 A* ===> *nellist-all2 A*) *ntake ntake*
unfolding *rel-fun-def*
by *transfer* (*auto simp add*: *llist-all2-ltakeI*)
declare *ntake.transfer* [*transfer-rule*]

lemma *ntaken-transfer* [*transfer-rule*]:
  ((=) ===> *nellist-all2 A* ===> *nellist-all2 A*) *ntaken ntaken*
unfolding *rel-fun-def*
by *transfer* (*auto simp add*: *llist-all2-ltakeI*)
declare *ntaken.transfer* [*transfer-rule*]

lemma *nzip-transfer* [*transfer-rule*]:
  (*nellist-all2 A* ===> *nellist-all2 B* ===> *nellist-all2* (*rel-prod A B*)) *nzip nzip*
by (*auto intro*: *nellist-all2-nzipI*)

lemma *niterates-transfer* [*transfer-rule*]:
  ((*A* ===> *A*) ===> *A* ===> *nellist-all2 A*) *niterates niterates*
unfolding *rel-fun-def*
apply *transfer*
using *iterates-transfer* **unfolding** *rel-fun-def*
by *blast*

lemma *nfilter-transfer* [*transfer-rule*]:
  ((*A* ===> (=)) ===> *nellist-all2 A* ===> *nellist-all2 A*) *nfilter nfilter*
unfolding *rel-fun-def*
by *transfer*
   (*auto intro*: *llist-all2-lfilterI dest*: *llist-all2-lfiniteD llist-all2-lsetD1*,
    meson llist-all2-lsetD2)
declare *nfilter.transfer* [*transfer-rule*]

lemma *nconcat-transfer* [*transfer-rule*]:

(*nellist-all2* (*nellist-all2 A*) ===> *nellist-all2 A*) *nconcat nconcat*
unfolding *rel-fun-def*
using *nellist-all2-nconcatI*
by *auto*
declare *nconcat.transfer* [*transfer-rule*]

lemma *nellist-all2-rsp*:
assumes *R1*: $\forall x\ y.\ R1\ x\ y \longrightarrow (\forall a\ b.\ R1\ a\ b \longrightarrow S\ x\ a = T\ y\ b)$
and *R2*: $\forall x\ y.\ R2\ x\ y \longrightarrow (\forall a\ b.\ R2\ a\ b \longrightarrow S'\ x\ a = T'\ y\ b)$
and *xsys*: *nellist-all2 R1 xs ys*
and *xs'ys'*: *nellist-all2 R1 xs' ys'*
shows *nellist-all2 S xs xs'* = *nellist-all2 T ys ys'*
proof
assume *nellist-all2 S xs xs'*
with *xsys xs'ys'* **show** *nellist-all2 T ys ys'*
proof(*coinduction arbitrary*: *ys ys' xs xs'*)
case (*ilist-all2 ys ys' xs xs'*)
thus *?case*
by *cases* (*auto 4 4 simp add*: *nellist-all2-NCons1 nellist-all2-NCons2 nellist-all2-NNil1 nellist-all2-NNil2 dest*: *R1*[*rule-format*] *R2*[*rule-format*])
qed
next
assume *nellist-all2 T ys ys'*
with *xsys xs'ys'* **show** *nellist-all2 S xs xs'*
proof(*coinduction arbitrary*: *xs xs' ys ys'*)
case (*ilist-all2 xs xs' ys ys'*)
thus *?case*
by *cases* (*auto 4 4 simp add*: *nellist-all2-NCons1 nellist-all2-NCons2 nellist-all2-NNil1 nellist-all2-NNil2 dest*: *R1*[*rule-format*] *R2*[*rule-format*])
qed
qed

lemma *nellist-all2-transfer2* [*transfer-rule*]:
((*R1* ===> *R1* ===> (=)) ===>
nellist-all2 R1 ===> *nellist-all2 R1* ===> (=)) *nellist-all2 nellist-all2*
by (*simp add*: *nellist-all2-rsp rel-fun-def*)
declare *nellist-all2-transfer* [*transfer-rule*]

end

Delete lifting rules for *'a nellist* because the parametricity rules take precedence over most of the transfer rules. They can be restored by including the bundle *nellist.lifting*.

lifting-update *nellist.lifting*
lifting-forget *nellist.lifting*

end

3 Extra operations on nellists

The operations ndropns, nkfilter, nleast, nidx, nfuse, nbutlast, nsubn, nfuse, nridx, nlastnfirst and nfusecat are defined for nellists together with a library of lemmas.

theory *NELList-Extras*
 imports *NELList*
begin

3.1 ndropns

primcorec *ndropns* :: *'a nellist* ⇒ *'a nellist nellist*
 where *ndropns nell* =
 (*case nell of* (*NNil b*) ⇒ (*NNil* (*NNil b*)) |
 (*NCons x nell'*) ⇒ (*NCons* (*NCons x nell'*) (*ndropns nell'*)))

simps-of-case *ndropns-code* [*code*, *simp*, *nitpick-simp*]: *ndropns.code*

lemma *ndropns-simps* [*simp*]:
 shows *nhd-ndropns*: ¬ *is-NNil nell* ⟹ *nhd* (*ndropns nell*) = *nell*
 and *ndropns-NCons*: *ntl* (*ndropns nell*) = (*case nell of* (*NNil b*) ⇒ (*NNil* (*NNil b*)) |
 (*NCons x nell'*) ⇒ *ndropns nell'*)
by (*auto simp add*: *nellist.case-eq-if*)
 (*metis ndropns-code*(*1*) *nellist.collapse*(*1*) *nellist.sel*(*4*))

lemma *ndropns-nnth*:
 assumes *i* ≤ *nlength nell*
 shows *nnth* (*ndropns nell*) *i* = *ndropn i nell*
using *assms*
proof (*induction i arbitrary*: *nell*)
case *0*
then show *?case*
 proof (*cases nell*)
 case (*NNil x1*)
 then show *?thesis* **by** (*simp add*: *nnth-NNil*)
 next
 case (*NCons x21 x22*)
 then show *?thesis* **by** *simp*
 qed
next
case (*Suc i*)
then show *?case*
 proof (*cases nell*)
 case (*NNil x1*)
 then show *?thesis* **by** (*simp add*: *nnth-NNil*)
 next
 case (*NCons x21 x22*)
 then show *?thesis* **using** *Suc* **by** (*simp add*: *Suc-ile-eq*)
 qed
qed

lemma *ndropns-nlength*:
  nlength (*ndropns nell*) = (*nlength nell*)
by (*coinduction arbitrary*: *nell rule*: *enat-coinduct*)
  (*case-tac nell*, *auto*)

lemma *in-nset-ndropns*:
  nell ∈ *nset*(*ndropns nellx*) ⟷ (∃ *i*. *i* ≤ *nlength nellx* ∧ *nell* = *ndropn i nellx*)
by (*metis in-nset-conv-nnth ndropns-nlength ndropns-nnth*)

lemma *nset-ndropns*:
  nset (*ndropns nell*) = { *ndropn i nell* | *i*. *i* ≤ *nlength nell*}
using *in-nset-conv-nnth*[*of - ndropns nell*] *nset-conv-nnth*[*of ndropns nell*]
using *in-nset-ndropns*[*of - nell*] **by** *auto*

lemma *nmap-first-ndropns*:
  nmap (λ *nell*. *nnth nell 0*) (*ndropns nell*) = *nell*
by (*coinduction arbitrary*: *nell*)
  (*case-tac nell*, *auto simp add*: *nnth-NNil*)

lemma *ndropn-ndropns*:
 assumes *i* ≤ *nlength*(*ndropns nell*)
 shows *ndropn i* (*ndropns nell*) = *ndropns* (*ndropn i nell*)
using *assms*
proof (*coinduction arbitrary*: *nell i*)
case (*Eq-nellist ia nellx*)
then show *?case*
  proof −
   have *1*: *enat nellx* ≤ *nlength* (*ndropns ia*) ⟹
      is-NNil (*ndropn nellx* (*ndropns ia*)) = *is-NNil* (*ndropns* (*ndropn nellx ia*))
    by (*simp add*: *is-NNil-ndropn ndropns-nlength*)
   have *2*: *enat nellx* ≤ *nlength* (*ndropns ia*) ⟹
      (*is-NNil* (*ndropn nellx* (*ndropns ia*)) ⟶
       is-NNil (*ndropns* (*ndropn nellx ia*)) ⟶
       nlast (*ndropn nellx* (*ndropns ia*)) = *nlast* (*ndropns* (*ndropn nellx ia*)))
    by (*metis dual-order.antisym ndropn-eq-NNil ndropns.disc*(*2*) *ndropns.simps*(*3*) *ndropns-nlength*
     ndropns-nnth nellist.case-eq-if nellist.collapse(*1*) *the-enat.simps*)
   have *3*: *enat nellx* ≤ *nlength* (*ndropns ia*) ⟹
      (¬ *is-NNil* (*ndropn nellx* (*ndropns ia*)) ⟶
       ¬ *is-NNil* (*ndropns* (*ndropn nellx ia*)) ⟶
       nhd (*ndropn nellx* (*ndropns ia*)) = *nhd* (*ndropns* (*ndropn nellx ia*)))
    by (*metis Nat.add-0-right ndropn-nnth ndropns.disc*(*1*) *ndropns-nlength ndropns-nnth nhd-conv-nnth*
     nhd-ndropns)
   have *4*: *enat nellx* ≤ *nlength* (*ndropns ia*) ⟹
      (¬ *is-NNil* (*ndropn nellx* (*ndropns ia*)) ⟶
       ¬ *is-NNil* (*ndropns* (*ndropn nellx ia*)) ⟶
       (∃ *nell i*.
        ntl (*ndropn nellx* (*ndropns ia*)) = *ndropn i* (*ndropns nell*) ∧
        ntl (*ndropns* (*ndropn nellx ia*)) = *ndropns* (*ndropn i nell*) ∧
        enat i ≤ *nlength* (*ndropns nell*)))
    by (*metis Suc-ile-eq is-NNil-ndropn ndropn-Suc-conv-ndropn ndropns-code*(*2*) *ndropns-nlength*

nellist.sel(*5*) *order.order-iff-strict*)
show *?thesis*
using *1 2 3 4 Eq-nellist* **by** *blast*
qed
qed

lemma *ndropns-nfilter-nnth*:
assumes $i \le$ *nlength* (*nfilter P* (*ndropns nell*))
$\exists$ *nelly* $\in$ *nset*(*ndropns nell*). *P nelly*
shows *P* (*nnth* (*nfilter P* (*ndropns nell*)) *i*)
using *assms* **using** *nset-nfilter*[*of ndropns nell P*]
by (*metis* (*full-types*) *Int-iff in-nset-conv-nnth mem-Collect-eq*)

lemma *nnth-zero-ndropn*:
nnth (*ndropn n nell*) *0* = *nnth nell n*
by *simp*

lemma *in-nset-ndropns-nhd*:
x $\in$ *nset nell* $\longleftrightarrow$($\exists$ *ys*. *x*=(*nnth ys 0*) $\wedge$ *ys* $\in$ *nset*(*ndropns nell*))
by *auto*
(*metis in-nset-conv-nnth ndropns-nlength nmap-first-ndropns nnth-nmap*,
metis Nat.add-0-right in-nset-conv-nnth in-nset-ndropns ndropn-nnth)

lemma *nset-ndropns-nhd*:
nset nell = {(*nnth nelly 0*) | *nelly*. *nelly* $\in$ *nset*(*ndropns nell*) }
by *auto*
(*meson in-nset-ndropns-nhd*,
metis in-nset-conv-nnth in-nset-ndropns nnth-zero-ndropn)

lemma *nellist-all2-ndropnsI*:
nellist-all2 A nellx nelly $\Longrightarrow$ *nellist-all2* (*nellist-all2 A*) (*ndropns nellx*) (*ndropns nelly*)
by (*coinduction arbitrary*: *nellx nelly*)
(*auto simp add*: *nellist.case-eq-if dest*: *nellist-all2-nhdD nellist-all2-is-NNilD*
intro: *nellist-all2-ntlI*)

3.2 Definitions

context
includes *nellist.lifting*
begin

lift-definition *nkfilter* :: ($'a \Rightarrow$ *bool*) $\Rightarrow$ *nat* $\Rightarrow'a$ *nellist* $\Rightarrow$ *nat nellist*
is λ *P n xs*. (*if lnull*(*kfilter P n xs*) *then* (*iterates Suc n*) *else kfilter P n xs*)
by *simp*

lift-definition *nleast* :: ($'a \Rightarrow$ *bool*) $\Rightarrow$ $'a$ *nellist* $\Rightarrow$ *nat*
is λ *P xs*. *lleast P xs*
by *auto*

lift-definition *nridx* :: ($'a \Rightarrow$ $'a \Rightarrow$ *bool*) $\Rightarrow$ $'a$ *nellist* $\Rightarrow$ *bool*

is λ *R xs. ridx R xs*
by *auto*

lift-definition *nidx* :: *nat nellist* $\Rightarrow$ *bool*
is λ *xs. lidx xs*
by *auto*

lift-definition *nbutlast* :: *'a nellist* $\Rightarrow$ *'a nellist*
is λ *xs.* (*if lnull* (*lbutlast xs*) *then* (*LCons* (*llast xs*) *LNil*) *else lbutlast xs*)
by *auto*

lift-definition *nfuse* :: *'a nellist* $\Rightarrow$ *'a nellist* $\Rightarrow$ *'a nellist*
is λ *xs ys. lfuse xs ys*
using *lfuse-conv-lnull* **by** *blast*

lift-definition *nsubn* :: *'a nellist* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *'a nellist*
is (λ *xs n1 n2. lsubc n1 n2 xs*)
unfolding *lsubc-def*
by *auto*
(*metis co.enat.exhaust-sel dual-order.refl enat-ile iless-Suc-eq leD llength-eq-0*)

lift-definition *nlastnfirst* :: *'a nellist nellist* $\Rightarrow$ *bool*
is λ *xss. llastlfirst xss*
apply (*simp add*: *pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)
unfolding *llastlfirst-def*
using *llist-all2-llist-of-nellist-1* **by** *blast*

lemma *nfusecat-help*:
assumes $\exists$ *y. llist-all2* (λ*x z. llist-of-nellist z* = *x*) *llist1 y* $\wedge$ *y* $\neq$ *LNil* $\wedge$
llist-all2 (λ*x z. llist-of-nellist z* = *x*) *llist2 y*
shows *lfusecat llist1* $\neq$ *LNil* $\wedge$ *lfusecat llist1* = *lfusecat llist2*
proof –
have *1*: *lfusecat llist1* = *lfusecat llist2*
using *assms llist-all2-llist-of-nellist-1 lnull-def* **by** *blast*
have *2*: *lfusecat llist1* $\neq$ *LNil*
using *assms* **apply** *auto*
using *lfusecat-not-lnull-var llist-all2-lnullD llist-all2-lsetD1* **by** *fastforce*
show *?thesis* **using** *1 2* **by** *auto*
qed

lift-definition *nfusecat* :: *'a nellist nellist* $\Rightarrow$ *'a nellist*
is λ *xss. lfusecat xss*
using *nfusecat-help*
by (*simp add*: *pcr-nellist-def OO-def cr-nellist-def nellist.pcr-cr-eq rel-fun-def lnull-def llist-all2-cases llist-all2-rsp llist.rel-eq not-lnull-conv-llist-of-nellist*) *blast*

3.3 nbutlast

lemma *nbutlast-NNil*[*simp*]:
nbutlast (*NNil x*) = (*NNil x*)

apply *transfer*
by *auto*

lemma *nbutlast-snoc* [*simp*]:
nbutlast (*nappend nell* (*NNil x*)) = *nell*
apply *transfer*
by *auto*

lemma *nbutlast-def1*:
nbutlast nell =
(*case nell of* (*NNil x*) ⇒ (*NNil x*) |
(*NCons x nell1*) ⇒
(*case nell1 of* (*NNil y*) ⇒ (*NNil x*) |
(*NCons z nell2*) ⇒ (*NCons x* (*nbutlast nell1*))))
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: *is-NNil x22* ⟹ *nbutlast* (*NCons x21 x22*) = (*NNil x21*)
by (*metis nappend-NNil nbutlast-snoc nellist.collapse*(*1*))
have *2*: ¬*is-NNil x22* ⟹ *nbutlast* (*NCons x21 x22*) = (*NCons x21* (*nbutlast x22*))
apply *transfer*
by *auto*
(*metis lhd-LCons-ltl llist.collapse*(*1*),
metis lbutlast-simps(*3*) *lhd-LCons-ltl*)
show *?thesis*
by (*simp add*: *1 2 NCons nellist.case-eq-if*)
qed
qed

lemma *ntl-nbutlast*:
ntl (*nbutlast nell*) =
(*if is-NNil nell then ntl nell else*
(*if is-NNil* (*ntl nell*) *then NNil* (*nhd nell*) *else nbutlast* (*ntl nell*)))
by (*auto simp add*: *nbutlast-def1 nellist.case-eq-if*)

lemma *nbutlast-not-nfinite*:
assumes ¬ *nfinite nell*
shows *nbutlast nell* = *nell*
using *assms*
apply *transfer*
by *simp*
(*metis lbutlast.disc-iff*(*2*) *lbutlast-not-lfinite*)

lemma *nbutlast-nfinite*:
nfinite (*nbutlast nell*) ⟷ *nfinite nell*

apply *transfer*
by (*metis* (*full-types*) *lbutlast-lfinite lbutlast-not-lfinite lfinite-LNil lfinite-code*(*2*))

lemma *nlength-nbutlast* [*simp*]:
nlength (*nbutlast nell*) = *epred* (*nlength nell*)
apply *transfer*
by (*simp*, *simp add*: *epred-llength*)

lemma *nbutlast-nappend*:
nbutlast (*nappend nellx nelly*) =
(*if is-NNil nelly then nellx else nappend nellx* (*nbutlast nelly*))
apply *transfer*
by *simp*
(*metis lbutlast-lappend lbutlast-snoc lhd-LCons-ltl lnull-def*)

lemma *nappend-nbutlast-nlast-id-nfinite*:
assumes *nfinite nellx*
¬*is-NNil nellx*
shows (*nappend* (*nbutlast nellx*) (*NNil* (*nlast nellx*))) = *nellx*
using *assms*
apply *transfer*
by *simp*
(*metis lhd-LCons-ltl llist.collapse*(*1*))

lemma *nappend-nbutlast-nlast-id-not-nfinite*:
assumes ¬*nfinite nellx*
¬ *is-NNil nellx*
shows (*nappend* (*nbutlast nellx*) (*NNil* (*nlast nellx*))) = *nellx*
using *assms*
apply *transfer*
by *simp*
(*metis lappend-inf lbutlast.disc-iff*(*2*) *lbutlast-snoc*)

lemma *nappend-nbutlast-nlast-id* [*simp*]:
shows ¬ *is-NNil nell* $\Longrightarrow$ (*nappend* (*nbutlast nell*) (*NNil* (*nlast nell*))) = *nell*
using *nappend-nbutlast-nlast-id-nfinite nappend-nbutlast-nlast-id-not-nfinite* **by** *blast*

lemma *nbutlast-eq-NNil-conv*:
nbutlast nell = (*NNil* (*nfirst nell*)) $\longleftrightarrow$
nell= (*NNil* (*nfirst nell*)) ∨ (∃ *x*. *nell* = (*NCons x* (*NNil* (*nlast nell*))))
apply *transfer*
by (*auto simp add*: *llist.expand lbutlast-not-lfinite*)
(*metis lhd-LCons-ltl llast-LCons*,
metis eq-LConsD lbutlast-eq-LNil-conv lbutlast-ltl lhd-LCons-ltl llast-LCons2 llast-singleton,
simp add: *eq-LConsD*,
simp add: *eq-LConsD lbutlast-eq-LCons-conv*,
metis lfinite-LNil lfinite-ltl llist.collapse(*1*))

lemma *nbutlast-eq-NCons-conv*:
nbutlast nell = (*NCons x ys*) $\longleftrightarrow$

nell = (*NCons x* (*nappend ys* (*NNil* (*nlast nell*))))
apply *transfer*
by (*auto simp add*: *eq-LConsD lbutlast-eq-LCons-conv llast-linfinite*)

lemma *nbutlast-conv-ntake*:
nbutlast nell = *ntake* (*epred* (*nlength nell*)) *nell*
apply *transfer*
by *simp*
(*metis co.enat.exhaust-sel ileI1 iless-eSuc0 lappend-lbutlast-llast-id lappend-lnull1 lbutlast.disc-iff*(*2*) *lbutlast-conv-ltake llength-eq-0 llength-lbutlast ltake-all*)

lemma *nmap-nbutlast*:
nmap f (*nbutlast nell*) = *nbutlast* (*nmap f nell*)
apply *transfer*
by *simp*
(*metis lfinite-ltl llast-lmap lmap-lbutlast lnull-imp-lfinite*)

lemma *snocs-eq-iff-nbutlast*:
nappend nell (*NNil x*) = *nell1* ⟷
((*nfinite nell1* ∧ ¬ *is-NNil nell1* ∧ *nbutlast nell1* = *nell* ∧ *nlast nell1* = *x*)
∨ (¬ *nfinite nell1* ∧ *nbutlast nell* = *nell1*))
by (*metis is-NNil-nappend nappend-inf nappend-nbutlast-nlast-id nbutlast-nfinite nbutlast-snoc nlast-NNil nlast-nappend*)

lemma *in-nset-nbutlastD*:
x ∈ *nset*(*nbutlast nell*) ⟹ *x* ∈ *nset nell*
by (*metis in-nset-snocn-iff nappend-nbutlast-nlast-id-nfinite nappend-snocn nbutlast-NNil nbutlast-not-nfinite nellist.collapse*(*1*))

lemma *in-nset-nbutlast-nappendI*:
x ∈ *nset* (*nbutlast nell*) ∨ (*nfinite nell* ∧ ¬ *is-NNil nell1* ∧ *x* ∈ *nset*(*nbutlast nell1*)) ⟹
x ∈ *nset* (*nbutlast* (*nappend nell nell1*))
unfolding *nbutlast-nappend*
by (*metis* (*full-types*) *Un-iff in-nset-nbutlastD nset-nappend*)

lemma *nnth-nbutlast*:
assumes *n* ≤ *nlength*(*nbutlast nell*)
shows *nnth* (*nbutlast nell*) *n* = *nnth nell n*
by (*metis assms nappend-nbutlast-nlast-id-nfinite nbutlast-eq-NNil-conv nbutlast-not-nfinite ndropn-is-NNil ndropn-nfirst nellist.collapse*(*1*) *nnth-nappend1 nnth-nlast*)

3.4 nsubn

lemma *nsubn-def1*:
nsubn nell i j = *ntaken* (*j*−*i*) (*ndropn i nell*)
apply *transfer*
unfolding *lsubc-def*
by *auto*
(*metis co.enat.exhaust-sel dual-order.refl enat-ile iless-Suc-eq leD llength-eq-0* ,

metis eSuc-enat enat-the-enat infinity-ileE ldrop-enat min.cobounded1)

lemma *nsubn-same*:
shows *nsubn nell k k* = (*NNil* (*nnth nell k*))
unfolding *nsubn-def1*
by (*simp add*: *ndropn-nfirst*)

lemma *nsubn-nlength*:
nlength(*nsubn nell i j*) = *min* (*j*−*i*) (*nlength nell* − *i*)
by (*simp add*: *nsubn-def1*)

lemma *nsubn-nlength-gr-one*:
assumes *k*<*n*
n ≤ *nlength nell*
shows *0*< *nlength* (*nsubn nell k n*)
using *assms*
unfolding *nsubn-nlength*
by (*metis enat-minus-mono1 enat-ord-simps*(*2*) *idiff-enat-enat min-def zero-enat-def zero-less-diff*)

lemma *nsubn-nfinite*:
shows *nfinite* (*nsubn nell k n*)
by (*simp add*: *nsubn-def1*)

lemma *nsubn-nnth*:
shows *nnth* (*nsubn nell i j*) *k* = *nnth nell* (*i* + *min k* (*j* − *i*))
unfolding *nsubn-def1*
using *ntaken-nnth*[*of j*−*i* (*ndropn i nell*) *k*] *ndropn-nnth*[*of i nell* (*min k* (*j* − *i*))]
by *simp*

lemma *ntaken-ndropn*:
ntaken n (*ndropn k nell*) = *nsubn nell k* (*n*+*k*)
by (*simp add*: *nsubn-def1*)

lemma *ntaken-ndropn-nfirst*:
nfirst (*ntaken n* (*ndropn k nell*)) = *nnth nell k*
by (*metis min-0L ndropn-nfirst ntaken-0 ntaken-ntaken nlast-NNil*)

lemma *ntaken-ndropn-nfirst-a*:
nfirst (*ntaken n* (*ndropn k nell*)) = *nfirst*(*ndropn k nell*)
by (*simp add*: *ndropn-nfirst ntaken-ndropn-nfirst*)

lemma *ntaken-ndropn-nlast*:
nlast(*ntaken n* (*ndropn k nell*)) = *nnth nell* (*n*+*k*)
by (*simp add*: *add.commute ntaken-nlast*)

lemma *nsubn-nfirst*:
nfirst (*nsubn nell i j*) = *nnth nell i*

by (*simp add*: *nsubn-def1 ntaken-ndropn-nfirst*)

lemma *nsubn-nlast*:
  nlast (*nsubn nell i j*) = *nnth nell* (*j* − *i* + *i*)
by (*simp add*: *nsubn-def1 ntaken-ndropn-nlast*)

lemma *nsubn-ndropn*:
 assumes *i*<*j*
 shows *nsubn nell* (*i*+*k*) (*j*+*k*) = *nsubn* (*ndropn k nell*) *i j*
using *assms*
by (*simp add*: *add.commute ndropn-ndropn nsubn-def1*)

lemma *pref-ntaken-3*:
  (*ntaken i* (*ntaken* (*i*+*k*) *nell*)) = (*ntaken i nell*)
by (*metis le-add1 min.orderE ntaken-ntaken*)

lemma *ntaken-ndropn-swap-nlength*:
 assumes *ia*+ *i* ≤ *nlength nell*
 shows *nlength* (*ntaken ia* (*ndropn i nell*)) = *nlength* (*ndropn i* (*ntaken* (*ia*+*i*) *nell*))
using *assms ndropn-nlength*[*of i* (*ntaken* (*ia*+*i*) *nell*)] *ntaken-nlength*[*of ia* (*ndropn i nell*)]
by *auto*
  (*metis add-diff-cancel-right′ enat-minus-mono1 idiff-enat-enat min.orderE*)

lemma *ntaken-ndropn-swap-nnth*:
 assumes *m* ≤ *ia*
   ia+ *i* ≤ *nlength nell*
 shows *nnth* (*ntaken ia* (*ndropn i nell*)) *m* = *nnth* (*ndropn i* (*ntaken* (*ia*+*i*) *nell*)) *m*
using *assms*
by (*simp add*: *ntaken-nnth*)

lemma *nellist-eq-nnth-eq*:
  (*nellx* = *nelly*) ⟷ *nlength nellx* = *nlength nelly* ∧ (∀ *i* ≤ *nlength nellx*. *nnth nellx i* = *nnth nelly i*)
by *transfer*
  (*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llist-eq-lnth-eq min-absorb1 the-enat.simps*)

lemma *ntaken-ndropn-swap*:
 assumes *ia*+ *i* ≤ *nlength nell*
 shows (*ntaken ia* (*ndropn i nell*)) = (*ndropn i* (*ntaken* (*ia*+*i*) *nell*))
using *assms nellist-eq-nnth-eq*[*of* (*ntaken ia* (*ndropn i nell*)) (*ndropn i* (*ntaken* (*ia*+*i*) *nell*))]
using *ntaken-ndropn-swap-nlength*
using *ntaken-ndropn-swap-nnth* **by** *fastforce*

lemma *ntaken-nsubn*:
 assumes *n* ≤ *nlength nell*
   m + *k* ≤ *n*
 shows *ntaken m* (*nsubn nell k n*) = *nsubn nell k* (*m*+*k*)
using *assms*
unfolding *nsubn-def1*

by *simp*

lemma *ndropn-nsubn*:
assumes $n \leq$ *nlength nell*
$m + k \leq n$
shows *ndropn m* (*nsubn nell k n*) = *nsubn nell* (*m*+*k*) *n*
proof −
have *1*: *ntaken* (*n*−*k*) (*ndropn k nell*) = *ndropn k* (*ntaken n nell*)
by (*metis add-leD2 assms*(*1*) *assms*(*2*) *diff-add ntaken-ndropn-swap plus-enat-simps*(*1*))
have *2*: *ndropn m* (*ndropn k* (*ntaken n nell*)) =
ndropn (*m* +*k*) (*ntaken n nell*)
by (*simp add*: *add.commute ndropn-ndropn*)
show *?thesis* **unfolding** *nsubn-def1* **using** *1 2*
by (*simp add*: *assms*(*1*) *assms*(*2*) *ntaken-ndropn-swap*)
qed

lemma *ntl-nsubn*:
assumes $n \leq$ *nlength nell*
$k \leq n$
shows *ntl*(*nsubn nell k n*) = (*if n*=*k then* (*NNil* (*nnth nell k*)) *else nsubn* (*ntl nell*) *k* (*n*−*1*))
using *assms* **unfolding** *nsubn-def1*
using *ntl-ntaken*[*of n*−*k ndropn k nell*] *ntl-ndropn*[*of k nell*]
by (*metis diff-diff-cancel diff-right-commute diff-zero nellist.sel*(*4*) *nsubn-def1 nsubn-same*)

lemma *nsubn-nsubn*:
assumes $n1 \leq n2$
$n0 \leq n4$
$n2 \leq n4 - n0$
$n4 \leq n3$
$n3 \leq$ *nlength nell*
shows (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) = (*nsubn* (*nsubn nell n0 n4*) *n1 n2*)
proof −
have *1*: *nlength*(*nsubn nell n0 n3*) = *n3* −*n0*
using *assms* **by** (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength*)
have *2*: *nlength* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) = *n2* −*n1*
using *assms*
by (*metis Nat.le-diff-conv2 enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat le-trans min.orderE nsubn-nlength*)
have *3*: *nlength*(*nsubn nell n0 n4*) = *n4* −*n0*
using *assms nsubn-nlength*[*of nell n0 n4*]
unfolding *min-def*
by (*metis enat-minus-mono1 idiff-enat-enat min.coboundedI1 min.left-commute min-absorb1 min-enat-simps*(*1*))
have *4*: *nlength* (*nsubn* (*nsubn nell n0 n4*) *n1 n2*) = *n2* −*n1*
using *assms*
by (*metis 3 enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat min.orderE nsubn-nlength*)
have *5*: $\bigwedge i.\ i \leq (n3 - n0) \longrightarrow$ (*nnth* (*nsubn nell n0 n3*) *i*) = (*nnth nell* (*n0*+*i*))
using *assms* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *6*: $\bigwedge i\ .\ i \leq (n2 - n1) \longrightarrow$
(*nnth* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) *i*) = (*nnth* (*nsubn nell n0 n3*) (*n1*+*i*))

using *assms* **by** (*simp add*: *Nat.le-diff-conv2 nsubn-nnth*)
have *7*: $\bigwedge i$. $i \leq (n2-n1) \longrightarrow$
(*nnth* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) *i*) = (*nnth nell* (*n0*+(*n1*+*i*)))
using *5 6 assms* **by** *auto*
have *8*: $n0 \leq n4 \wedge n4 \leq$ *nlength nell*
using *assms* **by** (*simp add*: *order-subst2*)
have *9*: $\bigwedge i$. $i \leq (n4 - n0) \longrightarrow$ (*nnth* (*nsubn nell n0 n4*) *i*) = (*nnth nell* (*n0*+*i*))
using *8* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *10*: $\bigwedge i$. $i \leq (n2-n1) \longrightarrow$
(*nnth* (*nsubn* (*nsubn nell n0 n4*) *n1 n2*) *i*) = (*nnth* (*nsubn nell n0 n4*) (*n1*+*i*))
by (*simp add*: *nsubn-def1 ntaken-nnth*)
have *11*: $\bigwedge i$. $i \leq (n2-n1) \longrightarrow$
(*nnth* (*nsubn* (*nsubn nell n0 n4*) *n1 n2*) *i*) = (*nnth nell* (*n0* +(*n1*+*i*)))
by (*metis 10 9 Nat.le-diff-conv2 add.commute assms*(*1*) *assms*(*3*) *le-trans*)
have *12*: $\bigwedge i$. $i \leq (n2-n1) \longrightarrow$
(*nnth* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) *i*) = (*nnth* (*nsubn* (*nsubn nell n0 n4*) *n1 n2*) *i*)
by (*simp add*: *11 7*)
from *12 2 4* **show** *?thesis*
using *nellist-eq-nnth-eq*[*of* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) (*nsubn* (*nsubn nell n0 n4*) *n1 n2*)]
enat-ord-simps(*1*) **by** *presburger*
qed

lemma *nsubn-nsubn-1*:
assumes $n1 \leq n2$
$n0 \leq n3$
$n2 \leq n3-n0$
$n3 \leq$ *nlength nell*
shows (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) = (*nsubn nell* (*n0*+*n1*) (*n0*+*n2*))
proof −
have *0*: *nlength*(*nsubn* (*nsubn nell n0 n3*) *n1 n2*) = *n2* − *n1*
using *assms*
by (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength of-nat-eq-enat of-nat-mono*)
have *1*: *nlength*(*nsubn nell* (*n1*+*n0*) (*n2*+*n0*)) = (*n2*+*n0*) −(*n1*+*n0*)
using *assms nsubn-nlength*[*of nell* (*n1*+*n0*) (*n2*+*n0*)]
unfolding *min-def*
by (*metis Nat.le-diff-conv2 dual-order.trans enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat*)
have *10*: (*n2*+*n0*) −(*n1*+*n0*) = *n2* −*n1*
using *diff-cancel2* **by** *blast*
have *2*: $\bigwedge i$. $i \leq (n2-n1) \longrightarrow$
(*nnth* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) *i*) = (*nnth nell* (*n0*+(*n1*+*i*)))
using *assms* **by** (*simp add*: *nsubn-nnth*)
have *3*: $\bigwedge i$. $i \leq (n2-n1) \longrightarrow$
(*nnth* (*nsubn nell* (*n0*+*n1*) (*n0*+*n2*)) *i*) = (*nnth nell* (*n0*+(*n1*+*i*)))
using *assms* **by** (*metis 10 add.assoc add.commute min.orderE nsubn-nnth*)
have *4*: $\bigwedge i$. $i \leq (n2-n1) \longrightarrow$
(*nnth* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*) *i*) = (*nnth* (*nsubn nell* (*n0*+*n1*) (*n0*+*n2*)) *i*)
by (*simp add*: *2 3*)
show *?thesis* **using** *nellist-eq-nnth-eq*[*of* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*)
(*nsubn nell* (*n0*+*n1*) (*n0*+*n2*))]
0 10 4 1

by (*metis add.commute enat-ord-simps*(*1*))
qed

lemma *nsubn-eq*:
assumes $n \le m$
$m \le$ *nlength nellx*
$m \le$ *nlength nelly*
$\forall j.\ n \le j \wedge j \le m \longrightarrow$ *nnth nellx j* = *nnth nelly j*
shows *nsubn nellx n m* = *nsubn nelly n m*
proof −
have *1*: *nlength* (*nsubn nellx n m*) = *nlength* (*nsubn nelly n m*)
using *assms* **by** (*metis enat-minus-mono1 min-def nsubn-nlength*)
have *2*: $\bigwedge j.\ j \le$ *nlength* (*nsubn nellx n m*) $\longrightarrow$ *nnth* (*nsubn nellx n m*) *j* = *nnth* (*nsubn nelly n m*) *j*
using *assms* **by** (*simp add*: *Nat.le-diff-conv2 nsubn-nlength nsubn-nnth*)
show *?thesis*
by (*simp add*: *1 2 nellist-eq-nnth-eq*)
qed

3.5 nfuse

lemma *nfuse-def1*:
nfuse nellx nelly = (*if is-NNil nelly then nellx else nappend nellx* (*ntl nelly*))
apply *transfer*
unfolding *lfuse-def* **by** *simp force*

lemma *nfuse-NCons-a*:
nfuse (*NCons x nellx*) *nelly* = (*NCons x* (*nfuse nellx nelly*))
by (*simp add*: *nfuse-def1*)

lemma *nfuse-NCons-b*:
nfuse nellx (*NCons y nelly*) = *nappend nellx nelly*
by (*simp add*: *nfuse-def1*)

lemma *nfuse-simps* [*simp*]:
shows *nhd-nfuse*: *nhd*(*nfuse nellx nelly*) =
(*if is-NNil nellx then*
(*if is-NNil nelly then nhd nellx else nlast nellx*)
else nhd nellx)

and *ntl-nfuse*: *ntl*(*nfuse nellx nelly*) =
(*if is-NNil nellx then*
(*if is-NNil nelly then ntl nellx else ntl nelly*)
else (*if is-NNil nelly then ntl nellx*
else nappend (*ntl nellx*) (*ntl nelly*)))
by (*simp-all add*: *nfuse-def1*)

lemma *nfuse-nbutlast*:
assumes *nlast nellx* = *nfirst nelly*
$\neg$ *is-NNil nellx*
$\neg$ *is-NNil nelly*

shows *nfuse nellx nelly* = *nappend* (*nbutlast nellx*) *nelly*
using *assms*
by (*metis nappend-NCons nappend-NNil nappend-assoc nappend-nbutlast-nlast-id nappend-snocn nellist.collapse*(*2*) *nellist.sel*(*3*) *nfuse-def1 nhd-snocn*)

lemma *nfuse-nlength*:
shows *nlength* (*nfuse nellx nelly*) = (*nlength nellx*) + (*nlength nelly*)
unfolding *nfuse-def1*
by(*cases nelly*) (*simp*, *simp add*: *eSuc-plus-1*)

lemma *nfuse-nnth*:
assumes *i*≤ *nlength* (*nfuse nellx nelly*)
nlast nellx = *nfirst nelly*
shows (*i* ≤ *nlength nellx* ⟶ *nnth* (*nfuse nellx nelly*) *i* = *nnth nellx i*)
∧
(*nlength nellx* < *i* ∧ *i*≤ *nlength* (*nfuse nellx nelly*) ⟶
nnth (*nfuse nellx nelly*) *i* = *nnth nelly* (*i* − (*the-enat* (*nlength nellx*))))
unfolding *nfuse-def1*
by (*cases nelly*)
(*auto simp add*: *nnth-nappend*,
metis Suc-diff-Suc enat-iless enat-ord-simps(*2*) *ndropn-Suc-NCons ndropn-nfirst the-enat.simps*)

lemma *nfuse-nnth-a*:
assumes *j* ≤ *nlength nelly*
nlast nellx = *nfirst nelly*
nfinite nellx
shows *nnth* (*nfuse nellx nelly*) ((*the-enat*(*nlength nellx*)) +*j*) = (*nnth nelly j*)
using *assms* **unfolding** *nfuse-def1*
by (*cases is-NNil nelly*)
(*simp-all*,
metis assms(*2*) *assms*(*3*) *is-NNil-def is-NNil-imp-nfinite ndropn-nlast ndropn-nfirst ndropn-nnth nnth-NNil*,
metis enat-le-plus-same(*2*) *gen-nlength-def nappend-NNil ndropn-nlast ndropn-nappend2 ndropn-nnth nellist.case-eq-if nellist.collapse*(*2*) *nfinite-nlength-enat nfirst-def nlength-code the-enat.simps*)

lemma *nfuse-nappend*:
assumes *nlast nellx* = *nfirst nelly*
shows *nfuse nellx nelly* =
(*if nfinite nellx then*
(*if is-NNil nellx then nelly else nappend* (*ntaken* (*the-enat*(*epred*(*nlength nellx*))) *nellx*) *nelly*)
else nellx)
proof (*cases nelly*)
case (*NNil x1*)
then show *?thesis*
proof (*cases nfinite nellx*)
case *True*
then show *?thesis*
proof (*cases nellx*)
case (*NNil x1*)

then show *?thesis* **using** *assms*
by (*simp add*: *nfuse-def1*)
(*metis nappend-snocn nellist.collapse*(*1*) *nellist.collapse*(*2*) *nhd-nappend nhd-snocn*)
next
case (*NCons x21 x22*)
then show *?thesis* **unfolding** *nfuse-def1* **using** *assms NNil NCons True*
by (*auto simp add*: *nfirst-def*)
(*metis True add-diff-cancel-left′ assms eSuc-enat ndropn-nfirst ndropn-nlast nellist-split-2-last nfinite-nlength-enat nlast-NCons nlength-NCons plus-1-eq-Suc the-enat.simps zero-less-Suc*)
qed
next
case *False*
then show *?thesis*
by (*simp add*: *NNil nfuse-def1*)
qed
next
case (*NCons x21 x22*)
then show *?thesis*
proof (*cases nfinite nellx*)
case *True*
then show *?thesis*
unfolding *nfuse-def1*
proof *auto*
show *is-NNil nelly* $\Longrightarrow$ *is-NNil nellx* $\Longrightarrow$ *nellx* = *nelly*
by (*simp add*: *NCons*)
show *nfinite nellx* $\Longrightarrow$ *is-NNil nelly* $\Longrightarrow$ ¬ *is-NNil nellx* $\Longrightarrow$
nellx = *nappend* (*ntaken* (*the-enat* (*epred* (*nlength nellx*))) *nellx*) *nelly*
using *assms NCons True* **by** *simp*
show ¬ *is-NNil nelly* $\Longrightarrow$ *is-NNil nellx* $\Longrightarrow$ *nappend nellx* (*ntl nelly*) = *nelly*
using *assms NCons True*
by (*metis nappend-NNil nellist.collapse*(*1*) *nellist.sel*(*5*) *nnth-0 ntaken-0 ntaken-nlast*)
show *nfinite nellx* $\Longrightarrow$ ¬ *is-NNil nelly* $\Longrightarrow$ ¬ *is-NNil nellx* $\Longrightarrow$
nappend nellx (*ntl nelly*) = *nappend* (*ntaken* (*the-enat* (*epred* (*nlength nellx*))) *nellx*) *nelly*
using *assms NCons True*
by (*cases nellx*)
(*auto*,
metis True add-diff-cancel-left′ eSuc-enat nappend-NCons nappend-NNil nappend-assoc ndropn-eq-NNil ndropn-nlast nellist.sel(*5*) *nellist-split-2-last nfinite-nlength-enat nlast-NNil nlast-ntl nlength-NCons nnth-0 ntaken-nlast ntl-ntaken-0 plus-1-eq-Suc the-enat.simps zero-less-Suc*)
qed
next
case *False*
then show *?thesis*
by (*simp add*: *nappend-inf nfuse-def1*)
qed
qed

lemma *nfuse-leftneutral* :

nfuse (*NNil* (*nfirst nell*)) *nell* = *nell*
by (*simp add*: *nfuse-nappend*)

lemma *nfuse-rightneutral* :
nfuse nell (*NNil* (*nlast nell*)) = *nell*
unfolding *nfuse-def*
by *simp*

lemma *nfirst-nfuse* :
assumes *nlast nellx* = *nfirst nelly*
shows *nfirst* (*nfuse nellx nelly*) = *nfirst nellx*
using *assms* **unfolding** *nfuse-def1* **by** *transfer auto*

lemma *nlast-nfuse* :
assumes *nlast nellx* = *nfirst nelly*
nfinite nellx
shows *nlast* (*nfuse nellx nelly*) = *nlast nelly*
using *assms* **unfolding** *nfuse-def1* **by** *transfer* (*auto*, *metis lhd-LCons-ltl llast-LCons llast-lappend*)

lemma *nfuseassoc* :
shows (*nfuse nellx* (*nfuse nelly nellz*)) = (*nfuse* (*nfuse nellx nelly*) *nellz*)
unfolding *nfuse-def1*
by *transfer* (*auto simp add*: *lappend-assoc*, *simp add*: *lappend-ltl*)

lemma *ntaken-nfuse* :
assumes *nlast nellx* = *nfirst nelly*
nfinite nellx
shows (*ntaken* (*the-enat* (*nlength nellx*)) (*nfuse nellx nelly*)) = *nellx*
proof (*cases is-NNil nelly*)
case *True*
then show *?thesis* **using** *assms* **by** (*metis ndropn-eq-NNil ndropn-nlast nfuse-def1 ntaken-all*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis add.left-neutral enat-le-plus-same*(*2*) *nfinite-nlength-enat nfuse-def1 ntaken-all ntaken-nappend1 the-enat.simps*)
qed

lemma *ndropn-nfuse* :
assumes *nlast nellx* = *nfirst nelly*
nfinite nellx
shows (*ndropn* (*the-enat* (*nlength nellx*)) (*nfuse nellx nelly*)) = *nelly*
proof (*cases is-NNil nelly*)
case *True*
then show *?thesis* **using** *assms* **by** (*metis ndropn-nlast nfuse-def1 nfuse-leftneutral*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis enat-le-plus-same*(*2*) *gen-nlength-def ndropn-nappend2 ndropn-nlast nfinite-nlength-enat nfuse-def1 nfuse-leftneutral nlength-code the-enat.simps*)

qed

lemma *nfuse-ntaken-ndropn-nlength* :
assumes $n \leq$ *nlength nell*
shows *nlength* (*nfuse* (*ntaken n nell*) (*ndropn n nell*)) = *nlength nell*
using *assms*
by (*metis dual-order.order-iff-strict enat-add-sub-same infinity-ileE less-eqE min.absorb1 ndropn-nlength nfuse-nlength ntaken-nlength*)

lemma *nfuse-ntaken-ndropn-nnth* :
assumes $n \leq$ *nlength nell*
$i \leq$ *nlength nell*
shows *nnth* (*nfuse* (*ntaken n nell*) (*ndropn n nell*)) *i* = *nnth nell i*
using *assms*
nfuse-nnth[*of i* (*ntaken n nell*) (*ndropn n nell*)]
nfuse-ntaken-ndropn-nlength[*of n nell*]
by (*metis dual-order.strict-iff-order enat-ord-simps*(*1*) *le-add-diff-inverse min.orderE ndropn-nfirst ndropn-nnth not-less ntaken-nlast ntaken-nlength ntaken-nnth the-enat.simps*)

lemma *nfuse-ntaken-ndropn*:
assumes $n\leq$ *nlength nell*
shows *nfuse* (*ntaken n nell*) (*ndropn n nell*) = *nell*
using *assms*
by (*simp add*: *nfuse-ntaken-ndropn-nlength nfuse-ntaken-ndropn-nnth nellist-eq-nnth-eq*)

lemma *nfuse-nnth-var*:
assumes *enat* $i \leq$ *nlength* (*nfuse nellx nelly*)
nlast nellx = *nfirst nelly*
shows (*enat* $i \leq$ *nlength nellx* $\longrightarrow$ *nnth* (*nfuse nellx nelly*) *i* = *nnth nellx i*) $\wedge$
(*nlength nellx* $\leq$ *enat i* $\wedge$ *enat* $i \leq$ *nlength* (*nfuse nellx nelly*) $\longrightarrow$
nnth (*nfuse nellx nelly*) *i* = *nnth nelly* ($i -$ *the-enat* (*nlength nellx*)))
using *nfuse-nnth assms*
by (*metis cancel-comm-monoid-add-class.diff-cancel dual-order.strict-iff-order ndropn-nfuse nlength-eq-enat-nfiniteD nnth-zero-ndropn the-enat.simps*)

lemma *nset-nfuse*:
nset (*nfuse nellx nelly*) =
(*if nfinite nellx then*
(*if is-NNil nelly then nset nellx else nset nellx* $\cup$ *nset* (*ntl nelly*))
else nset nellx)
by (*simp add*: *nfuse-def1 nset-nappend*)

lemma *nsubn-nfuse*:
assumes (*enat k*) $\leq n$
(*enat n*) $\leq m$
(*enat m*) $\leq$ *nlength nell*
shows *nfuse* (*nsubn nell k n*) (*nsubn nell n m*) = (*nsubn nell k m*)
using *assms*
proof –
have *1*: *nlast*(*nsubn nell k n*) = (*nnth nell n*)

by (*metis assms*(*1*) *enat-ord-simps*(*1*) *le-add-diff-inverse2 nsubn-def1 ntaken-ndropn-nlast*)
have *2*: *nfirst*(*nsubn nell n m*) = (*nnth nell n*)
by (*simp add*: *ndropn-nfirst nsubn-def1 ntaken-ndropn-nfirst-a*)
have *3*: *nlength* (*nsubn nell k n*) = *n* −*k*
using *assms nsubn-nlength*[*of nell k n*]
by (*metis dual-order.trans enat-minus-mono1 idiff-enat-enat min-absorb1*)
have *4*: *nlength* (*nsubn nell n m*) = *m* −*n*
by (*metis assms*(*3*) *enat-minus-mono1 idiff-enat-enat min-def nsubn-nlength*)
have *5*: *nlength* (*nsubn nell k m*) = *m* −*k*
by (*metis assms*(*3*) *enat-minus-mono1 idiff-enat-enat min-absorb1 nsubn-nlength*)
have *6*: *nlength*(*nfuse* (*nsubn nell k n*) (*nsubn nell n m*)) = *nlength*(*nsubn nell k m*)
unfolding *nsubn-def*
by (*metis 3 4 5 Nat.add-diff-assoc2 assms*(*1*) *assms*(*2*) *enat-ord-simps*(*1*) *nfuse-nlength nsubn-def ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-enat-simps*(*1*))
have *7*: (∀ *i*. *i* ≤ *nlength*(*nsubn nell k m*) ⟶
(*nnth* (*nfuse* (*nsubn nell k n*) (*nsubn nell n m*)) *i*) = (*nnth nell* (*k*+*i*)))
proof
fix *i*
show *i* ≤ *nlength*(*nsubn nell k m*) ⟶
(*nnth* (*nfuse* (*nsubn nell k n*) (*nsubn nell n m*)) *i*) = (*nnth nell* (*k*+*i*))
proof −
have *41*: *nlength* (*nsubn nell k m*) = (*m*−*k*)
using *5* **by** *blast*
have *42*: *i* ≤ *nlength* (*nsubn nell k m*) ⟶ *nnth* (*nfuse* (*nsubn nell k n*) (*nsubn nell n m*)) *i* =
(*if i* ≤ *n* −*k then* (*nnth* (*nsubn nell k n*) *i*)
else (*nnth* (*nsubn nell n m*) (*i*−(*n*−*k*))))
by (*simp add*: *1 2 3 6 nfuse-nnth*)
have *43*: *i* ≤ (*m*−*k*) ⟶ *nnth* (*nfuse* (*nsubn nell k n*) (*nsubn nell n m*)) *i* =
(*if i*≤ (*n*−*k*) *then* (*nnth nell* (*k*+*i*)) *else* (*nnth nell* (*n*+(*i*−(*n*−*k*)))))
using *assms 42 5* **unfolding** *nsubn-def1*
by (*auto simp add*: *ntaken-nnth*)
(*metis enat-ord-simps*(*1*) *min.bounded-iff*,
metis enat-ord-simps(*1*) *min.bounded-iff*)
have *44*: *i* ≤ (*m*−*k*) ⟶ *nnth* (*nfuse* (*nsubn nell k n*) (*nsubn nell n m*)) *i* =
(*nnth nell* (*k*+*i*))
by (*metis 43 Nat.diff-diff-right Nat.le-diff-conv2 add.commute assms*(*1*) *enat-ord-simps*(*1*) *le-add-diff-inverse nat-le-linear*)
show *?thesis*
by (*simp add*: *41 44*)
qed
qed
have *8*: (∀ *i*. *i* ≤ *nlength*(*nsubn nell k m*) ⟶ (*nnth* (*nsubn nell k m*) *i*) = (*nnth nell* (*k*+*i*)))
using *assms* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
show *?thesis*
by (*simp add*: *6 7 8 nellist-eq-nnth-eq*)
qed

lemma *nmap-nfuse*:
nmap f (*nfuse nellx nelly*) = *nfuse* (*nmap f nellx*) (*nmap f nelly*)
by (*simp add*: *nfuse-def1 nmap-nappend-distrib*)

3.6 nridx and nidx

lemma *nridx-nidx*:
nridx (<) *nell* = *nidx nell*
apply *transfer*
by (*simp add*: *ridx-lidx*)

lemma *nridx-expand*:
nridx R nell ⟷ (∀ *i*. (*Suc i*) ≤ *nlength nell* ⟶ *R* (*nnth nell i*) (*nnth nell* (*Suc i*)))
by *transfer*
(*auto simp add*: *min-def Suc-ile-eq*,
metis co.enat.exhaust-sel eSuc-enat ileI1 iless-Suc-eq llength-eq-0 ridx-def,
metis Extended-Nat.eSuc-mono co.enat.exhaust-sel eSuc-enat llength-eq-0 order-le-less ridx-def)

lemma *nidx-expand*:
nidx nell ⟷ (∀ *i*. (*Suc i*) ≤ *nlength nell* ⟶ *nnth nell i* < *nnth nell* (*Suc i*))
using *nridx-nidx nridx-expand* **by** *blast*

lemma *nridx-NCons*:
nridx R (*NCons x nell*) ⟷
(∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*)))
using *nridx-expand*[*of R* (*NCons x nell*)]
by *auto*

lemma *nidx-NCons*:
nidx (*NCons x nell*) ⟷
(∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ (*nnth* (*NCons x nell*) *n*) < (*nnth* (*NCons x nell*) (*Suc n*)))
using *nridx-NCons*[*of* (<) *x nell*] *nridx-nidx* **by** *blast*

lemma *nridx-LCons-conv*:
(∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*))) ⟷
R x (*nfirst nell*) ∧
(∀ *n*. *0*≤*n* ∧ (*Suc n*) ≤ (*nlength nell*) ⟶ *R* (*nnth nell n*) (*nnth nell* (*Suc n*)))
proof −
have *1*: (∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*))) ⟷
(∀ *n*. *0*≤*n* ∧ (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*)))
by *blast*
have *2*: (∀ *n*. *0*≤*n* ∧ (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*))) ⟷
R x (*nfirst nell*) ∧
(∀ *n*. *1*≤*n* ∧ (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*)))
by (*metis 1 One-nat-def add-diff-cancel-left′ eSuc-ile-mono enat-le-plus-same*(*1*) *gen-nlength-def le-SucE le-add1*
ndropn-Suc-NCons ndropn-nfirst ndropn-nfuse nfuse-leftneutral nlast-NNil nlength-NNil nlength-code
nlength-eq-enat-nfiniteD nnth-0 one-eSuc one-enat-def plus-1-eq-Suc the-enat-0 zero-enat-def)
have *3*: *R x* (*nfirst nell*) ∧
(∀ *n*. *1*≤*n* ∧ (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*)

(*Suc n*))) ⟷
R x (*nfirst nell*) ∧
(∀ *n*. *1*≤*n* ∧ (*n*) ≤ (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*)))
by (*metis eSuc-enat eSuc-ile-mono*)
have *4*: *R x* (*nfirst nell*) ∧
(∀ *n*. *1*≤*n* ∧ (*n*) ≤ (*nlength nell*) ⟶ *R* (*nnth* (*NCons x nell*) *n*) (*nnth* (*NCons x nell*) (*Suc n*)))
⟷
R x (*nfirst nell*) ∧
(∀ *n*. *0*≤(*n*−*1*) ∧ (*Suc*(*n*−*1*)) ≤ (*nlength nell*) ⟶ *R* (*nnth nell* (*n*−*1*)) (*nnth nell* (*Suc* (*n*−*1*))))
by (*metis add.commute add.right-neutral diff-add le-add1 nnth-Suc-NCons plus-1-eq-Suc*)
have *5*: *R x* (*nfirst nell*) ∧
(∀ *n*. *0*≤(*n*−*1*) ∧ (*Suc*(*n*−*1*)) ≤ (*nlength nell*) ⟶ *R* (*nnth nell* (*n*−*1*)) (*nnth nell* (*Suc* (*n*−*1*))))
⟷
R x (*nfirst nell*) ∧
(∀ *n*. *0*≤*n* ∧ (*Suc n*) ≤ (*nlength nell*) ⟶ *R* (*nnth nell n*) (*nnth nell* (*Suc n*)))
by (*metis diff-Suc-1*)
show *?thesis*
using *1 2 3 4 5* **by** *presburger*
qed

lemma *nidx-LCons-conv*:
(∀ *n*. (*Suc n*) ≤ *eSuc* (*nlength nell*) ⟶ (*nnth* (*NCons x nell*) *n*) < (*nnth* (*NCons x nell*) (*Suc n*))) ⟷
x < (*nfirst nell*) ∧
(∀ *n*. *0*≤*n* ∧ (*Suc n*) ≤ (*nlength nell*) ⟶ (*nnth nell n*) < (*nnth nell* (*Suc n*)))
by (*metis nridx-LCons-conv*)

lemma *nridx-LCons-1* [*simp*]:
nridx R (*NCons x nell*) ⟷ (*R x* (*nfirst nell*) ∧ *nridx R nell*)
by (*metis nridx-LCons-conv nridx-NCons nridx-expand zero-le*)

lemma *nidx-LCons-1* [*simp*]:
nidx (*NCons x nell*) ⟷ (*x* < (*nfirst nell*) ∧ *nidx nell*)
by (*metis nridx-LCons-1 nridx-nidx*)

lemma *nridx-less*:
assumes *nridx R nell*
Suc(*n*+*k*) ≤ *nlength nell*
transp R
shows *R* (*nnth nell n*) (*nnth nell* (*Suc*(*n*+*k*)))
using *assms*
proof (*induct k*)
case *0*
then show *?case*
by (*simp add*: *nridx-expand*)
next
case (*Suc k*)
then show *?case*
by (*metis add-Suc-right dual-order.trans eSuc-enat ile-eSuc nridx-expand transpE*)
qed

lemma *nidx-less*:
assumes *nidx nell*
Suc(*n*+*k*) ≤ *nlength nell*
shows *nnth nell n* < *nnth nell* (*Suc*(*n*+*k*))
using *assms*
by (*simp add*: *nridx-less nridx-nidx*)

lemma *nridx-less-eq*:
assumes *nridx R nell*
k ≤ *j*
j ≤ *nlength nell*
transp R
reflp R
shows *R* (*nnth nell k*) (*nnth nell j*)
proof (*cases k*=*j*)
case *True*
then show *?thesis* **using** *assms* **by** (*meson reflpE*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis* (*full-types*) *Suc-diff-Suc add.left-commute le-add-diff-inverse nridx-less order-neq-le-trans plus-1-eq-Suc*)
qed

lemma *nidx-less-eq*:
assumes *nidx nell*
k ≤ *j*
j ≤ *nlength nell*
shows (*nnth nell k*) ≤ (*nnth nell j*)
using *assms*
by (*metis Orderings.order-eq-iff antisym-conv2 less-iff-Suc-add nidx-less order.strict-implies-order*)

lemma *nridx-less-last*:
assumes *nridx R nell*
Suc i < *k*
nlength nell = (*enat k*)
transp R
shows *R* (*nnth nell i*) (*nnth nell* (*k*−*1*))
using *assms less-imp-Suc-add nridx-less* **by** *fastforce*

lemma *nidx-less-last*:
assumes *nidx nell*
Suc i < *k*
nlength nell = (*enat k*)
shows *nnth nell i* < *nnth nell* (*k*−*1*)
using *assms less-imp-Suc-add nidx-less* **by** *fastforce*

lemma *nidx-less-last-1*:
assumes *nidx nell*
i<*nlength nell*

nlength nell = (*enat k*)
shows *nnth nell i* < *nnth nell* (*k*)
using *assms*
by (*metis enat-ord-simps*(*2*) *less-imp-Suc-add linorder-le-cases nridx-less nridx-nidx transp-less*)

lemma *nridx-gr-first*:
assumes *nridx R nell*
0<*i*
i ≤ *nlength nell*
transp R
shows *R* (*nnth nell 0*) (*nnth nell i*)
using *assms nridx-less*[*of R nell 0 i*−*1*] **by** *simp*

lemma *nidx-gr-first*:
assumes *nidx nell*
0<*i*
i ≤ *nlength nell*
shows (*nnth nell 0*) < *nnth nell i*
using *assms nidx-less*[*of nell 0 i*−*1*]
by *simp*

lemma *nridx-ntake-a*:
assumes *nridx R nell*
n ≤ *nlength nell*
shows *nridx R* (*ntake n nell*)
using *assms*
by *transfer*
(*metis co.enat.exhaust-sel eSuc-ile-mono llength-eq-0 ridx-ltake-a*)

lemma *nidx-ntake-a*:
assumes *nidx nell*
n ≤ *nlength nell*
shows *nidx* (*ntake n nell*)
using *assms*
using *nridx-ntake-a nridx-nidx* **by** *blast*

lemma *nridx-nappend-nfinite*:
assumes *nfinite nell1*
shows *nridx R* (*nappend nell1 nell2*) ⟷
nridx R nell1 ∧ (*R* (*nlast nell1*) (*nfirst nell2*)) ∧ *nridx R nell2*
using *assms*
by *transfer*
(*simp add*: *ridx-lappend-lfinite*)

lemma *nidx-nappend-nfinite*:
assumes *nfinite nell1*
shows *nidx* (*nappend nell1 nell2*) ⟷
nidx nell1 ∧ ((*nlast nell1*) < (*nfirst nell2*)) ∧ *nidx nell2*

using *assms*
by (*metis nridx-nappend-nfinite nridx-nidx*)

lemma *nidx-nfuse*:
 assumes *nfinite nell1*
 nidx nell1
 nnth nell1 0 = 0
 nidx nell2
 nnth nell2 0 = nlast nell1
 shows *nidx* (*nfuse nell1 nell2*)
using *assms*
proof (*cases is-NNil nell2*)
case *True*
then show *?thesis* **unfolding** *nfuse-def1*
by (*simp add*: *assms*(*2*))
next
case *False*
then show *?thesis*
 proof −
 have *1*: *nlast nell1 = nfirst nell2*
 by (*metis assms*(*5*) *ndropn-0 ndropn-nfirst*)
 have *2*: *nidx* (*ntl nell2*)
 by (*metis False assms*(*4*) *eSuc-enat ileI1 iless-Suc-eq nellist.collapse*(*2*) *nidx-expand*
 nlength-NCons nnth-ntl)
 have *3*: *nlast nell1 < nfirst*(*ntl nell2*)
 by (*metis False assms*(*4*) *assms*(*5*) *eSuc-enat ileI1 iless-Suc-eq ndropn-0 ndropn-nfirst*
 nellist.collapse(*2*) *nhd-conv-nnth nidx-expand nlength-NCons nnth-ntl zero-enat-def zero-le*)
 have *4*: *nidx* (*nappend nell1* (*ntl nell2*))
 by (*simp add*: *2 3 assms*(*1*) *assms*(*2*) *nidx-nappend-nfinite*)
 show *?thesis* **using** *False* **unfolding** *nfuse-def1*
 using *4* **by** *auto*
 qed
qed

lemma *nridx-ndropn*:
 assumes *nridx R nell*
 n ≤ nlength nell
 shows *nridx R* (*ndropn n nell*)
using *assms*
by *transfer*
 (*metis co.enat.exhaust-sel iless-Suc-eq ldrop-enat llength-eq-0 min.orderE nless-le*
 ridx-ldrop the-enat.simps)

lemma *nidx-ndropn*:
 assumes *nidx nell*
 n ≤ nlength nell
 shows *nidx* (*ndropn n nell*)
using *assms*
using *nridx-ndropn nridx-nidx* **by** *blast*

lemma *nridx-ntake-all*:
 assumes $\bigwedge n.\ n \leq$ *nlength nell* $\Longrightarrow$ *nridx R* (*ntake* (*enat n*) *nell*)
 shows *nridx R nell*
using *assms*
by (*auto simp add*: *nridx-expand*)
 (*metis Suc-ile-eq linorder-le-cases ntake-nnth ntake-nnth order-less-imp-le*)

lemma *nidx-ntake-all*:
 assumes $\bigwedge n.\ n \leq$ *nlength nell* $\Longrightarrow$ *nidx* (*ntake* (*enat n*) *nell*)
 shows *nidx nell*
using *assms*
using *nridx-nidx nridx-ntake-all* **by** *blast*

lemma *nridx-ntake*:
 assumes *nridx R* (*ntake n nell*)
 $n \leq$ *nlength nell*
 $k \leq n$
 shows *nridx R* (*ntake* (*enat k*) *nell*)
using *assms*
using *nridx-ntake-a* **by** *fastforce*

lemma *nidx-ntake*:
 assumes *nidx* (*ntake n nell*)
 $n \leq$ *nlength nell*
 $k \leq n$
 shows *nidx* (*ntake* (*enat k*) *nell*)
using *assms*
using *nridx-nidx nridx-ntake* **by** *blast*

lemma *nidx-imp-ndistinct*:
 assumes *nidx nell*
 shows *ndistinct nell*
using *assms*
apply *transfer*
using *lidx-imp-ldistinct* **by** *auto*

lemma *ndistinct-Ex1*:
 assumes *ndistinct nell*
 $x \in$ *nset nell*
 shows $\exists!i.\ i \leq$ *nlength nell* $\wedge$ (*nnth nell i*) $= x$
using *assms*
by *transfer*
 (*auto*,
 metis co.enat.exhaust-sel iless-Suc-eq ldistinct-Ex1 llength-eq-0 min.orderE the-enat.simps,
 metis co.enat.exhaust-sel iless-Suc-eq ldistinct-conv-lnth llength-eq-0)

lemma *nidx-nset-eq*:
 assumes *nidx nellx*

nidx nelly
nset nellx = *nset nelly*
shows *nellx* = *nelly*
using *assms*
by *transfer*
(*simp add*: *bi-unique-cr-nellist-help lidx-lset-eq nidx.rep-eq*)

lemma *nridx-nfuse-nfirst-nlast*:
assumes *nridx R nell1*
(*nnth nell1 0*) = (*0*::*nat*)
nridx R nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell
nlast nell1 = *cp*
shows *nlast nell1* = *nfirst*(*nmap* (*λx. x*+*cp*) *nell2*)
using *assms*
by (*metis add.commute add.right-neutral ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-le*)

lemma *nidx-nfuse-nfirst-nlast*:
assumes *nidx nell1*
(*nnth nell1 0*) = (*0*::*nat*)
nidx nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell
nlast nell1 = *cp*
shows *nlast nell1* = *nfirst*(*nmap* (*λx. x*+*cp*) *nell2*)
using *assms*
by (*metis add.commute add.right-neutral ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-le*)

lemma *nridx-nfuse-nnth-cp*:
assumes *nridx R nell1*
(*nnth nell1 0*) = *0*
nridx R nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = *cp*
nlast nell2 = *the-enat*((*nlength nell*)) − *cp*
i ≤ (*nlength nell2*)
cp ≤ *nlength nell*
shows *nnth* (*nfuse nell1* (*nmap* (*λx. x*+*cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) = *cp* + (*nnth nell2 i*)
proof −
have *1*: *nlast nell1* = *nfirst* (*nmap* (*λx. x*+*cp*) *nell2*)
by (*metis add-0 assms*(*4*) *assms*(*8*) *ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-le*)
have *2*: *nnth* (*nfuse nell1* (*nmap* (*λx. x*+*cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) =
nnth (*nmap* (*λx. x*+*cp*) *nell2*) *i*
by (*simp add*: *1 assms nfuse-nnth-a*)

have *3*: *nnth* (*nmap* ($\lambda x.$ *x*+*cp*) *nell2*) *i* = *cp* + (*nnth nell2 i*)
by (*simp add*: *assms*)
show *?thesis*
by (*simp add*: *2 3*)
qed

lemma *nidx-nfuse-nnth-cp*:
assumes *nidx nell1*
(*nnth nell1 0*) = *0*
nidx nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = *cp*
nlast nell2 = *the-enat*((*nlength nell*)) − *cp*
i ≤ (*nlength nell2*)
cp ≤ *nlength nell*
shows *nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+*cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) = *cp* + (*nnth nell2 i*)
using *assms nridx-nfuse-nnth-cp nridx-nidx* **by** *blast*

lemma *nridx-nfuse-nnth-cp-a*:
assumes *nridx R nell1*
(*nnth nell1 0*) = (*0*::*nat*)
nridx R nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = *cp*
nlast nell2 = *the-enat*((*nlength nell*)) − *cp*
i ≤ ((*nlength nell1*)) + (*nlength nell2*)
the-enat((*nlength nell1*)) ≤ *i*
cp ≤ *nlength nell*
shows *nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+*cp*) *nell2*)) (*i*) = *cp* + (*nnth nell2* (*i* −*the-enat*((*nlength nell1*))))

proof −
have *1*: *i* = (*the-enat* ((*nlength nell1*)) + (*i* − *the-enat* ((*nlength nell1*))))
by (*simp add*: *assms*)
have *2*: *enat* ((*i*::*nat*) − *the-enat* ((*nlength nell1*))) ≤ *nlength nell2*
using *assms*
by (*metis enat-add-sub-same enat-minus-mono1 enat-ord-simps*(*1*)
idiff-enat-enat infinity-ileE nfinite-conv-nlength-enat the-enat.simps)
show *?thesis* **using** *1 2 assms nridx-nfuse-nnth-cp*[*of R nell1 nell2 nell cp* (*i* − *the-enat* ((*nlength nell1*)))]
by *presburger*
qed

lemma *nidx-nfuse-nnth-cp-a*:
assumes *nidx nell1*

(*nnth nell1 0*) = (*0*::*nat*)
nidx nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = *cp*
nlast nell2 = *the-enat*((*nlength nell*)) − *cp*
i ≤ ((*nlength nell1*)) + (*nlength nell2*)
the-enat((*nlength nell1*)) ≤ *i*
cp ≤ *nlength nell*
shows *nnth* (*nfuse nell1* (*nmap* (λ*x*. *x*+*cp*) *nell2*)) (*i*) = *cp* + (*nnth nell2* (*i* −*the-enat*((*nlength nell1*))))

using *assms nridx-nidx nridx-nfuse-nnth-cp-a* **by** *blast*

lemma *nridx-nfuse-nnth-cp-nlast*:
assumes *nridx R nell1*
(*nnth nell1 0*) = *0*
nridx R nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = *cp*
nlast nell2 = *the-enat*((*nlength nell*)) − *cp*
i ≤ (*nlength nell2*)
cp ≤ *nlength nell*
shows *nlast* (*nfuse nell1* (*nmap* (λ*x*. *x*+*cp*) *nell2*)) = (*the-enat* ((*nlength nell*)))
proof −
have *1*: *nlast* (*nfuse nell1* (*nmap* (λ*x*. *x*+*cp*) *nell2*)) = *nlast* (*nmap* (λ*x*. *x*+*cp*) *nell2*)
using *assms*
by (*metis add-0 ndropn-0 ndropn-nfirst nlast-nfuse nnth-nmap zero-enat-def zero-le*)
have *2*: *nlast* (*nmap* (λ*x*. *x*+*cp*) *nell2*) = *cp* + (*nlast nell2*)
by (*simp add*: *assms*(*6*))
have *3*: *cp* + (*nlast nell2*) = (*the-enat* ((*nlength nell*)))
using *assms*
by (*metis add.commute diff-add enat-ord-simps*(*1*) *nfinite-conv-nlength-enat the-enat.simps*)
show *?thesis*
by (*simp add*: *1 3 add.commute assms*(*6*))
qed

lemma *nidx-nfuse-nnth-cp-nlast*:
assumes *nidx nell1*
(*nnth nell1 0*) = *0*
nidx nell2
(*nnth nell2 0*) = *0*
nfinite nell1
nfinite nell2
nfinite nell
nlast nell1 = *cp*

nlast nell2 = *the-enat*((*nlength nell*)) − *cp*
i ≤ (*nlength nell2*)
cp ≤ *nlength nell*
shows *nlast* (*nfuse nell1* (*nmap* (*λx. x*+*cp*) *nell2*)) = (*the-enat* ((*nlength nell*)))
using *assms nridx-nidx nridx-nfuse-nnth-cp-nlast* **by** *blast*

lemma *nridx-nfuse-nnth-cp-infinite*:
assumes *nridx R nell1*
(*nnth nell1 0*) = (*0*::*nat*)
nridx R nell2
(*nnth nell2 0*) = *0*
nfinite nell1
¬*nfinite nell2*
¬*nfinite nell*
nlast nell1 = *cp*
shows *nnth* (*nfuse nell1* (*nmap* (*λx. x*+*cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) = *cp* + (*nnth nell2 i*)
proof −
have *1*: *nlast nell1* = *nfirst*(*nmap* (*λx. x*+*cp*) *nell2*)
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *assms*(*8*) *nridx-nfuse-nfirst-nlast*)
have *2*: *nnth* (*nfuse nell1* (*nmap* (*λx. x*+*cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *0*) = *nlast nell1*
using *assms* **by** (*metis 1 add.right-neutral ntaken-nfuse ntaken-nlast*)
have *3*: *nfirst*(*nmap* (*λx. x*+*cp*) *nell2*) = *cp* + (*nnth nell2 0*)
using *1 assms* **by** *auto*
have *8*: *i*≤ (*nlength nell2*)
by (*metis assms*(*6*) *enat-ile nfinite-conv-nlength-enat wlog-linorder-le*)
have *10*: (*nlength nell1*) ≤ *enat* (*the-enat* ((*nlength nell1*)) + (*i*::*nat*))
using *assms*(*5*) *nfinite-nlength-enat* **by** *fastforce*
have *11*: *enat* (*the-enat* ((*nlength nell1*)) + *i*) ≤ *nlength* (*nfuse nell1* (*nmap* (*λx*::*nat. x* + *cp*) *nell2*))
by (*metis 10 assms*(*6*) *enat-less-enat-plusI2 enat-ord-code*(*4*) *enat-the-enat leD nfuse-nlength nlength-eq-enat-nfiniteD nlength-nmap order-less-imp-le*)
have *12*: (*the-enat* ((*nlength nell1*)) + *i* − *the-enat* ((*nlength nell1*))) = *i*
by *auto*
have *4*: *nnth* (*nfuse nell1* (*nmap* (*λx. x*+*cp*) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) =
(*nnth* (*nmap* (*λx. x*+*cp*) *nell2*) *i*)
by (*simp add*: *1 8 assms nfuse-nnth-a*)
have *5*: (*nnth* (*nmap* (*λx. x*+*cp*) *nell2*) *i*) = *cp* + (*nnth nell2 i*)
by (*simp add*: *8*)
show *?thesis*
using *4 5* **by** *presburger*
qed

lemma *nidx-nfuse-nnth-cp-infinite*:
assumes *nidx nell1*
(*nnth nell1 0*) = *0*
nidx nell2
(*nnth nell2 0*) = *0*
nfinite nell1
¬*nfinite nell2*
¬*nfinite nell*
nlast nell1 = *cp*

shows *nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+cp$) *nell2*)) (*the-enat*((*nlength nell1*)) + *i*) = *cp* + (*nnth nell2 i*)
using *assms nridx-nidx nridx-nfuse-nnth-cp-infinite* **by** *blast*

lemma *nidx-nfuse-nidx*:
assumes *nidx nell1*
nnth nell1 0 = *0*
nidx nell2
nnth nell2 0 = *0*
nfinite nell1
nlast nell1 = *cp*
nfinite nell2
nfinite nell
nlast nell2 = *the-enat*((*nlength nell*)) −*cp*
cp ≤ *nlength nell*
shows *nidx* (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) ∧ (*nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) *0*) = *0*
proof −
have *1*: *nlast nell1* = *nfirst*(*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *assms*(*6*) *nidx-nfuse-nfirst-nlast* **by** *blast*
have *2*: *nfirst* (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) = *nfirst nell1*
by (*simp add*: *1 nfirst-nfuse*)
have *4*: ⋀*j*. *j*≤*nlength nell1* ⟹ *nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) *j*= *nnth nell1 j*
using *assms* **by** (*simp add*: *1 add-increasing2 nfuse-nlength nfuse-nnth*)
have *40*: ∃ *k1*. *nlength nell1* = (*enat k1*)
by (*simp add*: *assms*(*5*) *nfinite-nlength-enat*)
obtain *k1* **where** *41*: *nlength nell1* = (*enat k1*)
using *40* **by** *blast*
have *5*: ⋀*j*. (*nlength nell1*) ≤*j* ∧ *j* ≤(*nlength nell1*) + *nlength nell2* ⟹
nnth (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) *j* =
cp + (*nnth nell2* (*j*− *the-enat*((*nlength nell1*))))
using *assms nidx-nfuse-nnth-cp-a*[*of nell1 nell2 nell cp*]
by (*metis 41 enat-ord-simps*(*1*) *the-enat.simps*)
have *45*: ⋀*j*. *k1* ≤*j* ∧ *j* ≤ (*enat* (*k1*)) + *nlength nell2* ⟹
nnth (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) *j* =
cp + (*nnth nell2* (*j*− (*k1*)))
by (*simp add*: *41 5 order-less-imp-le*)
have *50*: *nlength*(*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) = (*nlength nell1*) + *nlength nell2*
by (*simp add*: *nfuse-nlength*)
have *51*: ⋀*j*. *enat* (*Suc j*) ≤ *nlength nell1* ⟹
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) *j*) <
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) (*Suc j*))
using *4 Suc-ile-eq assms*(*1*) *nidx-expand* **by** *auto*
have *52*: ⋀*j*. *k1* ≤*j* ∧ (*Suc j*) ≤ *enat* (*k1*) + *nlength nell2* ⟹
cp + (*nnth nell2* (*j*− (*k1*))) <
cp + (*nnth nell2* ((*Suc j*)− (*k1*)))
using *assms*(*3*) **unfolding** *nidx-def*
by (*metis Nat.add-diff-assoc add-strict-left-mono assms*(*3*) *enat.simps*(*3*) *enat-add-sub-same enat-minus-mono1 idiff-enat-enat nidx-expand plus-1-eq-Suc*)
have *6*: ⋀*j*. *enat* (*Suc j*) ≤ *nlength*(*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) ⟹
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.\ x+$ *cp*) *nell2*)) *j*) <

(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) (*Suc j*))
proof −
fix *j*
assume *a*: *enat* (*Suc j*) $\leq$ *nlength*(*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*))
show (*nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) *j*) < (*nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) (*Suc j*))
proof −
have *61*: *enat* (*Suc j*) $\leq$ *nlength nell1* $\Longrightarrow$
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) *j*) < (*nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) (*Suc j*))
using *51* **by** *blast*
have *62*: *k1* $\leq$*j* $\wedge$ (*Suc j*) $\leq$ *nlength*(*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) $\Longrightarrow$
(*nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) *j*) < (*nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) (*Suc j*))
using *41 5 50 52 Suc-ile-eq* **by** *force*
show *?thesis*
using *41 61 62 a* **by** *fastforce*
qed
qed
show *?thesis*
unfolding *nidx-expand*
using *4 6 assms zero-enat-def* **by** *force*
qed

lemma *nidx-nfuse-nidx-infinite*:
assumes *nidx nell1*
nnth nell1 0 = *0*
nidx nell2
nnth nell2 0 = *0*
nfinite nell1
nlast nell1 = *cp*
$\neg$*nfinite nell2*
$\neg$*nfinite nell*
shows *nidx* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) $\wedge$ (*nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) *0*) = *0*
proof −
have *1*: *nlast nell1* = *nfirst*(*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)
by (*metis add-0 assms*(*4*) *assms*(*6*) *assms*(*7*) *enat-le-plus-same*(*1*) *enat-le-plus-same*(*2*) *enat-the-enat infinity-ileE ndropn-0 ndropn-nfirst nfinite-conv-nlength-enat nnth-nmap*)
have *2*: *nfirst* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) = *nfirst nell1*
by (*simp add*: *1 nfirst-nfuse*)
have *4*: $\bigwedge$*j*. *j*$\leq$*nlength nell1* $\Longrightarrow$ *nnth* (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) *j*= *nnth nell1 j*
by (*simp add*: *1 add-increasing2 nfuse-nlength nfuse-nnth*)
have *40*: $\exists$ *k1*. *nlength nell1* = (*enat k1*)
by (*simp add*: *assms*(*5*) *nfinite-nlength-enat*)
obtain *k1* **where** *41*: *nlength nell1* = (*enat k1*)
using *40* **by** *blast*
have *5*: $\bigwedge$*j*. (*nlength nell1*) $\leq$*j* $\wedge$ *j* $\leq$ (*nlength nell1*) + *nlength nell2* $\Longrightarrow$
nnth (*nfuse nell1* (*nmap* ($\lambda x.$ *x*+ *cp*) *nell2*)) *j* =
cp + (*nnth nell2* (*j*− *the-enat*((*nlength nell1*))))

using *assms nidx-nfuse-nnth-cp-infinite*[*of nell1 nell2 nell cp*]
by (*metis 41 enat-ord-simps*(*1*) *le-add-diff-inverse the-enat.simps*)
have *45*: $\bigwedge j.\ k1 \le j \wedge j \le (enat\ (k1)) + nlength\ nell2 \Longrightarrow$
$nnth\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2))\ j =$
$cp + (nnth\ nell2\ (j-\ (k1)))$
using *41 5 enat-ord-simps*(*1*) *the-enat.simps* **by** *presburger*
have *50*: $nlength(nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2)) = (nlength\ nell1) + nlength\ nell2$
by (*simp add*: *nfuse-nlength*)
have *51*: $\bigwedge j.\ enat\ (Suc\ j) \le nlength\ nell1 \Longrightarrow$
$(nnth\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2))\ j) <$
$(nnth\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2))\ (Suc\ j))$
using *4 Suc-ile-eq assms*(*1*) *nidx-expand* **by** *auto*
have *52*: $\bigwedge j.\ k1 \le j \wedge (Suc\ j) \le enat\ (k1) + nlength\ nell2 \Longrightarrow$
$cp + (nnth\ nell2\ (j-\ (k1))) <$
$cp + (nnth\ nell2\ ((Suc\ j)-\ (k1)))$
by (*metis Nat.add-diff-assoc add-strict-left-mono assms*(*3*) *enat.simps*(*3*) *enat-add-sub-same enat-minus-mono1 idiff-enat-enat nidx-expand plus-1-eq-Suc*)
have *53*: $\bigwedge j.\ k1 \le j \wedge (Suc\ j) \le enat\ (k1) + nlength\ nell2 \Longrightarrow$
$nnth\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2))\ j <$
$nnth\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2))\ (Suc\ j)$
by (*metis 45 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le*)
have *6*: $\bigwedge j.\ enat\ (Suc\ j) \le nlength(nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2)) \Longrightarrow$
$(nnth\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2))\ j) <$
$(nnth\ (nfuse\ nell1\ (nmap\ (\lambda x.\ x+\ cp)\ nell2))\ (Suc\ j))$
by (*metis 41 50 51 53 enat-ord-simps*(*1*) *le-SucE linorder-le-cases*)
show *?thesis* **unfolding** *nidx-expand*
using *4 6 assms zero-enat-def* **by** *fastforce*
qed

lemma *nsubn-nfuse-nidx*:
assumes *nidx nl*
nfinite nl
nfinite nell
$nlast\ nl = (nlength\ nell)$
$(Suc\ i) \le (nlength\ nl)$
shows $nfuse\ (nsubn\ nell\ (nnth\ nl\ i)\ (nnth\ nl\ (Suc\ i)))\ (nsubn\ nell\ (nnth\ nl\ (Suc\ i))\ (nlast\ nl)) =$
$(nsubn\ nell\ (nnth\ nl\ i)\ (nlast\ nl))$
proof −
have *1*: $(nnth\ nl\ i) \le (nnth\ nl\ (Suc\ i))$
by (*simp add*: *assms*(*1*) *assms*(*5*) *nidx-less-eq*)
have *2*: $nlast\ nl = (nnth\ nl\ (the\text{-}enat(\ (nlength\ nl))))$
by (*simp add*: *assms*(*2*) *nnth-nlast*)
have *3*: $1 \le nlength\ nl$
by (*metis assms*(*5*) *dual-order.trans enat-0-iff*(*1*) *iless-Suc-eq le-zero-eq linorder-le-cases nat.simps*(*3*) *one-eSuc order-neq-le-trans*)
have *4*: $(nnth\ nl\ (Suc\ i)) \le (nlast\ nl)$
by (*metis 2 assms*(*1*) *assms*(*2*) *assms*(*5*) *dual-order.eq-iff enat-ord-simps*(*1*) *nfinite-conv-nlength-enat nidx-less-eq the-enat.simps*)
have *5*: $enat\ (nnth\ nl\ (Suc\ i)) \le nlength\ nell$

by (*metis 4 assms*(*4*) *enat-ord-simps*(*1*))
have *6*: *nlast* (*nsubn nell* (*nnth nl i*) (*nnth nl* (*Suc i*))) = (*nnth nell* (*nnth nl* (*Suc i*)))
by (*simp add*: *1 nsubn-def1 ntaken-nlast*)
have *7*: *nfirst* (*nsubn nell* (*nnth nl* (*Suc i*)) (*nlast nl*)) = (*nnth nell* (*nnth nl* (*Suc i*)))
by (*simp add*: *nsubn-def1 ntaken-ndropn-nfirst*)
show *?thesis*
by (*simp add*: *1 5 assms*(*4*) *nsubn-nfuse*)
qed

lemma *nidx-nfuse-split*:
assumes *nlast nell1* = *nfirst nell2*
shows *nridx R* (*nfuse nell1 nell2*) $\longleftrightarrow$
(*if nfinite nell1 then nridx R nell1* $\wedge$ *nridx R nell2 else nridx R nell1*)
proof (*cases nfinite nell1*)
case *True*
then show *?thesis*
by (*metis assms nappend-nbutlast-nlast-id nbutlast-nfinite nellist.collapse*(*2*) *nellist.disc*(*1*)
nfuse-def1 nfuse-leftneutral nhd-nfuse nlast-NNil nridx-LCons-1 nridx-nappend-nfinite)
next
case *False*
then show *?thesis* **by** (*simp add*: *assms nfuse-nappend*)
qed

lemma *nidx-all-le-nlast*:
assumes *nidx nell*
nfinite nell
j$\leq$ *nlength nell*
shows *nnth nell j* $\leq$ *nlast nell*
using *assms*
by (*metis nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast order.order-iff-strict the-enat.simps*)

lemma *nidx-shiftm* :
assumes *nidx nell*
nnth nell 0 = *k*
shows *nidx* (*nmap* (λx .*x*$-$ *k*) *nell*) $\wedge$ *nnth* (*nmap* (λx .*x*$-$ *k*) *nell*) *0* = *0* $\wedge$ *k* $\leq$ (*nnth nell 0*)
using *assms zero-enat-def*
by (*auto simp add*: *nidx-expand*)
(*metis Suc-ile-eq add-diff-inverse-nat add-gr-0 assms*(*1*) *diff-less-mono nidx-gr-first*
nnth-nmap order-less-imp-le zero-less-Suc zero-less-diff)

lemma *nidx-nsubn*:
assumes *k*$\leq$*n*
n $\leq$ *nlength nell*
nidx nell
nnth nell 0 = *0*
shows *nidx* (*nsubn nell k n*) $\wedge$ *nnth* (*nsubn nell k n*) *0* = (*nnth nell k*)
using *assms* **unfolding** *nidx-expand nsubn-def1*
by (*auto simp add*: *ntaken-nnth*)
(*simp add*: *Nat.le-diff-conv2 add.commute order-subst2*)

lemma *nidx-ntaken-niterates-Suc*:
nidx (*ntaken n* (*niterates Suc 0*))
proof –
have *1*: *nidx* (*ntaken n* (*niterates Suc 0*)) =
(∀ *i*::*nat*.
enat (*Suc i*) ≤ *nlength* (*ntaken n* (*niterates Suc* (*0*::*nat*))) ⟶
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) *i* <
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) (*Suc i*))
unfolding *nidx-expand* **by** *auto*
have *2*: (∀ *i*::*nat*.
enat (*Suc i*) ≤ *nlength* (*ntaken n* (*niterates Suc* (*0*::*nat*))) ⟶
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) *i* <
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) (*Suc i*))
proof
fix *i*
show *enat* (*Suc i*) ≤ *nlength* (*ntaken n* (*niterates Suc* (*0*::*nat*))) ⟶
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) *i* <
nnth (*ntaken n* (*niterates Suc* (*0*::*nat*))) (*Suc i*)
proof (*induct i arbitrary*: *n*)
case *0*
then show *?case*
by (*simp add*: *ntaken-nnth*)
next
case (*Suc i*)
then show *?case* **by** (*simp add*: *ntaken-nnth*)
qed
qed
show *?thesis*
using *1 2* **by** *fastforce*
qed

3.7 nlastnfirst

lemma *nlastnfirst-def2*:
nlastnfirst = *llastlfirst* ∘ (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
by (*simp add*: *map-fun-def nlastnfirst-def*)

lemma *nlastnfirst-NNil*[*simp*]:
nlastnfirst (*NNil x*)
apply *transfer*
by *simp*

lemma *nlastnfirst-LCons*[*simp*]:
shows *nlastnfirst* (*NCons nell nells*) ⟷
nlast nell = *nfirst* (*nfirst nells*) ∧ *nlastnfirst nells*
proof –
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlastnfirst* (*NCons nell nells*) =

(*llastlfirst* ∘ *?n2l*) (*NCons nell nells*)
by (*simp add*: *nlastnfirst-def2*)
have *2*: (*llastlfirst* ∘ *?n2l*) (*NCons nell nells*) =
(*llast* (*llist-of-nellist nell*) = *lfirst* (*lfirst* (*?n2l nells*)) ∧ *llastlfirst* (*?n2l nells*))
by *simp*
have *3*: *llastlfirst* (*?n2l nells*) = *nlastnfirst nells*
by (*simp add*: *nlastnfirst.rep-eq*)
have *4*: *llast* (*llist-of-nellist nell*) = *nlast nell*
by (*metis llast-linfinite nfinite-def nlast-llast nlast-not-nfinite*)
have *5*: *lfirst* (*lfirst* (*?n2l nells*)) = *nfirst* (*nfirst nells*)
by (*simp add*: *lfirst-def llist-of-nellist.simps*(*2*) *nfirst-def*)
show *?thesis*
using *1 2 3 4 5* **by** *presburger*
qed

lemma *nlastnfirst-def1*:
shows *nlastnfirst nells* =
(∀ *i*. (*Suc i*)≤ *nlength nells* ⟶ *nlast*(*nnth nells i*) = *nfirst*(*nnth nells* (*Suc i*)))
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlastnfirst nells* = (*llastlfirst* ∘ *?n2l*) *nells*
by (*simp add*: *nlastnfirst-def2*)
have *2*: (*llastlfirst* ∘ *?n2l*) *nells* = *llastlfirst* (*?n2l nells*)
by *simp*
have *3*: *llastlfirst* (*?n2l nells*) ⟷
(∀ *i*. (*Suc i*) < *llength* (*?n2l nells*) ⟶
llast (*lnth* (*?n2l nells*) *i*) = *lfirst* (*lnth* (*?n2l nells*) (*Suc i*)))
using *llastlfirst-def* **by** *blast*
have *4*: *epred* (*llength* (*?n2l nells*)) = *nlength nells*
by *simp*
have *5*: ⋀*xs j*. *j*≤ *nlength xs* ⟶ *nnth xs j* = *lnth* (*llist-of-nellist xs*) *j*
unfolding *nnth-def* **by** *auto*
have *6*: ⋀*lx j* . *j*<*llength lx* ∧ ¬ *lnull lx*⟶ *lnth lx j* = *nnth* (*nellist-of-llist lx*) *j*
by (*metis 5 co.enat.exhaust-sel iless-Suc-eq llength-eq-0 nellist-of-llist-inverse nlength.abs-eq*)
have *7*: ⋀*xs*. *nlast xs* = *llast* (*llist-of-nellist xs*)
by (*metis llast-linfinite nfinite-def nlast-llast nlast-not-nfinite*)
have *8*: ⋀*xs*. *nfirst xs* = *lfirst* (*llist-of-nellist xs*)
by (*simp add*: *lfirst-def llist-of-nellist.simps*(*2*) *nfirst-def*)
have *9*: ⋀*lx*. ¬ *lnull lx* ⟹ *llast lx* = *nlast* (*nellist-of-llist lx*)
by (*simp add*: *7*)
have *10*: ⋀*lx*. ¬ *lnull lx* ⟹ *lfirst lx* = *nfirst* (*nellist-of-llist lx*)
by (*simp add*: *8*)
have *11*: ⋀ *j*. *j* < *llength* (*?n2l nells*) ⟶
llast (*lnth* (*?n2l nells*) *j*) =
nlast (*nellist-of-llist* (*lnth* (*?n2l nells*) *j*))
by (*simp add*: *9*)
have *12*: ⋀ *j*. (*enat j*) < *llength* (*?n2l nells*) ⟶
nlast (*nellist-of-llist* (*lnth* (*?n2l nells*) *j*)) =
nlast (*nellist-of-llist* (*llist-of-nellist* (*lnth* (*llist-of-nellist nells*) *j*)))

by *simp*
have *13*: $\bigwedge$ *j*. (*enat j*) < *llength* (*?n2l nells*) ⟶
nlast (*nellist-of-llist* (*llist-of-nellist* (*lnth* (*llist-of-nellist nells*) *j*))) =
nlast (*lnth* (*llist-of-nellist nells*) *j*)
by *auto*
have *14*: $\bigwedge$ *j*. (*enat j*) < *llength* (*?n2l nells*) ⟶
nlast (*lnth* (*llist-of-nellist nells*) *j*) = *nlast*(*nnth nells j*)
by (*simp add*: *6*)
have *15*: $\bigwedge$ *j*. *j* < *llength* (*?n2l nells*) ⟶
llast (*lnth* (*?n2l nells*) *j*) =
nlast(*nnth nells j*)
using *11 12 13 14* **by** *presburger*
have *16*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) ⟶
lfirst (*lnth* (*?n2l nells*) (*Suc j*)) =
nfirst (*nellist-of-llist* (*lnth* (*?n2l nells*) (*Suc j*)))
by (*simp add*: *10*)
have *17*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) ⟶
nfirst (*nellist-of-llist* (*lnth* (*?n2l nells*) (*Suc j*))) =
nfirst (*nellist-of-llist* (*llist-of-nellist* (*lnth* (*llist-of-nellist nells*) (*Suc j*))))
by *simp*
have *18*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) ⟶
nfirst (*nellist-of-llist* (*llist-of-nellist* (*lnth* (*llist-of-nellist nells*) (*Suc j*)))) =
nfirst (*lnth* (*llist-of-nellist nells*) (*Suc j*))
by *auto*
have *19*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) ⟶
nfirst (*lnth* (*llist-of-nellist nells*) (*Suc j*)) = *nfirst* (*nnth nells* (*Suc j*))
by (*simp add*: *6*)
have *20*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) ⟶
lfirst (*lnth* (*?n2l nells*) (*Suc j*)) = *nfirst* (*nnth nells* (*Suc j*))
using *16 17 18 19* **by** *presburger*
have *21*: $\bigwedge$ *j*. (*Suc j*) < *llength* (*?n2l nells*) ⟷
(*Suc j*) ≤ *nlength nells*
by (*metis 4 co.enat.exhaust-sel dual-order.strict-iff-order enat-0-iff*(*1*) *epred-0 iless-Suc-eq nat.simps*(*3*))
have *22*: (∀ *i*. (*Suc i*) < *llength* (*?n2l nells*) ⟶
llast (*lnth* (*?n2l nells*) *i*) = *lfirst* (*lnth* (*?n2l nells*) (*Suc i*)))
⟷
(∀ *i*. (*Suc i*)≤ *nlength nells* ⟶ *nlast*(*nnth nells i*) = *nfirst*(*nnth nells* (*Suc i*)))
by (*metis 15 20 21 Suc-ile-eq order-less-imp-le*)
have *23*: *llastlfirst* (*?n2l nells*) ⟷
(∀ *i*. (*Suc i*)≤ *nlength nells* ⟶ *nlast*(*nnth nells i*) = *nfirst*(*nnth nells* (*Suc i*)))
using *22 3* **by** *presburger*
show *?thesis* **using** *1 23* **by** *auto*
qed

lemma *nlastnfirst-nridx*:
nlastnfirst nells = *nridx* (λ *a b*. *nlast a* = *nfirst b*) *nells*
by (*simp add*: *nlastnfirst-def1 nridx-expand*)

lemma *nlastnfirst-nappend-nfinite*:
assumes *nfinite nellxs*
shows *nlastnfirst* (*nappend nellxs nellys*) $\longleftrightarrow$
nlastnfirst nellxs $\wedge$ *nlastnfirst nellys* $\wedge$ *nlast*(*nlast nellxs*) = *nfirst*(*nfirst nellys*)
using *assms*
by (*metis* (*mono-tags*, *lifting*) *nlastnfirst-nridx nridx-nappend-nfinite*)

3.8 nfusecat

lemma *nfusecat-def2*:
nfusecat = *nellist-of-llist* $\circ$ *lfusecat* $\circ$ (*lmap llist-of-nellist* $\circ$ *llist-of-nellist*)
by (*simp add*: *map-fun-def nfusecat-def*)

lemma *not-null-lfusecat*:
$\neg$*lnull*((*lfusecat* $\circ$ (*lmap llist-of-nellist* $\circ$ *llist-of-nellist*)) *nells*)
by (*simp add*: *llist-of-nellist.code*)

lemma *nfusecat-def3*:
((*llist-of-nellist* $\circ$ *nfusecat*) *nells*) =
((*lfusecat* $\circ$ (*lmap llist-of-nellist* $\circ$ *llist-of-nellist*)) *nells*)
proof –
let *?n2l* = (*lmap llist-of-nellist* $\circ$ *llist-of-nellist*)
have *1*: *llist-of-nellist* $\circ$ *nfusecat* =
llist-of-nellist $\circ$ *nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*
by (*simp add*: *nfusecat-def2 rewriteL-comp-comp*)
have *2*: $\neg$*lnull*((*lfusecat* $\circ$ *?n2l*) *nells*)
using *not-null-lfusecat* **by** *auto*
have *3*: (*llist-of-nellist* $\circ$ *nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*) *nells* =
(*lfusecat* $\circ$ *?n2l*) *nells*
using *2 nellist-of-llist-inverse* **by** *simp*
show *?thesis*
by (*metis 1 3*)
qed

lemma *nfusecat-NNil*[*simp*]:
nfusecat (*NNil nell*) = *nell*
apply *transfer*
by *simp*

lemma *nfusecat-NCons*[*simp*]:
nfusecat (*NCons nell nells*) = *nfuse nell* (*nfusecat nells*)
apply *transfer*
by (*simp add*: *lfuse-def llist.case-eq-if*)

lemma *nfusecat-nfirst*:
assumes ($\exists$ *x1*. *nells* = *NNil x1*)
shows *nfusecat nells* = *nfirst nells*
proof –

obtain *nell* **where** *1*: *nells* = *NNil nell*
using *assms* **by** *auto*
have *2*: *nfusecat nells* = *nell*
by (*simp add*: *1*)
have *3*: *nfirst nells* = *nell*
by (*metis 1 ndropn-0 ndropn-nfirst nnth-NNil*)
show *?thesis*
by (*simp add*: *2 3*)
qed

lemma *nfusecat-NCons-nfirst*:
assumes ($\forall$ *nell*. *nells* $\neq$ *NNil nell*)
shows *nfusecat nells* = *nfuse* (*nfirst nells*) (*nfusecat* (*ntl nells*))
by (*metis assms nellist-split-2-first nlast-NNil nfusecat-NCons not-le-imp-less ntaken-0 ntaken-all ntaken-nlast zero-enat-def*)

lemma *nfusecat-expand*:
nfusecat nells =
(*if is-NNil nells then nfirst nells else nfuse* (*nfirst nells*) (*nfusecat* (*ntl nells*)))
unfolding *is-NNil-def*
using *nfusecat-NCons-nfirst nfusecat-nfirst* **by** *simp blast*

lemma *nfusecat-expand-case*:
nfusecat nells = (*case nells of* (*NNil nell*) $\Rightarrow$ *nell* |
(*NCons nell′ nells1*) $\Rightarrow$ *nfuse nell′* (*nfusecat nells1*))
by (*metis is-NNil-def nellist.case-eq-if nellist.collapse*(*2*) *nlast-NNil nfusecat-NCons nfusecat-NNil*)

lemma *nmap-nfusecat*:
nmap f (*nfusecat nells*) = (*nfusecat* (*nmap* (*nmap f*) *nells*))
proof −
let *?n2l* = (*lmap llist-of-nellist* $\circ$ *llist-of-nellist*)
have *1*: *nmap f* (*nfusecat nells*) =
nmap f ((*nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*) *nells*)
by (*simp add*: *nfusecat-def2*)
have *2*: *nmap f* ((*nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*) *nells*) =
nellist-of-llist (*lmap f* ((*lfusecat* $\circ$*?n2l*) *nells*))
using *nmap-nellist-of-llist not-null-lfusecat* **by** *auto*
have *3*: (*lmap f* ((*lfusecat* $\circ$ *?n2l*) *nells*)) =
lfusecat (*lmap* (*lmap f*) ((*lmap llist-of-nellist* $\circ$ *llist-of-nellist*) *nells*))
by (*simp add*: *lmap-lfusecat*)
have *4*: (*nfusecat* (*nmap* (*nmap f*) *nells*)) =
(*nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*) (*nmap* (*nmap f*) *nells*)
by (*simp add*: *nfusecat-def2*)
have *8*: (*lmap* (*lmap f*) (*lmap llist-of-nellist* (*llist-of-nellist nells*))) =
(*lmap llist-of-nellist* (*lmap* (*nmap f*) (*llist-of-nellist nells*)))
using *lmap-llist-of-nellist-nmap* **by** *blast*
show *?thesis*
using *1 2 3 4 8* **by** *fastforce*
qed

lemma *nfirst-nfuse-var*:
nfirst (*nfuse nellx nelly*) = *nfirst nellx*
by (*metis nfuse-def1 nlast-NNil ntaken-0 ntaken-nappend1 zero-enat-def zero-le*)

lemma *nfirst-nfusecat-nfirst*:
shows *nfirst*(*nfusecat nells*) = *nfirst*(*nfirst nells*)
proof (*cases nells*)
case (*NNil nell*)
then show *?thesis*
by (*simp add*: *nfusecat-expand*)
next
case (*NCons nell nells1*)
then show *?thesis*
proof −
have *1*: *nfusecat* (*NCons nell nells1*) = *nfuse nell* (*nfusecat nells1*)
by (*simp*)
have *2*: *nfirst* (*nfuse nell* (*nfusecat nells1*)) = *nfirst nell*
using *nfirst-nfuse-var* **by** *auto*
have *3*: *nfirst*(*nfirst* (*NCons nell nells1*)) = *nfirst nell*
by (*metis 1 nfirst-nfuse-var nfusecat-expand*)
show *?thesis*
using *1 2 3 NCons* **by** *fastforce*
qed
qed

lemma *nfirst-nfusecat*:
shows *nfirst*(*nfusecat* (*NCons nell nells*)) = *nfirst nell*
by (*simp add*: *nfirst-nfuse-var*)

lemma *nfusecat-nlength-eq-zero-conv*:
nlength (*nfusecat nells*) = *0* ⟷ (∀ *nell* ∈ *nset nells*. *nlength nell* = *0*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlength* (*nfusecat nells*) =
nlength ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)
by (*simp add*: *nfusecat-def2*)
have *2*: *nlength* ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*) =
epred (*llength* (*lfusecat* (*?n2l nells*)))
by *simp*
(*metis lbutlast-snoc llength-lbutlast nellist-of-llist-a-inverse nlength.rep-eq*)
have *3*: *nlength* (*nfusecat nells*) = *0* ⟷
epred (*llength* (*lfusecat* (*?n2l nells*))) = *0*
using *1 2* **by** *presburger*
have *4*: *llength* (*?n2l nells*) > *0*
by *simp*
have *5*: (*llength* (*lfusecat* (*?n2l nells*))) > *0*
by (*simp add*: *lfusecat-not-lnull-var*)

have *6*: *epred* (*llength* (*lfusecat* (*?n2l nells*))) = *0* ⟷
(*llength* (*lfusecat* (*lmap llist-of-nellist* (*llist-of-nellist nells*)))) = *1*
by (*metis 5 comp-apply epred-1 epred-inject not-iless0 zero-one-enat-neq*(*1*))
have *7*: (*llength* (*lfusecat* (*?n2l nells*))) ≤ *1* ⟷
(∀ *nell*∈ *lset* (*?n2l nells*). *llength nell* ≤ *1*)
using *lfusecat-all-empty-or-LNil lset-lltl-llength-var* **by** *blast*
have *8*: (∀ *nell*∈ *lset* (*?n2l nells*). *llength nell* > *0*)
by *simp*
have *9*: (*llength* (*lfusecat* (*?n2l nells*))) = *1* ⟷
(∀ *nell*∈ *lset* (*?n2l nells*). *llength nell* = *1*)
by (*metis 5 7 8 ileI1 one-eSuc order-antisym-conv*)
have *10*: (∀ *nell*∈ *lset* (*?n2l nells*). *llength nell* = *1*) ⟷
(∀ *nell* ∈ *nset nells*. *nlength nell* = *0*)
by *simp*
(*metis co.enat.exhaust-sel epred-1 llength-eq-0 llist-of-nellist-not-lnull nlength.rep-eq*
zero-neq-one)
show *?thesis* **using** *3 6 9 10* **by** *simp*
qed

lemma *is-NNil-nfusecat-a*:
assumes ∀ *i*. *i*≤*nlength nells* ⟶ *is-NNil* (*nnth nells i*)
shows *is-NNil*(*nfusecat nells*)
using *assms nfusecat-nlength-eq-zero-conv*[*of nells*]
by (*metis in-nset-conv-nnth is-NNil-def nle-le nlength-NNil ntaken-0 ntaken-all zero-enat-def*)

lemma *is-NNil-nfusecat-b*:
assumes *is-NNil*(*nfusecat nells*)
shows ∀ *i*. *i*≤*nlength nells* ⟶ *is-NNil* (*nnth nells i*)
using *assms nfusecat-nlength-eq-zero-conv*[*of nells*]
by (*metis in-nset-conv-nnth nellist.collapse*(*1*) *nellist.collapse*(*2*) *nlength-NCons nlength-NNil*
zero-ne-eSuc)

lemma *ntl-lfusecat* :
shows *ntl*(*nfusecat nells*) =
(*if is-NNil nells then ntl* (*nfirst nells*) *else*
(*if is-NNil* (*nfirst nells*) *then*
if is-NNil (*nfusecat* (*ntl nells*))
then ntl (*nfirst nells*)
else ntl (*nfusecat* (*ntl nells*))
else
if is-NNil (*nfusecat* (*ntl nells*))
then ntl (*nfirst nells*)
else nappend (*ntl* (*nfirst nells*)) (*ntl* (*nfusecat* (*ntl nells*)))))

proof (*cases nells*)
case (*NNil nell*)
then show *?thesis* **by** (*metis nellist.disc*(*1*) *nfusecat-NNil nfusecat-expand*)
next

case (*NCons nell nells1*)
then show *?thesis*
  using *nfusecat-NCons*[*of nell nells1*] *ntl-nfuse*[*of nell* (*nfusecat nells1*)] **by** *simp*
    (*metis ndropn-0 ndropn-nfirst nnth-0*)
qed

lemma *nfusecat-nappend*:
assumes *nfinite llx*
shows *nfusecat* (*nappend llx lly*) = *nfuse* (*nfusecat llx*) (*nfusecat lly*)
proof −
 let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
 have *1*: *nfusecat* (*nappend llx lly*) =
          ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*nappend llx lly*))
   by (*simp add*: *nfusecat-def2*)
 have *2*: ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*nappend llx lly*)) =
          nellist-of-llist (*lfusecat* (*?n2l* (*nappend llx lly*)))
    by *simp*
 have *3*: (*llist-of-nellist* (*nappend llx lly*)) =
          lappend (*llist-of-nellist llx*) (*llist-of-nellist lly*)
    by (*simp add*: *llist-of-nellist-nappend*)
 have *4*: (*lmap llist-of-nellist* (*lappend* (*llist-of-nellist llx*) (*llist-of-nellist lly*))) =
           lappend (*?n2l llx*) (*?n2l lly*)
    using *lmap-lappend-distrib* **by** *auto*
 have *5*: (*lfusecat* (*lappend* (*?n2l llx*) (*?n2l lly*))) =
          lfuse (*lfusecat* (*?n2l llx*)) (*lfusecat* (*?n2l lly*))
     using *assms lfinite-lmap lfusecat-lappend nfinite-def* **by** (*metis comp-def*)
 have *6*: *nellist-of-llist* (*lfuse* (*lfusecat* (*?n2l llx*)) (*lfusecat* (*?n2l lly*))) =
          nfuse (*nfusecat llx*) (*nfusecat lly*)
    by (*simp add*: *llist-of-nellist.code nfuse.abs-eq nfusecat-def2*)
 show *?thesis*
 using *1 2 3 4 5 6* **by** *simp*
qed

lemma *nfusecat-split*:
 assumes (*Suc n*) ≤ *nlength nells*
shows *nfusecat nells* = *nfuse* (*nfusecat* (*ntaken n nells*)) (*nfusecat* (*ndropn* (*Suc n*) *nells*))
using *assms*
by (*metis nappend-ntaken-ndropn nfinite-ntaken nfusecat-nappend*)

lemma *nfusecat-split-1*:
 assumes (*Suc n*) ≤ *nlength nells*
shows *nfusecat nells* = *nfuse* (*nfusecat* (*ntake n nells*)) (*nfusecat* (*ndropn* (*Suc n*) *nells*))
using *assms*
by (*simp add*: *nfusecat-split ntake-eq-ntaken*)

lemma *nfuse-nfinite*:
 shows   *nfinite* (*nfuse nellx nelly*) ⟷ *nfinite nellx* ∧ *nfinite nelly*
apply *transfer*
using *lfuse-lfinite* **by** *blast*

lemma *nfusecat-nfinite*:
assumes ∀ *nell* ∈ *nset nells*. *nlength nell* > *0*
∀ *nell* ∈ *nset nells*. *nfinite nell*
nlastnfirst nells
shows *nfinite* (*nfusecat nells*) ⟷ *nfinite nells*
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nfinite* (*nfusecat nells*) ⟷
nfinite (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))
by (*simp add*: *nfusecat-def2*)
have *2*: *nfinite* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)) ⟷
lfinite (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)))
using *nfinite-def* **by** *blast*
have *3*: *lfinite* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))) ⟷
lfinite (*lfusecat* (*?n2l nells*))
by (*simp add*: *lbutlast-lfinite*)
have *4*: ∀ *nell* ∈ *lset* (*llist-of-nellist nells*). *nlength nell* > *0*
using *assms*(*1*) **by** *force*
have *5*: ∀ *nell* ∈ *lset* (*llist-of-nellist nells*). *epred*(*llength* (*llist-of-nellist nell*)) > *0*
using *4* **by** *fastforce*
have *6*: ∀ *nell* ∈ *lset* (*llist-of-nellist nells*). ¬ *lnull* (*ltl* (*llist-of-nellist nell*))
by (*metis 5 epred-llength less-numeral-extra*(*3*) *llength-LNil llist.collapse*(*1*))
have *7*: ∀ *nell* ∈ *lset* (*?n2l nells*). ¬ *lnull* (*ltl nell*)
by (*simp add*: *6*)
have *8*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*2*) *nfinite-def* **by** *fastforce*
have *9*: *llastlfirst* (*?n2l nells*)
using *assms*(*3*) *nlastnfirst.rep-eq* **by** *auto*
have *10*: *lfinite* (*lfusecat* (*?n2l nells*)) ⟷ *lfinite* (*?n2l nells*)
using *7 8 9 lfusecat-lfinite* **by** *blast*
have *11*: *lfinite* (*?n2l nells*) ⟷ *nfinite nells*
by (*simp add*: *nfinite-def*)
show *?thesis* **using** *1 2 3 10 11* **by** *presburger*
qed

lemma *nlastfirst-nfusecat-nlast*:
assumes *nfinite nells*
nlastnfirst nells
∀ *nell* ∈ *nset nells*. *nfinite nell*
shows *nlast*(*nfusecat nells*) = *nlast*(*nlast nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlast*(*nfusecat nells*) =
nlast (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))
by (*simp add*: *nfusecat-def2*)
have *2*: *nlast* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)) =
llast (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)))
by (*metis* (*no-types*, *lifting*) *llast-linfinite nfinite-def nlast-llast nlast-not-nfinite*)
have *3*: *llast* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))) =

llast (*lfusecat* (*?n2l nells*))
by (*metis* (*no-types*, *lifting*) *2 comp-def lbutlast.disc-iff*(*1*) *nellist-of-llist-a-inverse nlast-nellist-of-llist-a-lnull not-lnull-eq-lappend-lbutlast-llast*)
have *4*: *lfinite* (*?n2l nells*)
using *assms*(*1*) *lfinite-lmap nfinite-def* **by** *auto*
have *5*: ¬*lnull* (*?n2l nells*)
by *simp*
have *6*: ∀ *nell* ∈ *lset* (*?n2l nells*). ¬*lnull nell*
by *simp*
have *7*: *llastlfirst* (*?n2l nells*)
using *assms*(*2*) *nlastnfirst.rep-eq* **by** *auto*
have *8*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*3*) *nfinite-def* **by** *auto*
have *9*: *llast* (*lfusecat* (*?n2l nells*)) = *llast*(*llast* (*?n2l nells*))
using *4 5 6 7 8 llastfirst-lfusecat-llast* **by** *blast*
have *10*: *llast*(*llast* (*?n2l nells*)) = *nlast* (*nlast nells*)
by (*metis assms*(*1*) *assms*(*3*) *comp-apply llast-lmap llist-of-nellist-not-lnull nfinite-def nlast-llast nset-nlast*)
show *?thesis* **using** *1 2 3 9 10* **by** *presburger*
qed

lemma *nfusecat-nlength-nfinite*:
assumes *nfinite nells*
∀ *nell* ∈ *nset nells*. *nfinite nell*
nlastnfirst nells
shows *nlength*(*nfusecat nells*) =
(∑ *i* = *0* .. (*the-enat*((*nlength nells*))) . (*nlength* (*nnth nells i*)))
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nlength*(*nfusecat nells*) = *nlength* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))
by (*simp add*: *nfusecat-def2*)
have *2*: *nlength* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)) =
epred (*llength* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))))
using *nlength.rep-eq* **by** *blast*
have *3*: *epred* (*llength* (*llist-of-nellist* (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)))) =
epred (*llength* (*lfusecat* (*?n2l nells*)))
using *one-eSuc plus-1-eSuc*(*2*) **by** *simp*
have *4*: *lfinite* (*?n2l nells*)
using *assms*(*1*) *lfinite-lmap nfinite-def* **by** *auto*
have *5*: ¬*lnull* (*?n2l nells*)
by *simp*
have *6*: ∀ *nell* ∈ *lset* (*?n2l nells*). ¬ *lnull nell*
by *simp*
have *7*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*2*) *nfinite-def* **by** *auto*
have *8*: *llastlfirst* (*?n2l nells*)
using *assms*(*3*) *nlastnfirst.rep-eq* **by** *auto*
have *9*: *epred* (*llength* (*lfusecat* (*?n2l nells*))) =
epred (*eSuc*(∑ *i* = *0* .. (*the-enat*(*epred*(*llength* (*?n2l nells*)))) .

epred(*llength* (*lnth* (*?n2l nells*) *i*))))
by (*metis 4 5 6 7 8 lfusecat-llength-lfinite*)
have *10*: *epred* (*eSuc*($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength* (*?n2l nells*)))) .
epred(*llength* (*lnth* (*?n2l nells*) *i*)))) =
(($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength* (*?n2l nells*)))) .
epred(*llength* (*lnth* (*?n2l nells*) *i*))))
using *epred-eSuc* **by** *blast*
have *11*: ((*epred*(*llength* (*?n2l nells*)))) = ((*nlength nells*))
by *simp*
have *12*: $\bigwedge$*j*. *j* $\leq$ ((*epred*(*llength* (*?n2l nells*))))$\longrightarrow$
epred(*llength* (*lnth* (*?n2l nells*) *j*)) = *nlength* (*nnth nells j*)
by (*metis 5 co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llength-llist-of-nellist llength-lmap lnth-llist-of-nellist lnth-lmap min-def the-enat.simps*)
have *13*: (($\sum$ *i* = *0* .. (*the-enat*(*epred*(*llength* (*?n2l nells*)))) .
epred(*llength* (*lnth* (*?n2l nells*) *i*)))) =
(($\sum$ *i* = *0* .. (*the-enat*(*nlength nells*)). *nlength* (*nnth nells i*)))
using *12 assms*(*1*) *nfinite-nlength-enat* **by** *force*
show *?thesis* **using** *1 2 3 9 10 13* **by** *metis*
qed

lemma *nlastnfirst-ntaken*:
assumes *n*$\leq$*nlength nells*
nlastnfirst nells
shows *nlastnfirst* (*ntaken n nells*)
using *assms*
by (*metis Suc-ile-eq antisym-conv2 nappend-ntaken-ndropn nfinite-ntaken nlastnfirst-nappend-nfinite ntaken-all*)

lemma *nlastnfirst-ntake*:
assumes *n*$\leq$*nlength nells*
nlastnfirst nells
shows *nlastnfirst* (*ntake n nells*)
proof (*cases n*)
case (*enat nat*)
then show *?thesis* **by** (*metis assms nlastnfirst-ntaken ntake-eq-ntaken*)
next
case *infinity*
then show *?thesis*
by (*simp add*: *assms ntake-all*)
qed

lemma *nfusecat-ntake*:
assumes (*enat n*) $\leq$ *nlength nells*
$\forall$ *nell* $\in$ *nset nells*. *nfinite nell*
nlastnfirst nells
shows *nfusecat* (*ntake n nells*) =

```
      ntake ( ((∑ i = 0 .. n . (nlength (nnth nells i)))))
           (nfusecat nells)
proof -
 let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
 have 1: nfusecat (ntake n  nells) =
       (((nellist-of-llist ∘ lfusecat ∘ ?n2l) (ntake n nells)))
   by (simp add: nfusecat-def2)
 have 2: (((nellist-of-llist ∘ lfusecat ∘ ?n2l) (ntake n nells))) =
        nellist-of-llist (lfusecat (?n2l (ntake n nells)))
    by simp
 have 3: llist-of-nellist(ntake n nells) = ltake (eSuc n) (llist-of-nellist nells)
     by (metis co.enat.distinct(1) llist-of-nellist-inverse-b llist-of-nellist-not-lnull
         ltake.disc(2) nellist-of-llist-inverse ntake.abs-eq)
 have 4: (?n2l (ntake n nells)) = ltake (eSuc n) (?n2l nells)
     using 3 by auto
 have 5: ¬ lnull (?n2l nells)
    by simp
 have 6:  (Suc n) ≤ llength (?n2l nells)
 by (metis 5 Suc-ile-eq assms(1) co.enat.collapse comp-apply iless-Suc-eq llength-eq-0 llength-lmap
     nlength.rep-eq)
 have 7:  ∀ nell ∈ lset (?n2l nells). ¬ lnull nell
    by simp
 have 8:  ∀ nell ∈ lset (?n2l nells). lfinite nell
   using assms(2) nfinite-def by fastforce
 have 9:    llastlfirst (?n2l nells)
   using assms(3) nlastnfirst.rep-eq by auto
 have 10: (lfusecat (ltake (eSuc n) (?n2l nells))) =
        ltake (if (Suc n) =0 then 0 else eSuc(∑ i = 0 .. n .
             epred(llength (lnth (?n2l nells) i))))
             (lfusecat (?n2l nells))
   using lfusecat-ltake[of (?n2l nells) Suc n ]
       5 6 7 8 9 diff-Suc-1 eSuc-enat by presburger
 have 11: (if (Suc n) =0 then 0 else eSuc(∑ i = 0 .. n .
            epred(llength (lnth (?n2l nells) i)))) =
         ( eSuc(∑ i = 0 .. n . epred(llength (lnth (?n2l nells) i))))
     using nat.simps(3) by presburger
 have 111: ⋀j. j ≤n ⟶ (min (enat j) (epred (llength (llist-of-nellist nells)))) = j
     by (metis assms(1) enat-ord-simps(1) llength-llist-of-nellist min.bounded-iff min-def)
 have 12: ⋀ j. j ≤n ⟶
          epred(llength (lnth (?n2l nells) j)) = nlength ( nnth nells j)
     using 5 111 assms lnth-llist-of-nellist[of nells]
     by (metis (full-types) co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lmap lnth-lmap
         min.order-iff nlength.rep-eq o-apply the-enat.simps)
 have 13: (lfusecat (?n2l nells)) = llist-of-nellist (nfusecat nells)
    by (simp add: lfusecat-not-lnull-var nfusecat-def2)
 have 14: ltake (if (Suc n) =0 then 0 else eSuc(∑ i = 0 .. n .
               epred(llength (lnth (?n2l nells) i))))
             (lfusecat (?n2l nells)) =
         ltake ( eSuc(∑ i = 0 .. n .  nlength ( nnth nells i))) (llist-of-nellist (nfusecat nells))
    using 12 13 by auto
```

have *15*: *nellist-of-llist* (*ltake* (*eSuc*($\sum$ *i* = *0* .. *n* .
nlength (*nnth nells i*))) (*llist-of-nellist* (*nfusecat nells*))) =
ntake (($\sum$ *i* = *0* .. *n* . *nlength* (*nnth nells i*))) (*nfusecat nells*)
by (*metis llist-of-nellist-inverse-b llist-of-nellist-not-lnull ntake.abs-eq*)
show *?thesis*
by (*metis 10 14 15 2 4 nfusecat-def2*)
qed

lemma *nfusecat-ndropn*:
assumes *n* < *nlength nells*
∀ *nell* ∈ *nset nells*. *nfinite nell*
nlastnfirst nells
shows *nfusecat* (*ndropn n nells*) =
ndropn (*the-enat*(*if n* = *0 then 0 else* ($\sum$ *i* = *0* .. (*n*−*1*) . *nlength*(*nnth nells i*))))
(*nfusecat nells*)
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nfusecat* (*ndropn n nells*) =
(((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*ndropn n nells*)))
by (*simp add*: *nfusecat-def2*)
have *2*: (((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) (*ndropn n nells*))) =
nellist-of-llist (*lfusecat* (*lmap llist-of-nellist* (*ldropn n* (*llist-of-nellist nells*))))
using *assms* **by** *simp*
have *4*: *nellist-of-llist* (*lfusecat* (*lmap llist-of-nellist* (*ldropn n* (*llist-of-nellist nells*)))) =
nellist-of-llist (*lfusecat* (*lmap llist-of-nellist* (*ldrop n* (*llist-of-nellist nells*))))
by (*simp add*: *ldrop-enat*)
have *5*: (*lmap llist-of-nellist* (*ldrop n* (*llist-of-nellist nells*))) = *ldrop n* (*?n2l nells*)
by (*simp*)
have *6*: ¬ *lnull* (*?n2l nells*)
by *simp*
have *7*: *n* < *llength* (*?n2l nells*)
by (*metis 5 assms*(*1*) *ldrop-enat llist.map-disc-iff llist-of-nellist-ndropn llist-of-nellist-not-lnull lnull-ldrop min-def nlength.rep-eq not-le-imp-less order-less-imp-le the-enat.simps*)
have *8*: ∀ *nell* ∈ *lset* (*?n2l nells*). ¬ *lnull nell*
by *simp*
have *9*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*2*) *nfinite-def* **by** *fastforce*
have *10*: *llastlfirst* (*?n2l nells*)
using *assms*(*3*) *nlastnfirst.rep-eq* **by** *auto*
have *11*: *lfusecat* (*ldrop* (*enat n*) (*?n2l nells*)) =
ldrop (*if n* = *0 then 0 else* ($\sum$ *i* = *0* .. (*n*−*1*) .
epred(*llength* (*lnth* (*?n2l nells*) *i*))))
(*lfusecat* (*?n2l nells*))
using *10 6 7 8 9 lfusecat-ldrop* **by** *blast*
have *111*: $\bigwedge$*j*. *0*< *j* ∧ *j* <*n*−*1* ⟶
(*min* (*enat j*) (*epred* (*llength* (*llist-of-nellist nells*)))) = *j*
by (*metis 7 comp-apply epred-enat epred-min less-imp-of-nat-less llength-lmap min.absorb3 min-less-iff-conj of-nat-eq-enat*)
have *112*: *n*−*1* < *llength* (*?n2l nells*)
by (*metis 7 One-nat-def Suc-ile-eq Suc-pred epred-0 epred-enat not-gr-zero order-less-imp-le zero-enat-def*)

```
have 12: ⋀j. 0< n ∧ j < n−1 ⟶
            epred(llength (lnth (?n2l nells) j)) = nlength(nnth nells j)
  using 6 7 111  lnth-llist-of-nellist[of nells]
  by simp
   (metis canonically-ordered-monoid-add-class.lessE diff-le-self dual-order.strict-trans1
    enat-less-enat-plusI enat-min-eq-0-iff lnth-lmap nlength.rep-eq not-gr-zero the-enat.simps
    zero-enat-def)
have 13: (lfusecat (?n2l nells)) = llist-of-nellist (nfusecat nells)
  by (simp add: lfusecat-not-lnull-var nfusecat-def2)
have 131: ⋀j . j< n ⟹ (min (enat j) (epred (llength (llist-of-nellist nells)))) =j
     by (metis (full-types) assms(1) min.assoc min.orderE min-enat-simps(1) nlength.rep-eq
        order-less-imp-le)
have 132: (if n = 0 then 0 else ∑ i = 0..n − 1. epred (llength (lnth (?n2l nells) i))) =
         (if n = 0 then 0 else (∑  i = 0 .. (n−1) . nlength(nnth nells i)))
    using sum.cong[of {0..n−1} {0..n−1} λi. epred (llength (lnth (?n2l nells) i))
         λi. nlength(nnth nells i)  ] assms 7 131 12 lnth-llist-of-nellist[of nells]
   by (metis 112 atLeastAtMost-iff comp-apply diff-less less-numeral-extra(1) llength-lmap
        lnth-lmap nlength.rep-eq not-gr-zero order-neq-le-trans the-enat.simps)
have 14: ldrop (if n = 0 then 0 else (∑  i = 0 .. (n−1) . epred(llength (lnth (?n2l nells) i))))
             (lfusecat (?n2l nells)) =
         ldrop (if n = 0 then 0 else (∑  i = 0 .. (n−1) . nlength(nnth nells i))) (llist-of-nellist (nfusecat
nells))
      using 13 132 by presburger
have 15: 0<n ⟹ (⋀ j.  j ≤n−1  ⟶  epred(llength (lnth (?n2l nells) j)) < ∞)
    using 7 9
    by (metis (no-types, opaque-lifting) 112 enat-ord-simps(1) enat-ord-simps(4) epred-llength
       in-lset-conv-lnth lfinite-ltl llength-eq-infty-conv-lfinite order-le-less-trans)
have 16: 0< n ⟹ ((∑  i = 0 .. (n−1) . epred(llength (lnth (?n2l nells) i)))) < ∞
  using 15
   proof (induct n)
   case 0
   then show ?case by simp
   next
   case (Suc n)
   then show ?case
     proof (cases n=0)
     case True
     then show ?thesis using Suc by simp
     next
     case False
     then show ?thesis using Suc
     by simp
       (metis One-nat-def Suc-pred′ dual-order.refl plus-enat-simps(1) sum.atLeast0-atMost-Suc)
     qed
   qed
 have 17: ∃ m. (enat m) = (if n = 0 then 0 else (∑  i = 0 .. (n−1) .
          epred(llength (lnth (?n2l nells) i))))
   by (metis 16 less-infinityE not-gr-zero zero-enat-def)
 obtain m where 18: (enat m) = (if n = 0 then 0 else
```

($\sum$ $i = 0$ .. $(n-1)$. *epred*(*llength* (*lnth* (*?n2l nells*) *i*))))
using *17* **by** *blast*
have *19*: *ldrop* (*if* $n = 0$ *then* 0 *else*
($\sum$ $i = 0$ .. $(n-1)$. *epred*(*llength* (*lnth* (*?n2l nells*) *i*))))
(*lfusecat* (*?n2l nells*)) =
ldropn m (*lfusecat* (*?n2l nells*))
using *18 ldrop-enat*[*of m* (*lfusecat* (*?n2l nells*))]
by *presburger*
have *20*: *enat m* $\leq$ *epred* (*llength* (*lfusecat* (*?n2l nells*)))
by (*metis* (*no-types*, *lifting*) *11 19 6 7 8 co.enat.exhaust-sel iless-Suc-eq in-lset-ldropD lfusecat-not-lnull-var linorder-not-le llength-eq-0 lnull-ldrop lnull-ldropn*)
have *21*: *nellist-of-llist* (*ldropn m* (*lfusecat* (*?n2l nells*))) = *ndropn m* (*nfusecat nells*)
using *20*
by (*metis 13 llist-of-nellist-inverse-b llist-of-nellist-not-lnull min-def ndropn.abs-eq the-enat.simps*)
have *22*: m = (*the-enat*(*if* $n = 0$ *then* 0 *else* ($\sum$ $i = 0$ .. $(n-1)$. *nlength*(*nnth nells i*))))
by (*metis* (*mono-tags*, *lifting*) *132 18 the-enat.simps*)
show *?thesis* **using** *1 2 4 5 11 19 21 22* **by** (*metis*)
qed

lemma *nridx-nfusecat-ntake*:
assumes *nridx R* (*nfusecat nells*)
(*enat n*) $\leq$ *nlength nells*
nlastnfirst nells
$\forall$ *nell* $\in$ *nset nells*. *nfinite nell*
shows *nridx R* (*nfusecat* (*ntake n nells*))
using *assms* **by** (*metis nfusecat-ntake nle-le nridx-ntake-a ntake-all*)

lemma *nridx-nfusecat-ndrop*:
assumes *nridx R* (*nfusecat nells*)
(*enat n*) < *nlength nells*
nlastnfirst nells
$\forall$ *nell* $\in$ *nset nells*. *nfinite nell*
shows *nridx R* (*nfusecat* (*ndropn n nells*))
proof –
let *?n2l* = (*lmap llist-of-nellist* $\circ$ *llist-of-nellist*)
have *1*: *nridx R* (*nfusecat* (*ndropn n nells*)) $\longleftrightarrow$
nridx R ((*nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*) (*ndropn n nells*))
by (*simp add*: *nfusecat-def2*)
have *2*: *nridx R* ((*nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*) (*ndropn n nells*)) $\longleftrightarrow$
ridx R (*llist-of-nellist* ((*nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*) (*ndropn n nells*)))
using *nridx.rep-eq* **by** *blast*
have *3*: *ridx R* (*llist-of-nellist* ((*nellist-of-llist* $\circ$ *lfusecat* $\circ$ *?n2l*) (*ndropn n nells*))) $\longleftrightarrow$
ridx R (((*lfusecat* $\circ$ *?n2l*) (*ndropn n nells*)))
using *not-null-lfusecat*
by (*metis* (*no-types*, *lifting*) *comp-apply nridx.abs-eq nridx.rep-eq*)
have *4*: *llastlfirst* (*?n2l nells*)
using *assms nlastnfirst.rep-eq* **by** *auto*
have *5*: $\forall$ *nell* $\in$ *lset* (*?n2l nells*). *lfinite nell*
using *assms nfinite-def* **by** *fastforce*

have *6*: (((*lfusecat* ∘ *?n2l*) (*ndropn n nells*))) =
lfusecat (*lmap llist-of-nellist* (*ldropn n* (*llist-of-nellist nells*)))
by (*simp add*: *assms*(*2*))
have *7*: (*lmap llist-of-nellist* (*ldropn n* (*llist-of-nellist nells*))) =
ldropn n (*lmap llist-of-nellist* (*llist-of-nellist nells*))
by *auto*
have *8*: *ridx R* (*lfusecat* (*?n2l nells*))
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *comp-apply nfusecat-def2 nridx.abs-eq ridx-expand-1*)
have *9*: *enat n* < *llength* (*?n2l nells*)
by (*metis assms*(*2*) *co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llength-lmap llist-of-nellist-not-lnull nlength.rep-eq order-less-imp-le*)
have *10*: *ridx R* (*lfusecat* (*ldropn n* (*?n2l nells*)))
using *ridx-lfusecat-ldrop*[*of R* (*?n2l nells*) *n*]
by (*metis* (*mono-tags*, *lifting*) *4 5 8 9 comp-apply in-lset-conv-lnth ldrop-enat llength-lmap llist-of-nellist-not-lnull lnth-lmap*)
show *?thesis*
using *1 10 2 3 6 7* **by** (*metis comp-apply*)
qed

lemma *nridx-nfusecat*:
assumes *nlastnfirst nells*
∀ *nell* ∈ *nset nells*. *0* < *nlength nell*
∀ *nell* ∈ *nset nells*. *nfinite nell*
shows *nridx R* (*nfusecat nells*) ⟷ (∀ *i*. *i* ≤ *nlength nells* ⟶ *nridx R* (*nnth nells i*))
proof −
let *?n2l* = (*lmap llist-of-nellist* ∘ *llist-of-nellist*)
have *1*: *nridx R* (*nfusecat nells*) ⟷
nridx R ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)
by (*simp add*: *nfusecat-def*)
have *2*: *nridx R* ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*) ⟷
ridx R (*llist-of-nellist* ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*))
using *nridx.rep-eq* **by** *blast*
have *3*: *ridx R* (*llist-of-nellist* ((*nellist-of-llist* ∘ *lfusecat* ∘ *?n2l*) *nells*)) ⟷
ridx R (((*lfusecat* ∘ *?n2l*) *nells*))
using *not-null-lfusecat nridx.abs-eq* **by** *fastforce*
have *4*: *llastlfirst* (*?n2l nells*)
using *assms*(*1*) *nlastnfirst.rep-eq* **by** *auto*
have *5*: ∀ *nell* ∈ *lset* (*?n2l nells*). *1* < *llength nell*
by (*metis assms*(*2*) *epred-1 comp-apply imageE less-numeral-extra*(*3*) *llist.set-map llist-of-nellist-not-lnull lset-llist-of-nellist-a nlength.rep-eq not-lnull-llength order-neq-le-trans*)
have *6*: ∀ *nell* ∈ *lset* (*?n2l nells*). *lfinite nell*
using *assms*(*3*) *nfinite-def* **by** *fastforce*
have *7*: *ridx R* (((*lfusecat* ∘ *?n2l*) *nells*)) ⟷
(∀ *i*. *i* < *llength* (*?n2l nells*) ⟶ *ridx R* (*lnth* (*?n2l nells*) *i*))
using *4 5 6 ridx-lfusecat* **by** *fastforce*
have *8*: *epred*(*llength* (*?n2l nells*)) = *nlength nells*
by *simp*
have *9*: ⋀*j*. *j* < *llength* (*?n2l nells*) ⟶

(*lnth* (*?n2l nells*) *j*) = *llist-of-nellist*(*nnth nells j*)
by *simp*
(*metis eSuc-epred eSuc-ile-mono gr-implies-not-zero ileI1 lnth-llist-of-nellist min-def the-enat.simps*)
have *10*: (∀ *i*. *i* < *llength* (*?n2l nells*) ⟶ *ridx R* (*lnth* (*?n2l nells*) *i*)) ⟷
(∀ *i*. *i* ≤ *nlength* (*nells*) ⟶ *nridx R* (*nnth nells i*))
by (*metis* (*mono-tags*, *lifting*) *8 9 co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llist.map-disc-iff llist-of-nellist-not-lnull nridx.rep-eq*)
show *?thesis* **using** *1 2 3 10 7* **by** *blast*
qed

lemma *nfusecat-split-nsubn*:
assumes *nlastnfirst nells*
∀ *nell* ∈ *nset nells*. *0* < *nlength nell*
∀ *nell* ∈ *nset nells*. *nfinite nell*
shows (∀ *i*. *i*≤ *nlength nells* ⟶ *nridx* (λ *a b*. *f* (*nsubn* σ *a b*)) (*nnth nells i*)) ⟷
nridx (λ *a b*. *f* (*nsubn* σ *a b*)) (*nfusecat nells*)
using *assms nridx-nfusecat* **by** *blast*

lemma *nidx-nfusecat*:
assumes *nlastnfirst nells*
∀ *nell* ∈ *nset nells*. *0* < *nlength nell*
∀ *nell* ∈ *nset nells*. *nfinite nell*
shows *nidx* (*nfusecat nells*) ⟷ (∀ *i*. *i* ≤ *nlength nells* ⟶ *nidx* (*nnth nells i*))
using *assms nridx-nfusecat nridx-nidx* **by** *blast*

lemma *nnth-sum-expand*:
assumes *nlastnfirst nells*
∀ *nell* ∈ *nset nells*. *0* < *nlength nell*
∀ *nell* ∈ *nset nells*. *nfinite nell*
(*enat i*) ≤ *nlength nells*
nfinite nells
shows (∑ *i* = *0* .. (*the-enat*((*nlength nells*))) . (*nlength* (*nnth nells i*))) =
(*nlength* (*nnth nells i*)) +
(∑ *j* ∈ {*k*. *k*≠*i* ∧ *k*≤ (*the-enat*((*nlength nells*)))} . (*nlength* (*nnth nells j*)))
proof −
have *1*: {*k*. *k*≤ (*the-enat*((*nlength nells*)))} = *insert i* {*k*. *k*≠*i* ∧ *k*≤ (*the-enat*((*nlength nells*)))}
using *assms* **by** *auto*
(*metis enat-ord-simps*(*1*) *nfinite-nlength-enat the-enat.simps*)
have *2*: {*0*.. (*the-enat*((*nlength nells*)))} = {*k*. *k*≤ (*the-enat*((*nlength nells*)))}
by *auto*
have *3*: (∑ *i* = *0* .. (*the-enat*((*nlength nells*))) . (*nlength* (*nnth nells i*))) =
(∑ *i* ∈{*k*. *k*≤ (*the-enat*((*nlength nells*)))} . (*nlength* (*nnth nells i*)))
using *2* **by** *presburger*
have *4*: (∑ *i* ∈{*k*. *k*≤ (*the-enat*((*nlength nells*)))} .(*nlength* (*nnth nells i*))) =
(∑ *j* ∈(*insert i* {*k*. *k*≠*i* ∧ *k*≤ (*the-enat*((*nlength nells*)))}) . (*nlength* (*nnth nells j*)))
using *1* **by** *presburger*
have *5*: (∑ *j* ∈(*insert i* {*k*. *k*≠*i* ∧ *k*≤ (*the-enat*((*nlength nells*)))}) . (*nlength* (*nnth nells j*))) =
(*nlength* (*nnth nells i*)) +
(∑ *j* ∈ {*k*. *k*≠*i* ∧ *k*≤ (*the-enat*((*nlength nells*)))} . (*nlength* (*nnth nells j*)))

by *auto*
show *?thesis*
using *3 4 5* **by** *presburger*
qed

lemma *nfusecat-nlength-a*:
assumes *nlastnfirst nells*
$\forall$ *nell* $\in$ *nset nells*. *0* $<$ *nlength nell*
$\forall$ *nell* $\in$ *nset nells*. *nfinite nell*
i $\leq$ *nlength nells*
(*enat j*) $\leq$ *nlength* (*nnth nells i*)
shows (*enat j*) $\leq$ *nlength* (*nfusecat nells*)
proof (*cases nfinite nells*)
case *True*
then show *?thesis*
proof −
have *2*: *nlength* (*nfusecat nells*) =
($\sum$ *i* = *0* .. (*the-enat*((*nlength nells*))) . (*nlength* (*nnth nells i*)))
using *True assms nfusecat-nlength-nfinite* **by** *fastforce*
have *3*: *nlength* (*nnth nells i*) $\leq$
($\sum$ *i* = *0* .. (*the-enat*((*nlength nells*))) . (*nlength* (*nnth nells i*)))
using *nnth-sum-expand*[*of nells i*] *assms*
by (*metis* (*no-types*, *lifting*) *True le-iff-add*)
show *?thesis* **using** *assms 2 3* **by** *force*
qed
next
case *False*
then show *?thesis*
proof −
have *5*: $\neg$ *nfinite* (*nfusecat nells*)
using *False assms nfusecat-nfinite* **by** *blast*
show *?thesis*
using *5*
by (*simp add*: *nfinite-conv-nlength-enat*)
qed
qed

3.9 nkfilter

lemma *nkfilter-NNil* [*simp*]:
shows *P b* $\Longrightarrow$ *nkfilter P n* (*NNil b*) = (*NNil n*)
by *transfer auto*

lemma *nkfilter-NCons* [*simp*]:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
shows *nkfilter P n* (*NCons x nell*) =
(*if P x then NCons n* (*nkfilter P* (*Suc n*) *nell*) *else nkfilter P* (*Suc n*) *nell*)
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv*)

lemma *nkfilter-NCons-a* [*simp*]:
assumes $\neg(\exists\ x \in nset\ nell.\ P\ x)$
$P\ x$
shows $nkfilter\ P\ n\ (NCons\ x\ nell) = (NNil\ n)$
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv*)

lemma *nkfilter-nlength*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows $nlength(nkfilter\ P\ n\ nell) = nlength(nfilter\ P\ nell)$
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv kfilter-llength*)

lemma *nkfilter-upperbound*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $nnth\ (nkfilter\ P\ n\ nell)\ i \leq n + nlength\ nell$
using *assms*
by *transfer*
(*auto*,
simp add: *kfilter-lnull-conv*,
metis co.enat.exhaust-sel iadd-Suc-right iless-Suc-eq kfilter-upperbound llength-eq-0)

lemma *nkfilter-lowerbound*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $n \leq nnth\ (nkfilter\ P\ n\ nell)\ i$
using *assms*
by *transfer*
(*auto*,
metis co.enat.collapse iless-Suc-eq kfilter-lnth-n-zero-a llength-eq-0)

lemma *nkfilter-mono*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$(Suc\ i) \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $nnth\ (nkfilter\ P\ n\ nell)\ i < nnth\ (nkfilter\ P\ n\ nell)\ (Suc\ i)$
using *assms*
by *transfer*
(*auto*,
metis diff-Suc-1 eSuc-epred epred-enat epred-le-epredI lidx-kfilter-gr iless-Suc-eq leD lessI
llength-eq-0 min.cobounded1 min-def the-enat.simps)

lemma *nkfilter-nfilter*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $(nnth\ nell\ ((nnth\ (nkfilter\ P\ n\ nell)\ i) - n)) = (nnth\ (nfilter\ P\ nell)\ i)$
using *assms*

apply *transfer*
proof (*auto simp add*: *kfilter-lnull-conv split*:*if-split-asm*)
fix *nella* :: *'a llist* **and** *Pa* :: *'a* ⇒ *bool* **and** *ia* :: *nat* **and** *na* :: *nat* **and** *x* :: *'a*
assume *a1*: *x* ∈ *lset nella*
assume *a2*: *Pa x*
assume *a3*: *enat ia* ≤ *epred* (*llength* (*kfilter Pa na nella*))
assume *a4*: ¬ *lnull nella*
have *f5*: ∀ *e ea*. ¬ (*e*::*enat*) ≤ *ea* ∨ *min e ea* = *e*
by (*simp add*: *min-def-raw*)
have *f6*: ∀ *n p na as*. ¬ *enat n* < *llength* (*kfilter p na* (*as*::*'a llist*)) ∨
enat (*lnth* (*kfilter p na as*) *n*) < *enat na* + *llength as*
by (*meson kfilter-upperbound*)
have *f7*: *min* (*enat ia*) (*epred* (*llength* (*kfilter Pa na nella*))) = *enat ia*
using *f5 a3* **by** *blast*
then have *f8*: (*enat ia* < *llength* (*kfilter Pa 0 nella*)) =
(*enat* (*the-enat* (*min* (*enat ia*) (*epred* (*llength* (*kfilter Pa na nella*))))) <
llength (*kfilter Pa na nella*))
by (*simp add*: *kfilter-llength*)
have *f9*: ∀ *n p na as*. ¬ *enat n* < *llength* (*kfilter p na* (*as*::*'a llist*)) ∨
lnth (*kfilter p na as*) *n* − *na* = *lnth* (*kfilter p 0 as*) *n*
using *kfilter-lnth-n-zero* **by** *blast*
have *eSuc* (*epred* (*llength nella*)) = *enat 0* + *llength nella*
using *a4* **by** (*metis co.enat.exhaust-sel gen-llength-def llength-code llength-eq-0*)
then have *enat* (*the-enat* (*min* (*enat ia*) (*epred* (*llength* (*kfilter Pa na nella*))))) <
llength (*kfilter Pa na nella*) ⟶
enat (*lnth* (*kfilter Pa na nella*)
(*the-enat* (*min* (*enat ia*) (*epred* (*llength* (*kfilter Pa na nella*))))) − *na*)
< *eSuc* (*epred* (*llength nella*))
using *f9 f8 f7 f6 the-enat.simps* **by** *presburger*
then have *f10*: *enat* (*the-enat* (*min* (*enat ia*) (*epred* (*llength* (*kfilter Pa na nella*))))) <
llength (*kfilter Pa na nella*) ⟶
enat (*lnth* (*kfilter Pa na nella*)
(*the-enat* (*min* (*enat ia*) (*epred* (*llength* (*kfilter Pa na nella*))))) − *na*)
≤ *epred* (*llength nella*)
using *iless-Suc-eq* **by** *blast*
have *f11*: ∀ *n p na as*. ¬ *enat n* < *llength* (*kfilter p na as*) ∨ *lnth as* (*lnth* (*kfilter p na as*) *n* − *na*) =
(*lnth* (*lfilter p as*) *n*::*'a*)
using *lfilter-kfilter* **by** *blast*
have *f12*: *the-enat* (*min* (*enat ia*) (*epred* (*llength* (*lfilter Pa nella*)))) =
the-enat (*min* (*enat ia*) (*epred* (*llength* (*kfilter Pa na nella*))))
by (*simp add*: *kfilter-llength*)
have ¬ *lnull* (*kfilter Pa na nella*)
using *a1 a2 kfilter-not-lnull-conv* **by** *auto*
then have *enat* (*the-enat* (*min* (*enat ia*) (*epred* (*llength* (*kfilter Pa na nella*))))) <
llength (*kfilter Pa na nella*)
using *f7 a3* **by** (*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 the-enat.simps*)
then show ¬ *lnull nella* ⟹
enat ia ≤ *epred* (*llength* (*kfilter Pa na nella*)) ⟹
x ∈ *lset nella* ⟹
Pa x ⟹

lnth nella (*the-enat* (*min* (*enat* (*lnth* (*kfilter Pa na nella*) *ia* − *na*)) (*epred* (*llength nella*)))) =
lnth (*lfilter Pa nella*) (*the-enat* (*min* (*enat ia*) (*epred* (*llength* (*lfilter Pa nella*)))))
using *f12 f11 f10 f5* **by** *simp*
qed

lemma *in-nkfilter-nset*:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows *x* ∈ *nset*(*nkfilter P n nell*) ⟷ *x* ∈ { *n*+*i* | *i*. *i* ≤ *nlength nell* ∧ *P* (*nnth nell i*)}
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv min-def*,
metis co.enat.exhaust-sel gen-llength-def iless-Suc-eq kfilter-holds kfilter-llength-n-zero
kfilter-lnth-n-zero kfilter-lowerbound kfilter-upperbound ldistinct-Ex1 ldistinct-kfilter
le-add-diff-inverse llength-code llength-eq-0,
metis add.commute co.enat.exhaust-sel iless-Suc-eq kfilter-holds-not-a llength-eq-0)

lemma *nkfilter-nset*:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows *nset*(*nkfilter P n nell*) = { *n*+*i* | *i*. *i* ≤ *nlength nell* ∧ *P* (*nnth nell i*)}
using *assms in-nkfilter-nset*[*of nell P - n*] **by** *blast*

lemma *nlength-nkfilter-le* [*simp*]:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows *nlength* (*nkfilter P n nell*) ≤ *nlength nell*
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv epred-le-epredI kfilter-llength llength-lfilter-ile*)

lemma *nkfilter-nleast*:
assumes ∃ *x* ∈ *nset xs*. *P x*
shows *nnth* (*nkfilter P n xs*) *0* = *n*+(*nleast P xs*)
using *assms*
by *transfer*
(*auto simp add*: *kfilter-not-lnull-conv kfilter-lnull-conv*,
metis enat-min-eq-0-iff kfilter-lnth-zero kfilter-lnull-conv the-enat-0 zero-enat-def)

lemma *ndistinct-nkfilter*:
assumes ∃ *x* ∈ *nset nell*. *P x*
shows *ndistinct*(*nkfilter P n nell*)
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv ldistinct-kfilter*)

lemma *nkfilter-ndropn-nset*:
assumes ∃ *x* ∈ *nset* (*ndropn k nell*). *P x*
k ≤ *nlength nell*
shows *nset*(*nkfilter P n* (*ndropn k nell*)) = { *n*+*i*|*i*. *i*≤*nlength nell* − *k* ∧ *P* (*nnth nell* (*k*+*i*))}
using *assms nkfilter-nset*[*of* (*ndropn k nell*) *P n*]
ndropn-nnth[*of - nell k*] **by** *auto*

lemma *nkfilter-ndropn-nset-b*:
assumes $\exists\ x \in nset\ (ndropn\ k\ nell).\ P\ x$
$k \leq nlength\ nell$
shows $nset(nkfilter\ P\ n\ (ndropn\ k\ nell)) = \{\ n+\ i\ |i.\ i\leq\ nlength\ nell\ -\ k\ \ \wedge\ P\ (nnth\ nell\ (i+k))\}$
proof −
have *1*: $\{\ n+i|i.\ i\leq nlength\ nell\ -\ k\ \wedge\ P\ (nnth\ nell\ (k+i))\} =$
$\{\ n+\ i\ |i.\ i\leq\ nlength\ nell\ -\ k\ \ \wedge\ P\ (nnth\ nell\ (i+k))\}$
using *assms* **by** (*auto simp add*: *add.commute*)
show *?thesis* **using** *assms* **by** (*simp add*: *1 nkfilter-ndropn-nset*)
qed

lemma *nfilter-nkfilter-ntaken-nlength-0*:
assumes $P\ (nnth\ nell\ (\ (nnth\ (nkfilter\ P\ n\ nell)\ k)\ -n\ ))$
$k \leq nlength\ (nfilter\ P\ nell)$
shows $(\exists\ x \in nset\ nell.\ P\ x)$
using *assms*
by (*metis enat-ile in-nset-conv-nnth linorder-le-cases ntaken-all ntaken-nlast*)

lemma *nkfilter-nlength-n-zero*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
shows $nlength(nkfilter\ P\ n\ nell) = nlength(nkfilter\ P\ 0\ nell)$
using *assms* **by** (*simp add*: *nkfilter-nlength*)

lemma *nkfilter-nnth-n-zero*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$k \leq nlength(nkfilter\ P\ n\ nell)$
shows $(nnth\ (nkfilter\ P\ n\ nell)\ k) - n = (nnth\ (nkfilter\ P\ 0\ nell)\ k)$
using *assms*
by *transfer*
(*auto split*: *if-split-asm*,
metis kfilter-lnull-conv,
metis kfilter-lnull-conv,
metis co.enat.exhaust-sel iless-Suc-eq kfilter-llength-n-zero kfilter-lnth-n-zero llength-eq-0
min.orderE the-enat.simps)

lemma *nkfilter-n-zero*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
shows $(nkfilter\ P\ n\ nell) = nmap\ (\lambda i.\ i+n)\ (nkfilter\ P\ 0\ nell)$
proof −
have *1*: $nlength(nkfilter\ P\ n\ nell) = nlength\ (nmap\ (\lambda i.\ i+n)\ (nkfilter\ P\ 0\ nell))$
using *assms nkfilter-nlength-n-zero* **by** *fastforce*
have *2*: $\bigwedge\ k.\ k \leq nlength\ (nkfilter\ P\ n\ nell) \longrightarrow$
$nnth\ (nkfilter\ P\ n\ nell)\ k = nnth\ (nmap\ (\lambda i.\ i+n)\ (nkfilter\ P\ 0\ nell))\ k$
using *1 assms nkfilter-nnth-n-zero*[*of nell P - n*]
by (*metis diff-add nkfilter-lowerbound nlength-nmap nnth-nmap*)
show *?thesis* **by** (*simp add*: *1 2 nellist-eq-nnth-eq*)
qed

lemma *nkfilter-n-zero-a*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$

shows $(nkfilter\ P\ 0\ nell) = nmap\ (\lambda i.\ i-n)\ (nkfilter\ P\ n\ nell)$
proof −
have *1*: $nlength(nkfilter\ P\ 0\ nell) = nlength\ (nmap\ (\lambda i.\ i-n)\ (nkfilter\ P\ n\ nell))$
by (*metis assms nkfilter-nlength-n-zero nlength-nmap*)
have *2*: $\bigwedge k.\ k \le nlength(nkfilter\ P\ 0\ nell) \longrightarrow$
$nnth\ (nkfilter\ P\ 0\ nell)\ k = nnth\ (nmap\ (\lambda i.\ i-n)\ (nkfilter\ P\ n\ nell))\ k$
by (*simp add*: *1 assms nkfilter-nnth-n-zero*)
show *?thesis* **by** (*simp add*: *1 2 nellist-eq-nnth-eq*)
qed

lemma *nkfilter-holds*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$y \in nset(nkfilter\ P\ n\ nell)$
shows $P\ (nnth\ nell\ (y-n))$
using *assms in-nkfilter-nset*[*of nell P*] **by** *force*

lemma *nkfilter-holds-not*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$y \in \{i+n\ |i.\ i \le nlength\ nell\} - (nset\ (nkfilter\ P\ n\ nell))$
shows $\neg\ P\ (nnth\ nell\ (y-n))$
using *assms nkfilter-nset*[*of nell P n*] **by** *auto*

lemma *nkfilter-holds-a*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$i \le nlength\ nell$
$(i+n) \in nset(nkfilter\ P\ n\ nell)$
shows $P\ (nnth\ nell\ i)$
using *assms nkfilter-holds*[*of nell P i+n n*] **by** *simp*

lemma *nkfilter-holds-not-a*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$i \le nlength\ nell$
$P\ (nnth\ nell\ i)$
shows $(i+n) \in\ nset(nkfilter\ P\ n\ nell)$
using *assms* **by** (*simp add*: *in-nkfilter-nset*)

lemma *nkfilter-holds-b*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$i \le nlength\ nell$
shows $(i+n) \in nset(nkfilter\ P\ n\ nell) = P\ (nnth\ nell\ i)$
using *assms* **by** (*meson nkfilter-holds-a nkfilter-holds-not-a*)

lemma *nkfilter-holds-c*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$n \le i$
$i-n \le nlength\ nell$
shows $i \in nset(nkfilter\ P\ n\ nell) = P\ (nnth\ nell\ (i-n))$
using *assms*
by (*metis diff-add idiff-enat-enat nkfilter-holds-b*)

lemma *nkfilter-holds-not-b*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$n \le i$
$i-n \le nlength\ nell$
shows $i \notin nset\ (nkfilter\ P\ n\ nell) = (\neg\ P\ (nnth\ nell\ (i-n)))$
using *assms*
by (*simp add*: *nkfilter-holds-c*)

lemma *nkfilter-disjoint-nset-coset*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
shows $(\{i+n|i.\ i \le nlength\ nell\} - nset(nkfilter\ P\ n\ nell)) \cap nset\ (nkfilter\ P\ n\ nell) = \{\}$
using *assms* **by** (*simp add*: *inf.commute*)

lemma *nidx-nkfilter-expand*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$(Suc\ i) \le nlength(nkfilter\ P\ n\ nell)$
shows $nnth\ (nkfilter\ P\ n\ nell)\ i < nnth\ (nkfilter\ P\ n\ nell)\ (Suc\ i)$
using *assms* **by** (*simp add*: *nkfilter-mono*)

lemma *nidx-nkfilter*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
shows *nidx* (*nkfilter P n nell*)
using *assms nidx-nkfilter-expand*[*of nell P - n*] *nidx-expand*[*of* (*nkfilter P n nell*)]
by *blast*

lemma *nidx-nkfilter-gr-eq*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$k \le j$
$j \le nlength\ (nkfilter\ P\ n\ nell)$
shows $nnth\ (nkfilter\ P\ n\ nell)\ k \le nnth\ (nkfilter\ P\ n\ nell)\ j$
using *assms nidx-nkfilter*[*of nell P n*] *nidx-less-eq*[*of nkfilter P n nell k j*]
by *blast*

lemma *nidx-nkfilter-gr*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
shows $(\forall\ j.\ k<j \wedge j \le nlength\ (nkfilter\ P\ n\ nell) \longrightarrow$
$nnth\ (nkfilter\ P\ n\ nell)\ k < nnth(nkfilter\ P\ n\ nell)\ j)$
using *assms nidx-nkfilter*[*of nell P n*] *nidx-less*[*of nkfilter P n nell*]
using *less-imp-Suc-add* **by** *blast*

lemma *nidx-nkfilter-less-eq*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$k \le nlength\ (nkfilter\ P\ n\ nell)$
shows $\forall j.\ j \le k \longrightarrow nnth\ (nkfilter\ P\ n\ nell)\ j \le nnth\ (nkfilter\ P\ n\ nell)\ k$
using *assms* **by** (*simp add*: *nidx-nkfilter-gr-eq*)

lemma *nkfilter-not-before*:
assumes $(\exists\ x \in nset\ nell.\ P\ x)$
$i < (nnth\ (nkfilter\ P\ 0\ nell)\ 0)$

shows ¬ *P* (*nnth nell i*)
using *assms*
by *transfer*
(*auto simp add*: *min-def split*: *if-split-asm*,
meson kfilter-not-before llength-eq-0 not-gr-zero,
metis dual-order.strict-trans kfilter-not-before less-enatE llength-eq-0 not-gr-zero not-le-imp-less the-enat.simps,
meson le-less-linear less-enatE less-nat-zero-code,
meson le-less-linear less-enatE not-less0)

lemma *nkfilter-n-not-before*:
assumes ($\exists$ *x* $\in$ *nset* (*ndropn n nell*). *P x*)
n $\leq$ *nlength nell*
n $\leq$ *i*
i $<$ (*nnth* (*nkfilter P n* (*ndropn n nell*)) *0*)
shows ¬ *P* (*nnth nell i*)
using *assms*
apply *transfer*
unfolding *min-def*
using *zero-enat-def*
proof (*auto split*: *if-split-asm*)
show $\bigwedge$*n nell P i x*.
enat 0 = *0* $\Longrightarrow$
¬ *lnull nell* $\Longrightarrow$
enat n $\leq$ *epred* (*llength nell*) $\Longrightarrow$
n $\leq$ *i* $\Longrightarrow$
¬ *lnull* (*kfilter P n* (*ldropn n nell*)) $\Longrightarrow$
i $<$ *lnth* (*kfilter P n* (*ldropn n nell*)) *0* $\Longrightarrow$
x $\in$ *lset* (*ldropn n nell*) $\Longrightarrow$ *P x* $\Longrightarrow$ *enat i* $\leq$ *epred* (*llength nell*) $\Longrightarrow$ *P* (*lnth nell i*) $\Longrightarrow$ *False*
by (*metis co.enat.exhaust-sel iless-Suc-eq kfilter-n-not-before llength-eq-0 not-gr-zero*)
next
fix *na* :: *nat* **and** *nella* :: *'b llist* **and** *Pa* :: *'b* $\Rightarrow$ *bool* **and** *ia* :: *nat* **and** *x* :: *'b*
assume *a1*: *ia* $<$ *lnth* (*kfilter Pa na* (*ldropn na nella*)) *0*
assume *a2*: ¬ *lnull* (*kfilter Pa na* (*ldropn na nella*))
assume *a3*: *enat na* $\leq$ *epred* (*llength nella*)
assume *a4*: ¬ *lnull nella*
assume *a5*: ¬ *enat ia* $\leq$ *epred* (*llength nella*)
assume *a6*: *Pa* (*lnth nella* (*the-enat* (*epred* (*llength nella*))))
have *f7*: $\forall$ *p n bs*. *lnull* (*kfilter p n* (*bs*::*'b llist*)) $\vee$ *lnth* (*kfilter p n bs*) *0* = *n* + *lleast p bs*
by (*meson kfilter-lnth-zero*)
then have *f8*: *lnth* (*kfilter Pa na* (*ldropn na nella*)) *0* = *na* + *lleast Pa* (*ldropn na nella*)
using *a2* **by** *blast*
have *f9*: *0* $<$ *llength* (*kfilter Pa na* (*ldropn na nella*))
using *a2* **by** *simp*
have *f10*: *na* $\leq$ *the-enat* (*epred* (*llength nella*))
by (*metis a3 a5 enat-ord-code*(*4*) *enat-ord-simps*(*1*) *enat-the-enat less-imp-le*)
have *f11*: *the-enat* (*epred* (*llength nella*)) $<$ *lnth* (*kfilter Pa na* (*ldropn na nella*)) *0*
by (*metis a1 a5 dual-order.strict-trans enat-ord-simps*(*2*) *enat-ord-simps*(*3*) *enat-the-enat not-le-imp-less*)
then show *False* **using** *kfilter-n-not-before*[*of Pa na nella* (*the-enat* (*epred* (*llength nella*)))]

using *f10 f11*
by (*metis a3 a4 a6 co.enat.exhaust-sel f9 iless-Suc-eq llength-eq-0*)
qed

lemma *nkfilter-not-after*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
nnth (*nkfilter P 0 nell*) *k* < *i*
nlength(*nkfilter P 0 nell*) = (*enat k*)
i $\leq$ *nlength nell*
shows $\neg$ *P* (*nnth nell i*)
using *assms*
by *transfer*
(*auto simp add*: *min-def split*: *if-split-asm*,
metis co.enat.exhaust-sel diff-Suc-1 eSuc-enat iless-Suc-eq kfilter-not-after llength-eq-0 not-gr-zero)

lemma *nkfilter-n-not-after*:
assumes ($\exists$ *x* $\in$ *nset* (*ndropn n nell*). *P x*)
n $\leq$ *nlength nell*
nnth (*nkfilter P n* (*ndropn n nell*)) *k* < *i*
nlength(*nkfilter P n* (*ndropn n nell*)) = (*enat k*)
i $\leq$ *nlength nell*
shows $\neg$ *P* (*nnth nell i*)
using *assms*
by *transfer*
(*auto split*: *if-split-asm*,
metis diff-Suc-1 eSuc-enat eSuc-epred iless-Suc-eq kfilter-n-not-after llength-eq-0 not-gr-zero)

lemma *nkfilter-not-between*:
assumes ($\exists$ *x* $\in$ *nset nell*. *P x*)
nnth (*nkfilter P 0 nell*) *k* < *i*
i < *nnth* (*nkfilter P 0 nell*) (*Suc k*)
(*Suc k*) $\leq$ *nlength* (*nkfilter P 0 nell*)
shows $\neg$ *P* (*nnth nell i*)
using *assms*
by *transfer*
(*auto simp add*: *min-def split*: *if-split-asm*,
metis co.enat.exhaust-sel iless-Suc-eq kfilter-not-between llength-eq-0,
metis co.enat.exhaust-sel gen-llength-def iless-Suc-eq kfilter-upperbound le-less-trans llength-code llength-eq-0 min.cobounded2 min.strict-order-iff min-enat-simps(*1*),
meson Suc-ile-eq less-imp-le,
meson Suc-ile-eq dual-order.strict-implies-order)

lemma *ntaken-nset*:
assumes *k*$\leq$ *nlength nell*
shows *nset* (*ntaken k nell*) = { (*nnth nell i*) | *i*. *i* $\leq$ *k*}
using *assms*
by (*auto simp add*: *in-nset-conv-nnth*)
(*metis min.orderE ntaken-nnth*,

metis min.orderE ntaken-nnth)

lemma *ndropn-nset*:
assumes $k \leq$ *nlength nell*
shows *nset* (*ndropn k nell*) = { (*nnth nell i*) | *i*. $k \leq i \wedge i \leq$ *nlength nell*}
using *assms*
by (*auto simp add*: *in-nset-conv-nnth*)
(*metis add.commute enat-min-eq le-add1 plus-enat-simps*(*1*),
metis enat-minus-mono1 idiff-enat-enat le-add-diff-inverse)

lemma *nfilter-nkfilter-ntaken-nidx-a*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*ntaken k* (*nkfilter P n nell*))
using *assms*
by (*simp add*: *nidx-expand nidx-nkfilter-gr ntaken-nnth*)

lemma *nfilter-nkfilter-ndropn-nidx-a*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*ndropn k* (*nkfilter P n nell*))
using *assms*
by *transfer*
(*auto simp add*: *kfilter-lnull-conv*,
metis (*no-types*, *lifting*) *gr-implies-not-zero kfilter-llength leI lfilter-kfilter-ldropn-lidx-a*
lidx-def llength-eq-0 lnull-ldropn)

lemma *nfilter-nkfilter-ntaken-nidx-b*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms nidx-nkfilter*[*of* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *P 0*]
by (*metis nfinite-ntaken nset-nlast ntaken-nlast*)

lemma *nfilter-nkfilter-ntaken-nidx-b-1*:
assumes $\exists$ $x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms nfilter-nkfilter-ntaken-nidx-b*[*of P nell k*] *nkfilter-holds*[*of nell P*]
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *nfilter-nkfilter-ntaken-nidx-b-2*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P n nell*) *k*) $-n$))
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P n* (*ntaken* (*nnth* (*nkfilter P n nell*) *k*) *nell*))
using *assms nidx-nkfilter*[*of* (*ntaken* (*nnth* (*nkfilter P n nell*) *k*) *nell*) *P n*]
by (*metis* (*mono-tags*, *lifting*) *diff-le-self linorder-le-cases mem-Collect-eq*
nfilter-nkfilter-ntaken-nlength-0 ntaken-all ntaken-nset)

lemma *nfilter-nkfilter-ntaken-nidx-b-3*:
assumes $\exists\ x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P n* (*ntaken* (*nnth* (*nkfilter P n nell*) *k*) *nell*))
using *assms nfilter-nkfilter-ntaken-nidx-b-2*[*of P nell n k*] *nkfilter-holds*
by (*metis in-nset-conv-nnth nkfilter-nlength*)

lemma *nfilter-nkfilter-ndropn-nidx-b*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms nidx-nkfilter*[*of* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *P 0*]
by (*metis diff-add diff-zero in-nset-conv-nnth ndropn-nnth zero-enat-def zero-le*)

lemma *nfilter-nkfilter-ndropn-nidx-b-1*:
assumes $\exists\ x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms nfilter-nkfilter-ndropn-nidx-b*[*of P nell k*] *nkfilter-holds*[*of nell P*]
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *nfilter-nkfilter-ndropn-nidx-b-2*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P n nell*) *k*) $-n$))
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P* ((*nnth* (*nkfilter P n nell*) *k*)$-n$)
(*ndropn* ((*nnth* (*nkfilter P n nell*) *k*)$-n$) *nell*))
proof −
have *1*: *enat* (*nnth* (*nkfilter P n nell*) *k*) $-n \leq$ *nlength nell*
using *nkfilter-upperbound*[*of nell P*] *nkfilter-nnth-n-zero*[*of nell P k n*] *assms*
by (*metis gen-nlength-def idiff-enat-enat nfilter-nkfilter-ntaken-nlength-0*
nkfilter-nlength nlength-code)
have *2*: *nset* (*ndropn* ((*nnth* (*nkfilter P n nell*) *k*)$-n$) *nell*) =
{ (*nnth nell i*) |*i*. ((*nnth* (*nkfilter P n nell*) *k*)$-n$) $\leq i \wedge i \leq$ *nlength nell* }
using *ndropn-nset*[*of* (*nnth* (*nkfilter P n nell*) *k*)$-n$ *nell*] *1* **by** *simp*
have *3*: $\exists$*x*$\in$*nset* (*ndropn* ((*nnth* (*nkfilter P n nell*) *k*)$-n$) *nell*). *P x*
using *1 2 assms*(*1*) **by** *auto*
show *?thesis* **using** *3*
nidx-nkfilter[*of* (*ndropn* ((*nnth* (*nkfilter P n nell*) *k*)$-n$) *nell*) *P* (*nnth* (*nkfilter P n nell*) *k*)$-n$]
by *blast*
qed

lemma *nfilter-nkfilter-ndropn-nidx-b-3*:
assumes $\exists\ x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nidx* (*nkfilter P* ((*nnth* (*nkfilter P n nell*) *k*)$-n$)
(*ndropn* ((*nnth* (*nkfilter P n nell*) *k*)$-n$) *nell*))
using *assms nfilter-nkfilter-ndropn-nidx-b-2*
by (*metis in-nset-conv-nnth nkfilter-holds nkfilter-nlength*)

lemma *ntaken-nkfilter-ntaken-nset-eq*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
k ≤ *nlength* (*nfilter P nell*)
shows *nset*(*ntaken k* (*nkfilter P 0 nell*)) =
nset(*nkfilter P 0* (*ntaken* ((*nnth* (*nkfilter P 0 nell*) *k*)) *nell*))
proof −
have *1*: ((*nnth* (*nkfilter P 0 nell*) *k*)) ≤ *nlength nell*
using *assms nkfilter-upperbound*[*of nell P k 0*]
by (*metis diff-zero gen-nlength-def nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength nlength-code*)
have *2*: ∃ *x* ∈ *nset nell*. *P x*
using *assms* **by** (*metis 1 exists-Pred-nnth-nset*)
have *3*: {*i*. *i* ≤ *nlength*(*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) ∧
P (*nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *i*) }
= {*i*. *i* ≤ (*nnth* (*nkfilter P 0 nell*) *k*) ∧ *P* (*nnth nell i*)}
by (*auto simp add*: *ntaken-nnth 1 order-subst2*)
have *4*: ∃ *x* ∈ *nset*(*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*). *P x*
using *1 assms*(*1*) *ntaken-nset* **by** *fastforce*
have *5*: {*i*. *i* ≤ (*nnth* (*nkfilter P 0 nell*) *k*) ∧ *P* (*nnth nell i*)} =
{*i*. *i* ≤ (*nnth* (*nkfilter P 0 nell*) *k*) ∧ *i* ∈ *nset*(*nkfilter P 0 nell*)}
using *4 1 2 nkfilter-holds-b*
by (*metis* (*mono-tags*, *opaque-lifting*) *add-cancel-left-right enat-ord-simps*(*1*) *order.trans*)
have *6*: {*i*. *i* ≤ (*nnth* (*nkfilter P 0 nell*) *k*) ∧ *i* ∈ *nset*(*nkfilter P 0 nell*)} =
{*i*. *i*≤ (*nnth* (*nkfilter P 0 nell*) *k*) ∧
i∈ { (*nnth* (*nkfilter P 0 nell*) *j*) | *j*. *j* ≤ *nlength* (*nkfilter P 0 nell*)}}
by (*simp add*: *nset-conv-nnth*)
have *7*: {*i*. *i*≤ (*nnth* (*nkfilter P 0 nell*) *k*) ∧
i∈ { (*nnth* (*nkfilter P 0 nell*) *j*) | *j*. *j* ≤ *nlength* (*nkfilter P 0 nell*)}} =
{(*nnth* (*nkfilter P 0 nell*) *j*) |*j*. *j* ≤ *k*}
by (*auto simp add*: *assms 2 nidx-nkfilter-less-eq nkfilter-nlength*)
(*metis 2 dual-order.antisym le-cases nidx-nkfilter-gr-eq nkfilter-nlength*,
metis assms(*2*) *dual-order.trans enat-ord-simps*(*1*))
have *8*: *k* ≤ *nlength* (*nkfilter P 0 nell*)
by (*simp add*: *2 assms*(*2*) *nkfilter-nlength*)
have *9*: {(*nnth* (*nkfilter P 0 nell*) *j*) |*j*. *j* ≤ *k*} = *nset*(*ntaken k* (*nkfilter P 0 nell*))
using *ntaken-nset*[*of k* (*nkfilter P 0 nell*)] *8* **by** *auto*
have *91*: *min* (*enat* (*nnth* (*nkfilter P 0 nell*) *k*)) (*nlength nell*) = (*nnth* (*nkfilter P 0 nell*) *k*)
by (*simp add*: *1 min-def*)
have *10*: *nset*(*nkfilter P 0* (*ntaken* ((*nnth* (*nkfilter P 0 nell*) *k*)) *nell*)) =
{*i*. *i* ≤ (*nnth* (*nkfilter P 0 nell*) *k*) ∧
P (*nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *i*) }
using *nkfilter-nset*[*of* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *P 0*]
1 4 91 ntaken-nlength[*of* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*]
by *auto*
have *11*: *nlength*((*ntaken* ((*nnth* (*nkfilter P 0 nell*) *k*)) *nell*)) = (*nnth* (*nkfilter P 0 nell*) *k*)
by (*simp add*: *1*)
have *12*: {*i*. *i* ≤ (*nnth* (*nkfilter P 0 nell*) *k*) ∧
P (*nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *i*) } =
{*i*. *i* ≤ *nlength*((*ntaken* ((*nnth* (*nkfilter P 0 nell*) *k*)) *nell*)) ∧
P (*nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *i*) }

using *11* **by** *auto*
show *?thesis*
using *10 12 3 5 6 7 9* **by** *auto*
qed

lemma *ntaken-nkfilter-ntaken-nset-eq-1*:
assumes $\exists\ x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nfilter P nell*)
shows *nset*(*ntaken k* (*nkfilter P 0 nell*)) =
nset(*nkfilter P 0* (*ntaken* ((*nnth* (*nkfilter P 0 nell*) *k*)) *nell*))
using *assms ntaken-nkfilter-ntaken-nset-eq*[*of P nell k*]
nkfilter-holds[*of nell P* (*nnth* (*nkfilter P 0 nell*) *k*) *0*]
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *ndropn-nkfilter-ndropn-nset-eq*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
$k \leq$ *nlength* (*nfilter P nell*)
shows *nset*(*ndropn k* (*nkfilter P 0 nell*)) =
nset(*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*) (*ndropn* ((*nnth* (*nkfilter P 0 nell*) *k*)) *nell*))
proof −
have *1*: (*nnth* (*nkfilter P 0 nell*) *k*) $\leq$ *nlength nell*
using *nkfilter-upperbound*[*of nell P k 0*] *assms*
by (*metis diff-zero gen-nlength-def nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength nlength-code*)
have *2*: $\exists\ x \in$ *nset nell*. *P x*
using *assms* **by** (*metis 1 exists-Pred-nnth-nset*)
have *4*: $\exists\ x \in$ *nset*(*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*). *P x*
using *1 assms*(*1*) *ndropn-nset* **by** *fastforce*
have *10*: *nset*(*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*)
(*ndropn* ((*nnth* (*nkfilter P 0 nell*) *k*)) *nell*)) =
{(*nnth* (*nkfilter P 0 nell*) *k*) + *i*| *i*. *i* $\leq$ *nlength nell* − (*nnth* (*nkfilter P 0 nell*) *k*) $\wedge$
P (*nnth nell* (*i*+ (*nnth* (*nkfilter P 0 nell*) *k*))) }
using *1*
nkfilter-ndropn-nset-b[*of* (*nnth* (*nkfilter P 0 nell*) *k*) *nell P* (*nnth* (*nkfilter P 0 nell*) *k*)]
4 **by** *linarith*
have *5*: {(*nnth* (*nkfilter P 0 nell*) *k*) + *i*| *i*. *i* $\leq$ *nlength nell* − (*nnth* (*nkfilter P 0 nell*) *k*) $\wedge$
P (*nnth nell* (*i*+ (*nnth* (*nkfilter P 0 nell*) *k*))) } =
{(*nnth* (*nkfilter P 0 nell*) *k*) + *i*| *i*. *i* $\leq$ *nlength nell* − (*nnth* (*nkfilter P 0 nell*) *k*) $\wedge$
(*i*+ (*nnth* (*nkfilter P 0 nell*) *k*)) $\in$ *nset*(*nkfilter P 0 nell*)}
proof (*auto simp add*: *add.commute*)
fix *i* :: *nat*
assume *a1*: *enat i* $\leq$ *nlength nell* − *enat* (*nnth* (*nkfilter P 0 nell*) *k*)
assume *a2*: *P* (*nnth nell* (*i* + *nnth* (*nkfilter P 0 nell*) *k*))
have *f0*: *enat* (*i* + (*nnth* (*nkfilter P 0 nell*) *k*)) $\leq$ *nlength nell*
using *1 a1 enat-min-eq* **by** *auto*
show *i* + *nnth* (*nkfilter P 0 nell*) *k* $\in$ *nset* (*nkfilter P 0 nell*)
by (*metis Nat.add-0-right a2 f0 in-nset-conv-nnth nkfilter-holds-b*)
next
fix *i*
assume *b0*: *enat i* $\leq$ *nlength nell* − *enat* (*nnth* (*nkfilter P 0 nell*) *k*)
assume *b1*: *i* + *nnth* (*nkfilter P 0 nell*) *k* $\in$ *nset* (*nkfilter P 0 nell*)

show P (*nnth nell* (i + *nnth* (*nkfilter P 0 nell*) k))
using *nkfilter-holds*[*of nell P*] *2* **by** (*metis b1 minus-nat.diff-0*)
qed
have *51*: $\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i \mid i.\ i \leq nlength\ nell - (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$(i + (nnth\ (nkfilter\ P\ 0\ nell)\ k)) \in nset(nkfilter\ P\ 0\ nell)\} =$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i \mid i.$
$(nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq i + (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$i + (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq nlength\ nell\ \wedge$
$(i + (nnth\ (nkfilter\ P\ 0\ nell)\ k)) \in nset(nkfilter\ P\ 0\ nell)\}$
by *auto*
(*metis 2 add.left-neutral in-nset-conv-nnth nkfilter-upperbound zero-enat-def*,
metis add-diff-cancel-right′ enat-minus-mono1 idiff-enat-enat)
have *52*: $\{(nnth\ (nkfilter\ P\ 0\ nell)\ k) + i \mid i.$
$(nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq i + (nnth\ (nkfilter\ P\ 0\ nell)\ k) \wedge$
$i + (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq nlength\ nell\ \wedge$
$(i + (nnth\ (nkfilter\ P\ 0\ nell)\ k)) \in nset(nkfilter\ P\ 0\ nell)\} =$
$\{j.\ (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq j \wedge j \leq nlength\ nell \wedge j \in nset(nkfilter\ P\ 0\ nell)\}$
by (*metis* (*no-types*, *lifting*) *add.commute diff-add*)
have *53* : $\{j.\ (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq j \wedge j \leq nlength\ nell \wedge j \in nset(nkfilter\ P\ 0\ nell)\} =$
$\{j.\ (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq j \wedge j \leq nlength\ nell \wedge j \in$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ jj) \mid jj.\ jj \leq nlength\ (nkfilter\ P\ 0\ nell)\}\}$
by (*simp add*: *nset-conv-nnth*)
have *54*: $\{j.\ (nnth\ (nkfilter\ P\ 0\ nell)\ k) \leq j \wedge j \leq nlength\ nell \wedge j \in$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ jj) \mid jj.\ jj \leq nlength\ (nkfilter\ P\ 0\ nell)\}\} =$
$\{(nnth\ (nkfilter\ P\ 0\ nell)\ j) \mid j.\ k \leq j \wedge j \leq nlength(nkfilter\ P\ 0\ nell)\}$
using *assms 2*
by (*auto*,
metis dual-order.antisym le-cases nidx-less-eq nidx-nkfilter nkfilter-nlength,
meson nidx-nkfilter-gr-eq,
metis gen-nlength-def nkfilter-upperbound nlength-code)
have *8*: $k \leq nlength\ (nkfilter\ P\ 0\ nell)$
by (*simp add*: *2 assms*(*2*) *nkfilter-nlength*)
have *9*: $\{(nnth\ (nkfilter\ P\ 0\ nell)\ j) \mid j.\ k \leq j \wedge j \leq nlength(nkfilter\ P\ 0\ nell)\} =$
nset(*ndropn k* (*nkfilter P 0 nell*))
by (*simp add*: *8 ndropn-nset*)
show *?thesis*
using *10 5 51 52 53 54 9* **by** *auto*
qed

lemma *ndropn-nkfilter-ndropn-nset-eq-1*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows *nset*(*ndropn k* (*nkfilter P 0 nell*)) =
nset(*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*) (*ndropn* ((*nnth* (*nkfilter P 0 nell*) *k*)) *nell*))
using *assms ndropn-nkfilter-ndropn-nset-eq*[*of P nell k*]
by (*metis diff-zero in-nset-conv-nnth nkfilter-holds nkfilter-nlength*)

lemma *nkfilter-nkfilter-ntaken*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$

shows *ntaken k* (*nkfilter P 0 nell*) =
(*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms*
by (*simp add*: *nfilter-nkfilter-ntaken-nidx-a nfilter-nkfilter-ntaken-nidx-b-1 nidx-nset-eq ntaken-nkfilter-ntaken-nset-eq-1*)

lemma *nkfilter-nkfilter-ndropn*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$k \leq nlength\ (nfilter\ P\ nell)$
shows *ndropn k* (*nkfilter P 0 nell*) =
(*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*) (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
proof −
have *1*: *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
using *nkfilter-holds*[*of nell P*] *assms*
by (*metis diff-zero in-nset-conv-nnth nkfilter-nlength*)
show *?thesis*
by (*metis 1 assms*(*1*) *assms*(*2*) *diff-zero ndropn-nkfilter-ndropn-nset-eq nfilter-nkfilter-ndropn-nidx-a nfilter-nkfilter-ndropn-nidx-b-3 nidx-nset-eq*)
qed

lemma *nkfilter-nmap-nfilter*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
shows *nmap* ($\lambda n.$ *nnth nell n*) (*nkfilter P 0 nell*) = *nfilter P nell*
using *assms nellist-eq-nnth-eq*[*of nmap* ($\lambda n.$ *nnth nell n*) (*nkfilter P 0 nell*) *nfilter P nell*]
nkfilter-nfilter[*of nell P - 0*] *nkfilter-nnth-n-zero*[*of nell P - 0*]
by (*simp add*: *nkfilter-nlength*)

lemma *nfilter-nkfilter-ntaken*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
$k \leq nlength\ (nkfilter\ P\ 0\ nell)$
shows *ntaken k* (*nfilter P nell*) =
nfilter P (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)
proof −
have *1*: $\exists\ x \in nset\ nell.\ P\ x$
using *assms*
by (*metis in-nset-conv-nnth min.cobounded1 min-def nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have *2*: *nfilter P nell* = *nmap* ($\lambda n.$ *nnth nell n*) (*nkfilter P 0 nell*)
using *1 assms* **by** (*simp add*: *nkfilter-nmap-nfilter*)
have *3*: *ntaken k* (*nfilter P nell*) = *ntaken k* (*nmap* ($\lambda n.$ *nnth nell n*) (*nkfilter P 0 nell*))
using *2* **by** *simp*
have *4*: *ntaken k* (*nmap* ($\lambda n.$ *nnth nell n*) (*nkfilter P 0 nell*)) =
nmap ($\lambda n.$ *nnth nell n*) (*ntaken k* (*nkfilter P 0 nell*))
by *simp*
have *5*: $\exists\ x \in nset$ (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*). *P x*
using *assms* **by** (*metis nfinite-ntaken nset-nlast ntaken-nlast*)
have *6*: (*nfilter P* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) =
nmap ($\lambda s.$ *nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *s*)

(*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *5* by (*simp add*: *nkfilter-nmap-nfilter*)
have *7*: *nmap* (λ*n. nnth nell n*) (*ntaken k* (*nkfilter P 0 nell*)) =
nmap (λ*n. nnth nell n*) (*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
by (*simp add*: *1 2 assms*(*2*) *nkfilter-nkfilter-ntaken*)
have *70*: *enat k* $\leq$ *nlength* (*nfilter P nell*)
using *2 assms* by *auto*
have *71*: ($\bigwedge$*z. z* $\in$ *nset* (*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) $\Longrightarrow$
(λ*n. nnth nell n*) *z* = (λ*s. nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *s*) *z*)
using *assms 1 70 ntaken-nkfilter-ntaken-nset-eq*[*of P nell k*]
ntaken-nset[*of k* (*nkfilter P 0 nell*)] *mem-Collect-eq*
nidx-nkfilter-gr-eq[*of nell P - - 0*]
proof *simp*
fix *z* :: *nat*
assume *a1*: *enat k* $\leq$ *nlength* (*nkfilter P 0 nell*)
assume *a2*: $\bigwedge$*k j.* [[*k* $\leq$ *j*; *enat j* $\leq$ *nlength* (*nkfilter P 0 nell*)]] $\Longrightarrow$
nnth (*nkfilter P 0 nell*) *k* $\leq$ *nnth* (*nkfilter P 0 nell*) *j*
assume *a3*: *z* $\in$ *nset* (*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
assume {*nnth* (*nkfilter P 0 nell*) *i* |*i. i* $\leq$ *k*} =
nset (*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
then have $\exists$ *n. z* = *nnth* (*nkfilter P 0 nell*) *n* $\wedge$ *n* $\leq$ *k*
using *a3* by *blast*
then have *z* $\leq$ *nnth* (*nkfilter P 0 nell*) *k*
using *a2 a1* by *meson*
then show *nnth nell z* = *nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *z*
by (*simp add*: *ntaken-nnth*)
qed
have *8*: *nmap* (λ*n. nnth nell n*)
(*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) =
nmap (λ*s. nnth* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *s*)
(*nkfilter P 0* (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *1 5 assms nellist.map-cong0*
using *71* by *blast*
show *?thesis*
by (*simp add*: *3 6 7 8*)
qed

lemma *nfilter-nkfilter-ntaken-1*:
assumes $\exists$ *x* $\in$ *nset nell. P x*
k $\leq$ *nlength* (*nkfilter P 0 nell*)
shows *ntaken k* (*nfilter P nell*) =
nfilter P (*ntaken* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)
using *assms nfilter-nkfilter-ntaken*[*of P nell k*]
by (*metis in-nset-conv-nnth nkfilter-holds nkfilter-nnth-n-zero*)

lemma *nkfilter-nmap-shift*:
assumes $\exists$ *x* $\in$ *nset nell. P x*
shows *nmap* (λ*s. nnth nell* (*s*+*n*)) (*nkfilter P 0 nell*) =
nmap (λ*s. nnth nell s*) (*nkfilter P n nell*)
proof –

have *1*: *nlength* (*nmap* (*λs. nnth nell* (*s*+*n*)) (*nkfilter P 0 nell*)) =
nlength (*nmap* (*λs. nnth nell s*) (*nkfilter P n nell*))
by (*simp add*: *assms nkfilter-nlength*)
have *2*: ⋀*i. i*≤ *nlength* (*nmap* (*λs. nnth nell* (*s*+*n*)) (*nkfilter P 0 nell*)) ⟶
nnth (*nmap* (*λs. nnth nell* (*s*+*n*)) (*nkfilter P 0 nell*)) *i* =
nnth nell ((*nnth* (*nkfilter P 0 nell*) *i*)+*n*)
by *simp*
have *3*: ⋀*i. i*≤ *nlength* (*nmap* (*λs. nnth nell s*) (*nkfilter P n nell*)) ⟶
nnth (*nmap* (*λs. nnth nell s*) (*nkfilter P n nell*)) *i* =
nnth nell (*nnth* (*nkfilter P n nell*) *i*)
by *simp*
have *4*: ⋀*i. i*≤ *nlength* (*nmap* (*λs. nnth nell* (*s*+*n*)) (*nkfilter P 0 nell*)) ⟶
nnth nell ((*nnth* (*nkfilter P 0 nell*) *i*)+*n*) = *nnth nell* (*nnth* (*nkfilter P n nell*) *i*)
using *nkfilter-n-zero*[*of nell P n*]
by (*simp add*: *assms*)
show *?thesis* **using** *1 4 nellist-eq-nnth-eq* **by** *force*
qed

lemma *nkfilter-nmap-shift-ndropn*:
assumes ∃ *x* ∈ *nset* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*). *P x*
k ≤ *nlength* (*nkfilter P 0 nell*)
shows *nmap* (*λs. nnth nell* (*s*+(*nnth* (*nkfilter P 0 nell*) *k*)))
(*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) =
nmap (*λs. nnth nell s*) (*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*)
(*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *assms*
nellist-eq-nnth-eq[*of nmap* (*λs. nnth nell* (*s*+(*nnth* (*nkfilter P 0 nell*) *k*)))
(*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
nmap (*λs. nnth nell s*) (*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*)
(*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))]
using *nkfilter-n-zero*[*of* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *P*
(*nnth* (*nkfilter P 0 nell*) *k*)]
by *simp*

lemma *nfilter-nkfilter-ndropn*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *k*)))
k ≤ *nlength* (*nkfilter P 0 nell*)
shows *ndropn k* (*nfilter P nell*) =
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)
proof −
have *1*: ∃ *x* ∈ *nset nell*. *P x*
using *assms*
by (*metis in-nset-conv-nnth min.cobounded1 min-def nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have *2*: (*nfilter P nell*) = *nmap* (*λn. nnth nell n*) (*nkfilter P 0 nell*)
by (*simp add*: *1 nkfilter-nmap-nfilter*)
have *3*: *ndropn k* (*nfilter P nell*) = *ndropn k* (*nmap* (*λn. nnth nell n*) (*nkfilter P 0 nell*))
by (*simp add*: *2*)
have *4*: *ndropn k* (*nmap* (*λn. nnth nell n*) (*nkfilter P 0 nell*)) =
nmap (*λn. nnth nell n*) (*ndropn k* (*nkfilter P 0 nell*))
using *ndropn-nmap* **by** *blast*

have *5*: $\exists\ x \in$ *nset*(*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*). *P x*
by (*metis add.commute add.left-neutral assms*(*1*) *in-nset-conv-nnth ndropn-nnth zero-enat-def zero-le*)
have *6*: *nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) =
nmap (λ*s. nnth* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *s*)
(*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
by (*metis 5 nkfilter-nmap-nfilter*)
have *7*: *nmap* (λ*s. nnth* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*) *s*)
(*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) =
nmap (λ*s. nnth nell* (*s*+(*nnth* (*nkfilter P 0 nell*) *k*)))
(*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
by (*simp add*: *add.commute*)
have *8*: *nmap* (λ*s. nnth nell* (*s*+(*nnth* (*nkfilter P 0 nell*) *k*)))
(*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) =
nmap (λ*s. nnth nell s*)
(*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*) (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*))
using *5 assms*(*2*) *nkfilter-nmap-shift-ndropn* **by** *fastforce*
have *9*: *nmap* (λ*s. nnth nell s*)
(*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *k*) (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)) =
nmap (λ*s. nnth nell s*)
(*ndropn k* (*nkfilter P 0 nell*))
by (*simp add*: *1 2 assms*(*2*) *nkfilter-nkfilter-ndropn*)
show *?thesis* **using** *3 4 6 7 8 9* **by** *auto*
qed

lemma *nfilter-nkfilter-ndropn-1*:
assumes $\exists x \in$ *nset nell*. *P x*
$k \leq$ *nlength* (*nkfilter P 0 nell*)
shows *ndropn k* (*nfilter P nell*) =
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) *k*) *nell*)
using *assms nfilter-nkfilter-ndropn*
by (*metis in-nset-conv-nnth minus-nat.diff-0 nkfilter-holds*)

lemma *nfilter-nkfilter-nsubn*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *i*)))
enat i $\leq$ *nlength* (*nkfilter P 0 nell*)
P (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *j*)))
enat j $\leq$ *nlength* (*nkfilter P 0 nell*)
$i \leq j$
shows *nsubn* (*nfilter P nell*) *i j* =
(*nfilter P* (*nsubn nell*
(*nnth* (*nkfilter P 0 nell*) *i*)
(*nnth* (*nkfilter P 0 nell*) *j*)
)
)

proof –
have *0*: *nsubn* (*nfilter P nell*) *i j* = *ntaken* ($j - i$) (*ndropn i* (*nfilter P nell*))

using *nsubn-def1* **by** *blast*
have *1*: *ntaken* (*j* − *i*) (*ndropn i* (*nfilter P nell*)) =
ntaken (*j* − *i*) (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*))
by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nkfilter-ndropn*)
have *4*: ((*nnth* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*)) +
(*nnth* (*nkfilter P 0 nell*) *i*)) =
((*nnth* (*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *i*) (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*)))
proof −
have *f1*: $\bigwedge$*n ns*. (*nfirst* (*ndropn n ns*)::*nat*) = *nlast* (*ntaken n ns*)
by (*metis* (*lifting*) *ndropn-0 ndropn-nfirst ndropn-nnth ntaken-ndropn-nlast*)
have $\bigwedge$*n f ns*. *nlast* (*ntaken n* (*nmap f ns*)) = (*f* (*nlast* (*ntaken n ns*)::*nat*)::*nat*)
by *simp*
then show *?thesis*
using *f1* **by** (*metis* *assms*(*1*) *in-nset-conv-nnth ndropn-0 ndropn-nfirst nkfilter-n-zero zero-enat-def zero-le*)
qed
have *5*: ((*nnth* (*nkfilter P* (*nnth* (*nkfilter P 0 nell*) *i*) (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*))) =
(*nnth* (*nkfilter P 0 nell*) *j*)
using *assms nkfilter-nkfilter-ndropn*[*of nell P i*]
by (*metis in-nset-conv-nnth le-add-diff-inverse linorder-le-cases linorder-not-less ndropn-nnth nfinite-ntaken nkfilter-nlength nnth-beyond nset-nlast ntaken-all*)
have *10*: ∃ *x*∈*nset* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*). *P x*
by (*metis assms*(*1*) *in-nset-conv-nnth le-zero-eq nle-le nnth-zero-ndropn zero-enat-def*)
have *11*: *ndropn i* (*nfilter P nell*) = *nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nkfilter-ndropn*)
have *12*: *nlength* (*ndropn i* (*nfilter P nell*)) = *nlength* (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*))
by (*simp add*: *11*)
have *13*: *nlength* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) =
nlength (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*))
by (*simp add*: *10 nkfilter-nlength*)
have *14*: *enat i* ≤ *nlength* (*nfilter P nell*)
by (*metis assms*(*1*) *assms*(*2*) *in-nset-conv-nnth linorder-le-cases nfinite-ntaken nkfilter-nlength nset-nlast*

ntaken-all ntaken-nlast)
have *15*: *enat j* ≤ *nlength* (*nfilter P nell*)
by (*metis 14 assms*(*1*) *assms*(*4*) *diff-zero nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength*)
have *16*: *enat* (*j* − *i*) ≤ *nlength* (*ndropn i* (*nfilter P nell*))
by (*metis 15 enat-minus-mono1 idiff-enat-enat ndropn-nlength*)
have *2*: *ntaken* (*j* − *i*) (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) =
(*nfilter P* (*ntaken* (*nnth* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*))
(*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)))
by (*metis 10 12 13 16 nfilter-nkfilter-ntaken-1*)
have *3*: (*nfilter P* (*ntaken* (*nnth* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*))
(*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*))) =
(*nfilter P* (*nsubn nell*
(*nnth* (*nkfilter P 0 nell*) *i*)
((*nnth* (*nkfilter P 0* (*ndropn* (*nnth* (*nkfilter P 0 nell*) *i*) *nell*)) (*j* − *i*)) +
(*nnth* (*nkfilter P 0 nell*) *i*))
))

by (*simp add*: *ntaken-ndropn*)
show *?thesis* **using** *0 1 2 3 4 5* **by** *auto*
qed

lemma *nfilter-nkfilter-nsubn-zero*:
assumes *P* (*nnth nell* ((*nnth* (*nkfilter P 0 nell*) *j*)))
enat j ≤ *nlength* (*nkfilter P 0 nell*)
shows *nsubn* (*nfilter P nell*) *0 j* =
(*nfilter P* (*nsubn nell*
0
(*nnth* (*nkfilter P 0 nell*) *j*)
)
)

by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nkfilter-ntaken nsubn-def1*)

lemma *nkfilter-nnth-aa*:
assumes ∃ *x* ∈ *nset nell*. *P x*
n ≤ *nlength* (*nfilter P nell*)
shows *P* (*nnth* (*nfilter P nell*) *n*)
using *assms in-nset-conv-nnth nkfilter-holds*[*of nell P - 0*] *nkfilter-nfilter*[*of nell P - 0*]
by (*metis nkfilter-nlength*)

lemma *nfilter-nlength-zero-conv-a*:
assumes ∃ *x* ∈ *nset nell*. *P x*
nlength(*nfilter P nell*) = *0*
shows (∃ *k* ≤ *nlength nell*. *P* (*nnth nell k*) ∧
(∀ *j*≤ *nlength nell*. *j* ≠*k* ⟶ ¬ *P* (*nnth nell j*)))
using *assms*
apply *transfer*
proof (*auto simp add*: *epred-llength min-def split*: *if-split-asm*)
fix *nella* :: *'a llist* **and** *Pa* :: *'a* ⇒ *bool* **and** *x* :: *'a*
assume *a1*: ¬ *lnull nella*
assume *a2*: *Pa x*
assume *a3*: *x* ∈ *lset nella*
assume *a4*: *lnull* (*ltl* (*lfilter Pa nella*))
show ∃ *k*. (*enat k* ≤ *llength* (*ltl nella*) ⟶
Pa (*lnth nella k*) ∧ (∀ *j*. *enat j* ≤ *llength* (*ltl nella*) ⟶ *j* ≠ *k* ⟶ ¬ *Pa* (*lnth nella j*))) ∧
enat k ≤ *llength* (*ltl nella*)
proof −
have *1*: *LCons* (*lhd nella*) (*ltl nella*) = *nella*
by (*metis a2 a3 lfilter-LNil llist.disc*(*1*) *llist.exhaust-sel lnull-lfilter*)
have *2*: *llength nella* = *eSuc* (*llength* (*ltl nella*))
by (*metis 1 llength-LCons*)
have *3*: ¬ *lnull* (*lfilter Pa nella*)
by (*meson a2 a3 lnull-lfilter*)
show *?thesis*
by (*metis 2 3 a4 iless-Suc-eq lfilter-llength-one-conv-a lhd-LCons-ltl llength-LCons llength-LNil*

llist.collapse(*1*) *one-eSuc*)
qed
qed

lemma *nfilter-nlength-zero-conv-c*:
$(\exists\ k \leq \mathit{nlength\ nell}.\ P\ (\mathit{nnth\ nell\ k}) \wedge$
$(\forall j \leq \mathit{nlength\ nell}.\ j \neq k \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j}))) =$
$(\exists\ k \leq \mathit{nlength\ nell}.\ P\ (\mathit{nnth\ nell\ k}) \wedge$
$(\forall j \leq \mathit{nlength\ nell}.\ j<k \vee k<j \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j})))$
using *antisym-conv3* **by** *auto*

lemma *nfilter-nlength-zero-conv-d*:
$(\exists\ k \leq \mathit{nlength\ nell}.\ P\ (\mathit{nnth\ nell\ k}) \wedge$
$(\forall j \leq \mathit{nlength\ nell}.\ j<k \vee k<j \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j}))) \longleftrightarrow$
$(\exists\ k \leq \mathit{nlength\ nell}.\ P\ (\mathit{nnth\ nell\ k}) \wedge$
$(\forall j.\ j<k \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j})) \wedge$
$(\forall j \leq \mathit{nlength\ nell}.\ k<j \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j})))$
by (*meson enat-ord-simps*(*2*) *le-cases le-less-trans*)

lemma *nfilter-nlength-zero-conv-b*:
assumes $(\exists\ k \leq \mathit{nlength\ nell}.\ P\ (\mathit{nnth\ nell\ k}) \wedge$
$(\forall j \leq \mathit{nlength\ nell}.\ j \neq k \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j})))$
shows $(\exists x \in \mathit{nset\ nell}.\ P\ x) \wedge \mathit{nlength}(\mathit{nfilter}\ P\ \mathit{nell}) = \mathit{0}$
using *assms*
by *transfer*
(*auto split*: *if-split-asm*,
metis co.enat.exhaust-sel iless-Suc-eq in-lset-conv-lnth llength-eq-0,
metis co.enat.exhaust-sel co.enat.sel(*2*) *iless-Suc-eq lfilter-llength-one-conv-b llength-eq-0*
one-eSuc)

lemma *nfilter-nlength-zero-conv*:
$((\exists x \in \mathit{nset\ nell}.\ P\ x) \wedge \mathit{nlength}(\mathit{nfilter}\ P\ \mathit{nell}) = \mathit{0}) \longleftrightarrow$
$(\exists\ k \leq \mathit{nlength\ nell}.\ P\ (\mathit{nnth\ nell\ k}) \wedge$
$(\forall j \leq \mathit{nlength\ nell}.\ j \neq k \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j})))$
using *nfilter-nlength-zero-conv-a*[*of nell P*] *nfilter-nlength-zero-conv-b*[*of nell P*]
by *blast*

lemma *nfilter-nlength-zero-conv-1*:
$((\exists x \in \mathit{nset\ nell}.\ P\ x) \wedge \mathit{nlength}(\mathit{nfilter}\ P\ \mathit{nell}) = \mathit{0}) \longleftrightarrow$
$(\exists\ k \leq \mathit{nlength\ nell}.\ P\ (\mathit{nnth\ nell\ k}) \wedge$
$(\forall j \leq \mathit{nlength\ nell}.\ j<k \vee k<j \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j})))$
using *nfilter-nlength-zero-conv*[*of nell P*] *nfilter-nlength-zero-conv-c*[*of nell P*]
by *blast*

lemma *nfilter-nlength-zero-conv-2*:
$((\exists x \in \mathit{nset\ nell}.\ P\ x) \wedge \mathit{nlength}(\mathit{nfilter}\ P\ \mathit{nell}) = \mathit{0}) \longleftrightarrow$
$(\exists\ k \leq \mathit{nlength\ nell}.\ P\ (\mathit{nnth\ nell\ k}) \wedge$
$(\forall j.\ j<k \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j})) \wedge$
$(\forall j \leq \mathit{nlength\ nell}.\ k<j \longrightarrow \neg\ P\ (\mathit{nnth\ nell\ j})))$
using *nfilter-nlength-zero-conv-1*[*of nell P*] *nfilter-nlength-zero-conv-d*[*of nell P*]

by *blast*

lemma *nfilter-ndropns-nmap-help-0*:
assumes $\exists\ x \in nset\ nell.\ P\ x$
$j \leq nnth(nkfilter\ P\ 0\ nell)\ 0$
shows $(nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ 0)\ nell)) = (nfilter\ P\ (ndropn\ j\ nell))$
using *assms*
proof (*induction j arbitrary*: *nell*)
case *0*
then show *?case*
by (*metis ndropn-0 nfilter-nkfilter-ndropn-1 zero-enat-def zero-le*)
next
case (*Suc j*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis*
by *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: $is\text{-}NNil\ x22 \Longrightarrow$
$nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ 0)\ nell) = nfilter\ P\ (ndropn\ (Suc\ j)\ nell)$
by (*metis NCons Suc.prems*(*2*) *Suc-le-D ndropn-Suc-NCons ndropn-is-NNil*)
have *2*: $P\ x21 \wedge \neg is\text{-}NNil\ x22 \Longrightarrow$
$nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ 0)\ nell) = nfilter\ P\ (ndropn\ (Suc\ j)\ nell)$
using *Suc NCons*
by *simp*
(*metis Suc.prems*(*1*) *dual-order.strict-trans1 nkfilter-not-before nnth-0 zero-less-Suc*)
have *3*: $\neg P\ x21 \wedge \neg is\text{-}NNil\ x22 \Longrightarrow$
$nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ 0)\ nell) = nfilter\ P\ (ndropn\ (Suc\ j)\ nell)$
using *Suc NCons* **by** (*simp add*: *nkfilter-nleast*)
show *?thesis*
using *1 2 3* **by** *blast*
qed
qed
qed

lemma *nfilter-nappend-ntaken*:
assumes $\exists\ x \in nset\ (ntaken\ k\ nell).\ P\ x$
$k \leq nlength\ nell$
shows $nfilter\ P\ (ntaken\ k\ nell) =$
$ntaken\ (the\text{-}enat\ (nlength(nfilter\ P\ (ntaken\ k\ nell))))\ (nfilter\ P\ nell)$
using *assms*
apply *transfer*
proof *auto*
show $\bigwedge k\ nell\ P\ x.$
$\neg\ lnull\ nell \Longrightarrow$
$enat\ k \leq epred\ (llength\ nell) \Longrightarrow$

$x \in$ *lset* (*ltake* (*enat* (*Suc* *k*)) *nell*) $\Longrightarrow$
P *x* $\Longrightarrow$
$\forall$ *x*$\in$*lset* *nell*. $\neg$ *P* *x* $\Longrightarrow$
lfilter *P* (*ltake* (*enat* (*Suc* *k*)) *nell*) =
ltake (*enat* (*Suc* (*the-enat* (*epred* (*llength* (*lfilter* *P* (*ltake* (*enat* (*Suc* *k*)) *nell*))))))) *nell*
by (*metis dual-order.strict-trans1 in-lset-conv-lnth llength-ltake lprefix-lnthD ltake-is-lprefix min.cobounded2*)
next
fix *k*
fix *nell* :: *'a llist*
fix *P*
fix *x*
fix *xa*
assume *a0*: $\neg$ *lnull nell*
assume *a1*: *enat* *k* $\leq$ *epred* (*llength nell*)
assume *a2*: $x \in$ *lset* (*ltake* (*enat* (*Suc* *k*)) *nell*)
assume *a3*: *P* *x*
assume *a4*: *xa* $\in$ *lset nell*
assume *a5*: *P* *xa*
show *lfilter* *P* (*ltake* (*enat* (*Suc* *k*)) *nell*) =
ltake (*enat* (*Suc* (*the-enat* (*epred* (*llength* (*lfilter* *P* (*ltake* (*enat* (*Suc* *k*)) *nell*))))))) (*lfilter* *P* *nell*)
proof (*cases* *k* = *epred* (*llength nell*))
case *True*
then show *?thesis*
proof −
have *f1*: *eSuc* (*enat* *k*) = *llength nell*
by (*simp add*: *True a0*)
then have *llength* (*lfilter* *P* *nell*) $\neq$ *0*
by (*metis* (*no-types*) *a2 a3 eSuc-enat llength-eq-0 lnull-lfilter ltake-all order-refl*)
then show *?thesis*
using *f1* **by** (*metis True co.enat.exhaust-sel eSuc-enat enat-the-enat epred-le-epredI infinity-ileE llength-lfilter-ile ltake-all order-refl*)
qed
next
case *False*
then show *?thesis*
proof −
have *f1*: *llength nell* $\neq$ *0*
using *a0 llength-eq-0* **by** *blast*
have *g1*: $\neg$ *lnull* (*lfilter* *P* (*ltake* (*enat* (*Suc* *k*)) *nell*))
by (*meson a2 a3 lnull-lfilter*)
then show *?thesis*
using *f1*
proof −
have *f1*: *enat* *k* < *epred* (*llength nell*)
using *False a1 order.not-eq-order-implies-strict* **by** *blast*
have *f2*: $\forall$ *e*. *e* = *0* $\vee$ *e* = *eSuc* (*epred* *e*)
by (*metis co.enat.exhaust-sel*)
then have *g2*: *enat* (*Suc* *k*) < *llength nell*
using *f1* **by** (*metis* (*no-types*) *Suc-ile-eq* ‹*llength nell* $\neq$ *0*› *iless-Suc-eq*)

then show *?thesis*
using *f2*
using *lfilter-lappend-ltake*[*of Suc k*]
proof −
have *eSuc* (*enat k*) = *min* (*enat* (*Suc k*)) (*llength nell*)
by (*metis* ‹*enat* (*Suc k*) < *llength nell*› *eSuc-enat min.strict-order-iff*)
then have *eSuc* (*enat k*) = *llength* (*ltake* (*enat* (*Suc k*)) *nell*)
by *simp*
then show *?thesis*
by (*metis g1 g2 co.enat.exhaust-sel eSuc-enat eSuc-infinity enat-the-enat infinity-ileE lfilter-lappend-ltake llength-eq-0 llength-lfilter-ile*)
qed
qed
qed
qed
qed

lemma *nappend-nfilter-nfinite*:
assumes ∃ *x* ∈ *nset* (*nellx*). *P x*
∃ *x* ∈ *nset* (*nelly*). *P x*
nfinite nellx
shows *nfilter P* (*nappend nellx nelly*) = *nappend* (*nfilter P nellx*) (*nfilter P nelly*)
using *assms*
by *transfer auto*

lemma *nfilter-nappend-ndropn*:
assumes ∃ *x* ∈ *nset* (*ndropn* (*Suc k*) *nell*). *P x*
∃ *x* ∈ *nset* (*ntaken k nell*). *P x*
(*Suc k*) ≤ *nlength nell*
shows *nfilter P* (*ndropn* (*Suc k*) *nell*) =
ndropn (*the-enat* (*eSuc*(*nlength*(*nfilter P* (*ntaken k nell*))))) (*nfilter P nell*)
proof −
have *1*: *nfinite* (*nfilter P* (*ntaken k nell*))
by (*metis Suc-ile-eq assms*(*3*) *dual-order.strict-implies-order enat-ile length-nfilter-le min.absorb1 nfinite-conv-nlength-enat ntaken-nlength*)
have *2*: *nfilter P nell* = *nappend* (*nfilter P* (*ntaken k nell*)) (*nfilter P* (*ndropn* (*Suc k*) *nell*))
using *assms nappend-nfilter-nfinite*[*of* (*ntaken k nell*) *P* (*ndropn* (*Suc k*) *nell*)]
nappend-ntaken-ndropn[*of k nell*] *nfinite-ntaken*[*of k nell*] **by** *simp*
have *3*: *ndropn* (*the-enat* (*eSuc*(*nlength*(*nfilter P* (*ntaken k nell*)))))
(*nappend* (*nfilter P* (*ntaken k nell*)) (*nfilter P* (*ndropn* (*Suc k*) *nell*)))
= (*nfilter P* (*ndropn* (*Suc k*) *nell*))
by (*metis 1 antisym-conv2 gr-zeroI ile-eSuc iless-Suc-eq less-imp-le ndropn-0 ndropn-nappend3 nfinite-conv-nlength-enat one-enat-def plus-1-eSuc*(*2*) *plus-enat-simps*(*1*) *the-enat.simps zero-less-diff*)
show *?thesis*
by (*simp add*: *2 3*)
qed

lemma *nkfilter-nappend-ntaken*:
assumes $\exists\ x \in nset\ (ntaken\ k\ nell).\ P\ x$
$k\leq\ nlength\ nell$
shows $nkfilter\ P\ n\ (ntaken\ k\ nell) =$
$ntaken\ (the\text{-}enat\ (nlength(nkfilter\ P\ n\ (ntaken\ k\ nell))))\ (nkfilter\ P\ n\ nell)$
using *assms*
apply *transfer*
proof (*auto simp add*: *kfilter-not-lnull-conv kfilter-lnull-conv*)
show $\bigwedge k\ nell\ P\ n\ x.$
$\neg\ lnull\ nell \Longrightarrow$
$enat\ k \leq\ epred\ (llength\ nell) \Longrightarrow$
$x \in\ lset\ (ltake\ (enat\ (Suc\ k))\ nell) \Longrightarrow$
$P\ x \Longrightarrow$
$\forall\, x{\in}lset\ nell.\ \neg\ P\ x \Longrightarrow$
$kfilter\ P\ n\ (ltake\ (enat\ (Suc\ k))\ nell) =$
$ltake\ (enat\ (Suc\ (the\text{-}enat\ (epred\ (llength\ (kfilter\ P\ n\ (ltake\ (enat\ (Suc\ k))\ nell)))))))\ (iterates\ Suc\ n)$
by (*meson lset-ltake subsetD*)
next
fix $ka :: nat$ **and** $nella :: 'a\ llist$ **and** $Pa :: 'a \Rightarrow bool$ **and** $na :: nat$ **and** $x :: 'a$ **and** $xa :: 'a$
assume *a1*: $Pa\ x$
assume *a2*: $Pa\ xa$
assume *a3*: $xa \in\ lset\ nella$
assume *a4*: $x \in\ lset\ (ltake\ (enat\ (Suc\ ka))\ nella)$
have *f5*: $\bigwedge e.\ e = enat\ 0 \vee eSuc\ (epred\ e) = e$
by (*metis* (*no-types*) *co.enat.exhaust-sel zero-enat-def*)
have *f6*: $\bigwedge as.\ min\ \infty\ (llength\ (as{::}'a\ llist)) = llength\ as$
by *simp*
have *f7*: $eSuc\ \infty = \infty$
by *simp*
have *f8*: $(enat\ (Suc\ (the\text{-}enat\ (epred\ (llength\ (kfilter\ Pa\ na\ (ltake\ (enat\ (Suc\ ka))\ nella))))))) =$
$(llength\ (kfilter\ Pa\ na\ (ltake\ (enat\ (Suc\ ka))\ nella)))$
proof −
have *f1*: $\forall\, n.\ \neg\ \infty \leq\ enat\ n$
by (*meson infinity-ileE*)
have $\neg\ lnull\ (kfilter\ Pa\ na\ (ltake\ (enat\ (Suc\ ka))\ nella))$
by (*metis* (*no-types*) *a1 a4 kfilter-not-lnull-conv*)
then show *?thesis*
using *f1*
by (*metis co.enat.exhaust-sel dual-order.trans eSuc-enat eSuc-ile-mono enat-the-enat kfilter-llength llength-eq-0 llength-lfilter-ile llength-ltake min.cobounded1*)
qed
moreover
{ assume $llength\ nella \neq \infty$
then have $ltake\ (enat\ (Suc\ ka))\ nella = nella \longrightarrow$
$ltake\ (llength\ (lfilter\ Pa\ (ltake\ (enat\ (Suc\ ka))\ nella)))\ (kfilter\ Pa\ na\ nella) =$
$kfilter\ Pa\ na\ (ltake\ (enat\ (Suc\ ka))\ nella) \wedge$
$epred\ (llength\ (lfilter\ Pa\ (ltake\ (enat\ (Suc\ ka))\ nella))) \neq \infty$
using *f7 f6 f5 a3 a2* **by** (*metis* (*no-types*) *kfilter-llength llength-eq-0 llength-lfilter-ile lnull-lfilter ltake-all min.orderE not-le-imp-less not-less-iff-gr-or-eq zero-enat-def*)
moreover

{ **assume** *ltake* (*enat* (*Suc ka*)) *nella* ≠ *nella*
then have *enat* (*Suc ka*) < *llength nella* ∧ *min* (*enat* (*Suc ka*)) (*llength nella*) ≠ ∞ ∨
enat (*Suc ka*) < *llength nella* ∧ *llength* (*lfilter Pa* (*ltake* (*enat* (*Suc ka*)) *nella*)) ≠ *eSuc* ∞
by (*metis* (*no-types*) *enat-ord-code*(*4*) *ltake-all min.orderE not-le-imp-less not-less-iff-gr-or-eq*) }
ultimately have *enat* (*Suc ka*) < *llength nella* ∧ *min* (*enat* (*Suc ka*)) (*llength nella*) ≠ ∞ ∨
enat (*Suc ka*) < *llength nella* ∧ *llength* (*lfilter Pa* (*ltake* (*enat* (*Suc ka*)) *nella*)) ≠ *eSuc* ∞ ∨
ltake (*llength* (*lfilter Pa* (*ltake* (*enat* (*Suc ka*)) *nella*))) (*kfilter Pa na nella*) =
kfilter Pa na (*ltake* (*enat* (*Suc ka*)) *nella*) ∧
epred (*llength* (*lfilter Pa* (*ltake* (*enat* (*Suc ka*)) *nella*))) ≠ ∞
by *fastforce* }
ultimately show *kfilter Pa na* (*ltake* (*enat* (*Suc ka*)) *nella*) =
ltake (*enat* (*Suc* (*the-enat* (*epred* (*llength* (*kfilter Pa na* (*ltake* (*enat* (*Suc ka*)) *nella*))))))
(*kfilter Pa na nella*)
using *f6 f5 a4 a1*
kfilter-lappend-ltake[*of* (*enat* (*Suc ka*)) *nella Pa na*]
by (*metis enat-ord-code*(*4*) *kfilter-llength*)
qed

lemma *nfilter-ndropns-nmap-help-1*:
assumes ∃ *x* ∈ *nset nell*. *P x*
j ≤ *nnth*(*nkfilter P 0 nell*) (*Suc 0*)
nnth (*nkfilter P 0 nell*) *0* < *j*
(*Suc 0*) ≤ *nlength*(*nfilter P nell*)
shows (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc 0*)) *nell*)) =
(*nfilter P* (*ndropn j nell*))
using *assms*
proof
(*induction j arbitrary*: *nell*)
case *0*
then show *?case*
by *blast*
next
case (*Suc j*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis*
by *auto*
next
case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: *is-NNil x22* ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc 0*)) *nell*) =
nfilter P (*ndropn* (*Suc j*) *nell*)
by (*metis NCons Suc.prems*(*2*) *Suc-le-D ndropn-Suc-NCons ndropn-is-NNil*)
have *2*: *P x21* ∧ ¬ *is-NNil x22* ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc 0*)) *nell*) =
nfilter P (*ndropn* (*Suc j*) *nell*)
using *Suc NCons*

proof *simp*
assume *a1*: *P x21* ∧ ¬ *is-NNil x22*
assume *a2*: *enat* (*Suc 0*) ≤ *nlength* (*nfilter P* (*NCons x21 x22*))
assume *a3*: *nell* = *NCons x21 x22*
assume *a4*: *Suc j* ≤ *nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc 0*)
have *f5*: ∀ *as p a n*. ((∃ *a*. (*a*::'*a*) ∈ *nset as* ∧ *p a*) ∨ ¬ *p a*) ∨
nkfilter p n (*NCons a as*) = *NNil n*
by (*metis* (*no-types*) *nkfilter-NCons-a*)
have *f6*: ∀ *as p n*. (∀ *a*. (*a*::'*a*) ∉ *nset as* ∨ ¬ *p a*) ∨
nlength (*nkfilter p n as*) = *nlength* (*nfilter p as*)
by (*meson nkfilter-nlength*)
have *f7*: ∀ *p as*. (∀ *a*. (*a*::'*a*) ∉ *nset as* ∨ ¬ *p a*) = (∀ *a*. *a* ∉ *nset as* ∨ ¬ *p a*)
by *meson*
have *f8*: *nlength* (*nkfilter P 0* (*NCons x21 x22*)) =
nlength (*nfilter P* (*NCons x21 x22*))
by (*meson a1 nellist.set-intros*(*2*) *nkfilter-nlength*)
have *f9*: ∀ *p as*. (∀ *a*. (*a*::'*a*) ∉ *nset as* ∨ ¬ *p a*) = (∀ *a*. *a* ∉ *nset as* ∨ ¬ *p a*)
by *meson*
have *f10*: (∃ *a*. *a* ∈ *nset x22* ∧ *P a*) ⟶
nkfilter P 0 (*NCons x21 x22*) = *NCons 0* (*nkfilter P* (*Suc 0*) *x22*)
using *a1* **by** (*simp add*: *Bex-def-raw*)
then have *f11*: (∃ *a*. *a* ∈ *nset x22* ∧ *P a*) ⟶
nnth (*nkfilter P* (*Suc 0*) *x22*) *0* − *Suc 0* = *nnth* (*nkfilter P 0 x22*) *0*
using *f9 f8 f7 a2* **by** (*metis Suc-ile-eq iless-Suc-eq nkfilter-nnth-n-zero nlength-NCons*)
have *f14*: *j* < *nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc 0*)
using *a4 Suc-le-eq* **by** *blast*
have ∃ *a*. *a* ∈ *nset x22* ∧ *P a*
using *f8 f5 a2 a1* **by** (*metis Suc-ile-eq gr-implies-not-zero nlength-NNil*)
then have (∃ *a*. *a* ∈ *nset x22* ∧ *P a*) ∧ *nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 x22*) *0*) *x22*) =
nfilter P (*ndropn j x22*)
using *f14 f11 a4*
proof −
have *j* ≤ *nnth* (*nkfilter P 0 x22*) *0*
using ‹∃ *a*. *a* ∈ *nset x22* ∧ *P a*› *f10 f11 f14* **by** *fastforce*
then show *?thesis*
by (*simp add*: *Bex-def-raw* ‹∃ *a*. *a* ∈ *nset x22* ∧ *P a*› *nfilter-ndropns-nmap-help-0*)
qed
then show *nfilter P* (*ndropn* (*nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc 0*)) (*NCons x21 x22*)) =
nfilter P (*ndropn j x22*)
using *f11 a4*
by (*metis One-nat-def Suc-le-D diff-Suc-1 f10 ndropn-Suc-NCons nnth-Suc-NCons*)
qed
have *3*: ¬*P x21* ∧ ¬ *is-NNil x22* ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc 0*)) *nell*) =
nfilter P (*ndropn* (*Suc j*) *nell*)
using *Suc NCons*
proof *simp*
assume *a1*: ⋀*nell*. ⟦∃ *a*∈*nset nell*. *P a*; *j* ≤ *nnth* (*nkfilter P 0 nell*) (*Suc 0*);
nnth (*nkfilter P 0 nell*) *0* < *j*; *enat* (*Suc 0*) ≤ *nlength* (*nfilter P nell*)⟧ ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc 0*)) *nell*) =

nfilter P (*ndropn j nell*)
assume *a2*: *enat* (*Suc 0*) ≤ *nlength* (*nfilter P x22*)
assume *a3*: *Suc j* ≤ *nnth* (*nkfilter P* (*Suc 0*) *x22*) (*Suc 0*)
assume *a4*: ∃ *x*∈*nset x22*. *P x*
assume *a5*: *nnth* (*nkfilter P* (*Suc 0*) *x22*) *0* < *Suc j*
have *f12*: *nnth* (*nkfilter P 0 x22*) (*Suc 0*) =
((*nnth* (*nkfilter P* (*Suc 0*) *x22*) (*Suc 0*)) − (*Suc 0*))
using *nkfilter-nnth-n-zero*[*of x22 P Suc 0 Suc 0*]
by (*simp add*: *a2 a4 nkfilter-nlength*)
then have *f13*: *j* ≤ *nnth* (*nkfilter P 0 x22*) (*Suc 0*)
using *a3* **by** *linarith*
have *f14*: *enat 0* ≤ *nlength* (*nfilter P x22*)
using *a2* **by** (*metis One-nat-def one-enat-def order.trans zero-enat-def zero-le-one*)
have *f15*: *nlength* (*nkfilter P* (*Suc 0*) *x22*) = *nlength* (*nfilter P x22*)
by (*simp add*: *a4 nkfilter-nlength*)
then have *Suc 0* ≤ *nnth* (*nkfilter P* (*Suc 0*) *x22*) *0*
by (*simp add*: *a4 f14 nkfilter-lowerbound*)
then have *Suc* (*nnth* (*nkfilter P 0 x22*) *0*) ≤ *j*
by (*metis a4 a5 add-Suc leD nkfilter-nleast not-less-eq-eq*)
then have *nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 x22*) (*Suc 0*)) *x22*) =
nfilter P (*ndropn j x22*)
by (*simp add*: *Suc.IH Suc-le-lessD a2 a4 f13*)
then show *nfilter P* (*ndropn* (*nnth* (*nkfilter P* (*Suc 0*) *x22*) (*Suc 0*)) (*NCons x21 x22*)) =
nfilter P (*ndropn j x22*)
using *ndropn-Suc-NCons*
by (*metis One-nat-def a2 a4 f12 f15 le-add-diff-inverse nkfilter-lowerbound plus-1-eq-Suc*)
qed
show *?thesis*
using *1 2 3* **by** *blast*
qed
qed
qed

lemma *nfilter-ndropns-nmap-help-j*:
assumes ∃ *x* ∈ *nset nell*. *P x*
j ≤ *nnth*(*nkfilter P 0 nell*) (*Suc i*)
nnth (*nkfilter P 0 nell*) *i* < *j*
(*Suc i*) ≤ *nlength*(*nfilter P nell*)
shows (*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*)) =
(*nfilter P* (*ndropn j nell*))
using *assms*
proof (*induction j arbitrary*: *nell i*)
case *0*
then show *?case*
by *blast*
next
case (*Suc j*)
then show *?case*
proof (*cases nell*)
case (*NNil x1*)

then show *?thesis*
by *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
  proof −
   have *1*: *is-NNil x22* $\Longrightarrow$
      nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) =
      nfilter P (*ndropn* (*Suc j*) *nell*)
    by (*metis NCons Suc.prems*(*2*) *Suc-le-D ndropn-Suc-NCons ndropn-is-NNil*)
   have *2*: $\neg$ *is-NNil x22* $\wedge$ *i* = *0* $\Longrightarrow$
      nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) =
      nfilter P (*ndropn* (*Suc j*) *nell*)
    using *Suc nfilter-ndropns-nmap-help-1*[*of nell P*] **by** *metis*
   have *3*: *P x21* $\wedge$ $\neg$ *is-NNil x22* $\wedge$ *0*< *i* $\Longrightarrow$
      nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) = *nfilter P* (*ndropn* (*Suc j*) *nell*)
    using *Suc NCons*
    proof *simp*
    assume *a1*: $\bigwedge$*nell i.* $[\![\exists$ *a*$\in$*nset nell. P a*; *j* $\leq$ *nnth* (*nkfilter P 0 nell*) (*Suc i*);
          nnth (*nkfilter P 0 nell*) *i* < *j*; *enat* (*Suc i*) $\leq$ *nlength* (*nfilter P nell*)$]\!]$ $\Longrightarrow$
        nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) =
        nfilter P (*ndropn j nell*)
    assume *a2*: *P x21* $\wedge$ $\neg$ *is-NNil x22* $\wedge$ *0* < *i*
    assume *a3*: *enat* (*Suc i*) $\leq$ *nlength* (*nfilter P* (*NCons x21 x22*))
    assume *a4*: *nell* = *NCons x21 x22*
    assume *a5*: *Suc j* $\leq$ *nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*)
    assume *a6*: *nnth* (*nkfilter P 0* (*NCons x21 x22*)) *i* < *Suc j*
    show *nfilter P* (*ndropn* (*nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*)) (*NCons x21 x22*)) =
       nfilter P (*ndropn j x22*)
    proof −
     have *f0*: $\exists$ *a*$\in$*nset x22. P a*
      by (*metis a2 a3 dual-order.antisym enat.inject nat.distinct*(*1*) *nfilter-NCons-a*
        nlength-NNil zero-enat-def zero-le)
     have *f1*: *nlength* (*nkfilter P 0* (*NCons x21 x22*)) = *nlength* (*nfilter P* (*NCons x21 x22*))
      using *a4* **by** (*metis* (*no-types*) *Suc*(*2*) *nkfilter-nlength*)
     have *f2*: *nkfilter P 0* (*NCons x21 x22*) = *NCons 0* (*nkfilter P* (*Suc 0*) *x22*)
      by (*metis a2 a3 dual-order.antisym enat.inject f1 nat.distinct*(*1*) *nkfilter-NCons*
        nkfilter-NCons-a nlength-NNil zero-enat-def zero-le)
     have *f3*: *nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*) =
        nnth ((*nkfilter P* (*Suc 0*) *x22*)) (*i*)
      by (*simp add*: *f2*)
     have *f4*: *enat i* $\leq$ *nlength* (*nkfilter P* (*Suc 0*) *x22*)
      by (*metis Suc-ile-eq a3 f1 f2 iless-Suc-eq nlength-NCons*)
     have *f5*: *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*i*) = (*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*i*)
      using *nkfilter-nnth-n-zero*[*of x22 P i Suc 0*]
      using *a5 f0 f3 f4* **by** *linarith*
     have *f6*: *ndropn* (*nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*)) (*NCons x21 x22*) =
        ndropn ((*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*i*)) (*NCons x21 x22*)
      using *f3 f5* **by** *presburger*
     have *f7*: *ndropn* ((*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*i*)) (*NCons x21 x22*) =

ndropn (*nnth* ((*nkfilter P 0 x22*)) (*i*)) *x22*
using *ndropn-Suc-NCons* **by** *auto*
have *f8*: $j \leq$ *nnth* (*nkfilter P 0 x22*) (*i*)
using *a5 f3 f5* **by** *linarith*
have *f11*: *nnth* (*nkfilter P 0* (*NCons x21 x22*)) *i* = *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*i*−*1*)
by (*metis One-nat-def Suc-pred a2 f2 nnth-Suc-NCons*)
have *f12*: *enat* (*i* − *1*) ≤ *nlength* (*nkfilter P* (*Suc 0*) *x22*)
by (*metis One-nat-def Suc-ile-eq Suc-pred a2 f4 less-imp-le*)
have *f13*: (*Suc 0*) ≤ *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*i*−*1*)
by (*meson f0 f12 nkfilter-lowerbound*)
have *f14*: *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*i*−*1*) = (*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*i*−*1*)
using *nkfilter-nnth-n-zero*[*of x22 P i*−*1 Suc 0*] *f12 f0 f13* **by** (*metis le-add-diff-inverse*)
have *f9*: *nnth* (*nkfilter P 0 x22*) (*i*−*1*) < *j*
using *a6 f11 f14* **by** *linarith*
have *f10*: *i* ≤ *nlength* (*nfilter P x22*)
by (*metis f0 f4 nkfilter-nlength*)
show *?thesis* **using** *a1*[*of x22 i*−*1*]
by (*metis Suc-diff-1 a2 f0 f10 f3 f5 f7 f8 f9*)
qed
qed
have *4*: ¬*P x21* ∧ ¬ *is-NNil x22* ∧ *0*< *i* ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) =
nfilter P (*ndropn* (*Suc j*) *nell*)
using *Suc NCons*
proof *simp*
assume *a0*: ¬ *P x21* ∧ ¬ *is-NNil x22* ∧ *0* < *i*
assume *a2*: ⋀*nell i*. ⟦∃ *a*∈*nset nell*. *P a*; *j* ≤ *nnth* (*nkfilter P 0 nell*) (*Suc i*);
nnth (*nkfilter P 0 nell*) *i* < *j*; *enat* (*Suc i*) ≤ *nlength* (*nfilter P nell*)⟧ ⟹
nfilter P (*ndropn* (*nnth* (*nkfilter P 0 nell*) (*Suc i*)) *nell*) =
nfilter P (*ndropn j nell*)
assume *a1*: ∃ *x*∈*nset x22*. *P x*
assume *a4*: *Suc j* ≤ *nnth* (*nkfilter P* (*Suc 0*) *x22*) (*Suc i*)
assume *a5*: *nnth* (*nkfilter P* (*Suc 0*) *x22*) *i* < *Suc j*
assume *a3*: *enat* (*Suc i*) ≤ *nlength* (*nfilter P x22*)
assume *a6*: *nell* = *NCons x21 x22*
show *nfilter P* (*ndropn* (*nnth* (*nkfilter P* (*Suc 0*) *x22*) (*Suc i*)) (*NCons x21 x22*)) =
nfilter P (*ndropn j x22*)
proof −
have *f1*: *nlength* (*nkfilter P 0* (*NCons x21 x22*)) = *nlength* (*nfilter P* (*NCons x21 x22*))
by (*metis NCons Suc.prems*(*1*) *nkfilter-nlength*)
have *f2*: *nkfilter P 0* (*NCons x21 x22*) = (*nkfilter P* (*Suc 0*) *x22*)
using *NCons Suc.prems*(*2*) *a1 a5* **by** *auto*
have *f3*: *nnth* (*nkfilter P 0* (*NCons x21 x22*)) (*Suc i*) =
nnth (*nkfilter P* (*Suc 0*) *x22*) (*Suc i*)
by (*simp add*: *f2*)
have *f4*: *enat* (*Suc i*) ≤ *nlength* (*nkfilter P* (*Suc 0*) *x22*)
using *NCons Suc.prems*(*4*) *f1 f2* **by** *auto*
have *f5*: *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*Suc i*) = (*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*Suc i*)
by (*metis One-nat-def Suc-le-D a1 a4 diff-Suc-1 f4 nkfilter-nnth-n-zero plus-1-eq-Suc*)
have *f6*: (*ndropn* (*nnth* (*nkfilter P* (*Suc 0*) *x22*) (*Suc i*)) (*NCons x21 x22*)) =

ndropn ((*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*Suc i*)) (*NCons x21 x22*)
by (*simp add*: *f5*)
have *f7*: *ndropn* ((*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*Suc i*)) (*NCons x21 x22*) =
ndropn (*nnth* ((*nkfilter P 0 x22*)) (*Suc i*)) (*x22*)
by *simp*
have *f8*: *j* ≤ *nnth* (*nkfilter P 0 x22*) (*Suc i*)
using *a4 f5* **by** *force*
have *f11*: *nnth* (*nkfilter P 0* (*NCons x21 x22*)) *i* = *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*i*)
by (*simp add*: *f2*)
have *f12*: *enat* (*i*) ≤ *nlength* (*nkfilter P* (*Suc 0*) *x22*)
using *Suc-ile-eq f4* **by** *auto*
have *f13*: (*Suc 0*) ≤ *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*i*)
by (*simp add*: *a1 f12 nkfilter-lowerbound*)
have *f14*: *nnth* ((*nkfilter P* (*Suc 0*) *x22*)) (*i*) = (*Suc 0*) +*nnth* ((*nkfilter P 0 x22*)) (*i*)
by (*metis a1 f12 f13 le-add-diff-inverse nkfilter-nnth-n-zero*)
have *f9*: *nnth* (*nkfilter P 0 x22*) (*i*) < *j*
using *a5 f14* **by** *auto*
show *?thesis*
using *Suc.IH a1 a3 f6 f7 f8 f9* **by** *presburger*
qed
qed
show *?thesis*
using *1 2 3 4* **by** *fastforce*
qed
qed
qed

lemma *nfilter-ndropns-nmap*:
assumes ∃ *x* ∈ *nset* (*ndropns nell*). *P x*
(*Suc i*) ≤ *nlength*(*nfilter P* (*ndropns nell*))
shows *ndropn* (*Suc i*) (*nmap* (λ*s*. *nnth s 0*) (*nfilter P* (*ndropns nell*))) =
(*nmap* (λ*s*. *nnth s 0*)
(*nfilter P* (*ndropns* (*ndropn* (*Suc* (*nnth* (*nkfilter P 0* (*ndropns nell*)) *i*)) *nell*))))
proof −
have *1*: *ndropn* (*Suc i*) (*nmap* (λ*s*. *nnth s 0*) (*nfilter P* (*ndropns nell*))) =
nmap (λ*s*. *nnth s 0*) (*ndropn* (*Suc i*) (*nfilter P* (*ndropns nell*)))
by (*simp add*: *ndropn-nmap*)
have *2*: (*Suc* (*nnth* (*nkfilter P 0* (*ndropns nell*)) *i*)) ≤ *nlength* (*ndropns nell*)
using *assms nkfilter-nkfilter-ntaken*[*of* (*ndropns nell*) *P i*]
by (*metis Suc-ile-eq antisym-conv2 less-imp-le min.orderE nkfilter-nlength not-le-imp-less ntaken-all ntaken-nlength*)
have *3*: (*nfilter P* (*ndropns* (*ndropn* (*Suc* (*nnth* (*nkfilter P 0* (*ndropns nell*)) *i*)) *nell*))) =
(*nfilter P* (*ndropn* (*Suc* (*nnth* (*nkfilter P 0* (*ndropns nell*)) *i*)) (*ndropns nell*)))
by (*simp add*: *2 ndropn-ndropns*)
have *4*: (*ndropn* (*Suc i*) (*nfilter P* (*ndropns nell*))) =
(*nfilter P* (*ndropn* (*nnth* (*nkfilter P 0* (*ndropns nell*)) (*Suc i*)) (*ndropns nell*)))
by (*simp add*: *assms*(*1*) *assms*(*2*) *nfilter-nkfilter-ndropn-1 nkfilter-nlength*)
have *5*: (*Suc* (*nnth* (*nkfilter P 0* (*ndropns nell*)) *i*)) ≤ (*nnth* (*nkfilter P 0* (*ndropns nell*)) (*Suc i*))
by (*simp add*: *Suc-leI assms*(*1*) *assms*(*2*) *nidx-nkfilter-expand nkfilter-nlength*)
have *6*: *i* < *nlength*(*nfilter P* (*ndropns nell*))

using *Suc-ile-eq assms(2)* **by** *blast*
have *7*: *nnth (nkfilter P 0 (ndropns nell)) i* < *(Suc (nnth (nkfilter P 0 (ndropns nell)) i))*
by *simp*
have *8*: *(nfilter P (ndropn (nnth (nkfilter P 0 (ndropns nell)) (Suc i)) (ndropns nell)))* =
(nfilter P (ndropn (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) (ndropns nell)))
using *5 6 7 assms*
nfilter-ndropns-nmap-help-j[of ndropns nell P (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) i]
by *blast*
show *?thesis*
using *1 3 4 8* **by** *auto*
qed

lemma *ndropns-nfilter-nnth-exist-ndropn*:
assumes $\exists$ *x* $\in$ *nset (ndropns nell). P x*
j $\leq$ *nlength (nfilter P (ndropns nell))*
shows ($\exists$ *i. enat i* $\leq$ *nlength nell* $\wedge$ *nnth (nfilter P (ndropns nell)) j* = *ndropn i nell*)
proof –
have *1*: *nnth (nfilter P (ndropns nell)) j* $\in$ *nset (ndropns nell)*
using *assms in-nset-conv-nnth nfilter-nnth* **by** *fastforce*
show *?thesis* **using** *1***using** *in-nset-ndropns* **by** *blast*
qed

lemma *nfilter-ndropns-nnth-bound*:
assumes ($\exists$ *ys* $\in$ *nset (ndropns xs). P ys*)
j $\leq$ *nlength (nfilter P (ndropns xs))*
shows *nlength ((nnth (nfilter P (ndropns xs)) j))* $\leq$ *nlength xs*
using *assms ndropns-nfilter-nnth-exist-ndropn[of xs P j]*
by (*metis add.commute enat.simps(3) enat-add-sub-same enat-le-plus-same(1) less-eqE ndropn-nlength*)

lemma *ndropns-nfilter-ndropn*:
assumes *(Suc k)* $\leq$ *nlength nell*
$\exists$ *x* $\in$ *nset (ndropns (ndropn (Suc k) nell)). P x*
$\exists$ *x*$\in$*nset (ntaken k (ndropns nell)). P x*
shows *(nfilter P (ndropns (ndropn (Suc k) nell)))* =
(ndropn (the-enat (eSuc(nlength(nfilter P (ntaken k (ndropns nell)))))) (nfilter P (ndropns nell)))
using *assms*
ndropn-ndropns[of Suc k nell] ndropns-nlength[of nell] nfilter-nappend-ndropn[of k ndropns nell P]
by *simp*

lemma *ndropns-nfilter-ndropn-a*:
assumes *k* $\leq$ *nlength(nfilter P (ndropns nell))*
$\exists$ *x* $\in$ *nset (ndropns nell). P x*
shows *ndropn k (nfilter P (ndropns nell))* =
nfilter P (ndropns (ndropn (nnth (nkfilter P 0 (ndropns nell)) k) nell))
using *assms ndropn-ndropns nfilter-nkfilter-ndropn-1[of ndropns nell P k]*
nkfilter-upperbound[of ndropns nell P k 0]
by (*metis (mono-tags, lifting) add.left-neutral nkfilter-nlength zero-enat-def*)

lemma *nfilter-nlength-imp*:
assumes $\exists$ *x* $\in$ *nset nell. P x* $\wedge$ *Q x*

shows *nlength* (*nfilter* ($\lambda x.$ *P x* $\wedge$ *Q x*) *nell*) $\leq$ *nlength* (*nfilter P nell*)
using *assms* **by** *transfer* (*auto simp add*: *lfilter-nlength-imp epred-le-epredI*)

lemma *nkfilter-chop*:
assumes *nlast nellx* = *nfirst nelly*
P (*nlast nellx*)
nfinite nellx
shows *nkfilter P k* (*nfuse nellx nelly*) =
nfuse (*nkfilter P k nellx*) (*nkfilter P* (*k*+(*the-enat* (*nlength nellx*))) *nelly*)
using *assms*
proof (*auto simp add*: *nfuse-def1*)
show *nlast nellx* = *nfirst nelly* $\Longrightarrow$
P (*nfirst nelly*) $\Longrightarrow$
nfinite nellx $\Longrightarrow$
is-NNil nelly $\Longrightarrow$
$\neg$ *is-NNil* (*nkfilter P* (*k* + *the-enat* (*nlength nellx*)) *nelly*) $\Longrightarrow$
nkfilter P k nellx = *nappend* (*nkfilter P k nellx*) (*ntl* (*nkfilter P* (*k* + *the-enat* (*nlength nellx*)) *nelly*))
by *transfer auto*
show *nlast nellx* = *nfirst nelly* $\Longrightarrow$
P (*nfirst nelly*) $\Longrightarrow$
nfinite nellx $\Longrightarrow$ $\neg$ *is-NNil nelly* $\Longrightarrow$
is-NNil (*nkfilter P* (*k* + *the-enat* (*nlength nellx*)) *nelly*) $\Longrightarrow$
nkfilter P k (*nappend nellx* (*ntl nelly*)) = *nkfilter P k nellx*
apply *transfer*
proof (*auto simp add*: *kfilter-lappend-lfinite lappend-lnull2*)
fix *nellxa* :: *'a llist* **and** *nellya* :: *'a llist* **and** *Pa* :: *'a* $\Rightarrow$ *bool* **and** *ka* :: *nat* **and** *b* :: *nat*
assume *a1*: (*if lnull* (*kfilter Pa* (*ka* + *the-enat* (*epred* (*llength nellxa*))) *nellya*)
then iterates Suc (*ka* + *the-enat* (*epred* (*llength nellxa*)))
else kfilter Pa (*ka* + *the-enat* (*epred* (*llength nellxa*))) *nellya*) = *LCons b LNil*
assume *a2*: $\neg$ *lnull nellya*
assume *a3*: *Pa* (*lhd nellya*)
assume *a4*: $\neg$ *lnull* (*kfilter Pa* (*ka* + *the-enat* (*llength nellxa*)) (*ltl nellya*))
have *f5*: *lnull* (*LNil*::*nat llist*)
using *llength-LNil llength-eq-0* **by** *blast*
have *f6*: *nellya* = *LCons* (*lhd nellya*) (*ltl nellya*)
using *a2* **by** *auto*
then have *Pa* (*lhd nellya*)
using *a3* **by** *auto*
then have *False*
by (*metis a1 a2 a4 eq-LConsD f6 kfilter-code*(*2*) *kfilter-lnull-conv llength-LNil llength-eq-0 llist.set-sel*(*1*))
then show *lappend* (*kfilter Pa ka nellxa*) (*kfilter Pa* (*ka* + *the-enat* (*llength nellxa*)) (*ltl nellya*)) = *iterates Suc ka*
by *meson*
next
fix *nellxa* :: *'b llist* **and** *nellya* :: *'b llist* **and** *Pa* :: *'b* $\Rightarrow$ *bool* **and** *ka* :: *nat* **and** *b* :: *nat*
assume *a1*: (*if lnull* (*kfilter Pa* (*ka* + *the-enat* (*epred* (*llength nellxa*))) *nellya*)
then iterates Suc (*ka* + *the-enat* (*epred* (*llength nellxa*)))
else kfilter Pa (*ka* + *the-enat* (*epred* (*llength nellxa*))) *nellya*) = *LCons b LNil*
assume *a2*: $\neg$ *lnull nellya*
assume *a3*: *Pa* (*lhd nellya*)

assume *a4*: ¬ *lnull* (*kfilter Pa* (*ka* + *the-enat* (*llength nellxa*)) (*ltl nellya*))
have *f5*: *lnull* (*LNil*::*nat llist*)
using *llength-LNil llength-eq-0* **by** *blast*
have *f6*: *nellya* = *LCons* (*lhd nellya*) (*ltl nellya*)
using *a2* **by** *auto*
then have *Pa* (*lhd nellya*)
using *a3* **by** *auto*
then have *False*
using *f6 f5 a4 a1* **by** (*metis kfilter-LCons kfilter-llength-n-zero llength-eq-0 ltl-simps*(*2*) *not-lnull-conv*)
then show *lappend* (*kfilter Pa ka nellxa*) (*kfilter Pa* (*ka* + *the-enat* (*llength nellxa*)) (*ltl nellya*)) =
kfilter Pa ka nellxa
by *meson*
qed
next
show *nlast nellx* = *nfirst nelly* ⟹
P (*nfirst nelly*) ⟹
nfinite nellx ⟹
¬ *is-NNil nelly* ⟹
¬ *is-NNil* (*nkfilter P* (*k* + *the-enat* (*nlength nellx*)) *nelly*) ⟹
nkfilter P k (*nappend nellx* (*ntl nelly*)) =
nappend (*nkfilter P k nellx*) (*ntl* (*nkfilter P* (*k* + *the-enat* (*nlength nellx*)) *nelly*))
apply *transfer*
proof (*auto simp add*: *lappend-iterates kfilter-lnull-conv split*:*if-split-asm*)
show *f1*:⋀*nellx nelly P k x xa*.
¬ *lnull nellx* ⟹
¬ *lnull nelly* ⟹
llast nellx = *lhd nelly* ⟹
P (*lhd nelly*) ⟹
lfinite nellx ⟹
∀ *b*. *nelly* ≠ *LCons b LNil* ⟹
∀ *b*. *kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly* ≠ *LCons b LNil* ⟹
x ∈ *lset nelly* ⟹ *P x* ⟹ ∀ *x*∈*lset nellx*. ¬ *P x* ⟹ *P xa* ⟹ *xa* ∈ *lset* (*ltl nelly*) ⟹
kfilter P k (*lappend nellx* (*ltl nelly*)) = *iterates Suc k*
by (*metis in-lset-lappend-iff lappend-lbutlast-llast-id lbutlast-lfinite llist.set-intros*(*1*))
next
show *f2*:⋀*nellx nelly P k x xa xb*.
¬ *lnull nellx* ⟹
¬ *lnull nelly* ⟹
llast nellx = *lhd nelly* ⟹
P (*lhd nelly*) ⟹
lfinite nellx ⟹
∀ *b*. *nelly* ≠ *LCons b LNil* ⟹
∀ *b*. *kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly* ≠ *LCons b LNil* ⟹
x ∈ *lset nelly* ⟹
P x ⟹
xa ∈ *lset nellx* ⟹
P xa ⟹
P xb ⟹
xb ∈ *lset nellx* ⟹
kfilter P k (*lappend nellx* (*ltl nelly*)) =

lappend (*kfilter P k nellx*) (*ltl* (*kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly*))
proof −
fix *nellxa* :: *'b llist* **and** *nellya* :: *'b llist* **and** *Pa* :: *'b* ⇒ *bool* **and**
ka :: *nat* **and** *x* :: *'b* **and** *xa* :: *'b* **and** *xb* :: *'b*
assume *a1*: ¬ *lnull nellya*
assume *a2*: *Pa* (*lhd nellya*)
assume *a3*: *llast nellxa* = *lhd nellya*
assume *a4*: ¬ *lnull nellxa*
assume *a5*: *lfinite nellxa*
have *f6*: *nellya* = *LCons* (*lhd nellya*) (*ltl nellya*)
using *a1* **by** *auto*
have *f7*: *LCons* (*lhd nellya*) (*ltl nellya*) ≠ *LNil* ∧ *lhd* (*LCons* (*lhd nellya*) (*ltl nellya*)) = *lhd nellya* ∧
ltl (*LCons* (*lhd nellya*) (*ltl nellya*)) = *ltl nellya*
by *auto*
then have *f8*: *kfilter Pa* (*the-enat* (*epred* (*llength nellxa*)) + *ka*) (*LCons* (*lhd nellya*) (*ltl nellya*)) =
LCons (*the-enat* (*epred* (*llength nellxa*)) + *ka*)
(*kfilter Pa* (*Suc* (*the-enat* (*epred* (*llength nellxa*)) + *ka*)) (*ltl nellya*))
using *f6 a2* **by** (*metis* (*no-types*) *kfilter-LCons*)
have *f9*: ∀ *bs p n bsa*. ¬ *lfinite* (*bs*::*'b llist*) ∨ *kfilter p n* (*lappend bs bsa*) =
lappend (*kfilter p n bs*) (*kfilter p* (*n* + *the-enat* (*llength bs*)) *bsa*)
by (*meson kfilter-lappend-lfinite*)
have *f10*: *lfinite* (*lbutlast nellxa*)
using *a5* **by** (*meson lbutlast-lfinite*)
have *f11*: *lappend* (*kfilter Pa ka* (*lbutlast nellxa*))
(*kfilter Pa* (*ka* + *the-enat* (*llength* (*lbutlast nellxa*))) *LNil*) =
kfilter Pa ka (*lbutlast nellxa*)
by *simp*
have *LCons* (*the-enat* (*epred* (*llength nellxa*)) + *ka*) *LNil* =
kfilter Pa (*the-enat* (*epred* (*llength nellxa*)) + *ka*) (*LCons* (*lhd nellya*) *LNil*)
using *a2* **by** *simp*
then have *f12*: *lappend* (*lappend* (*kfilter Pa ka* (*lbutlast nellxa*))
(*kfilter Pa* (*ka* + *the-enat* (*llength* (*lbutlast nellxa*))) *LNil*))
(*LCons* (*the-enat* (*epred* (*llength nellxa*)) + *ka*) *LNil*) = *kfilter Pa ka nellxa*
using *f11 f10 f9 a5 a4 a3* **by** (*metis add.commute lappend-lbutlast-llast-id-lfinite llength-lbutlast*)
have *ltl* (*kfilter Pa* (*ka* + *the-enat* (*epred* (*llength nellxa*))) *nellya*) =
kfilter Pa (*Suc* (*the-enat* (*epred* (*llength nellxa*)) + *ka*)) (*ltl nellya*)
using *f8 f6* **by** (*simp add*: *add.commute*)
then show *kfilter Pa ka* (*lappend nellxa* (*ltl nellya*)) =
lappend (*kfilter Pa ka nellxa*) (*ltl* (*kfilter Pa* (*ka* + *the-enat* (*epred* (*llength nellxa*))) *nellya*))
using *f12 f11 f10 f9 f8 f7 f6 a5 a4 a3*
by (*metis add.commute lappend-lbutlast-llast-id-lfinite lappend-snocL1-conv-LCons2 llength-lbutlast*)
qed
show ⋀*nellx nelly P k x xa xb*.
¬ *lnull nellx* ⟹
¬ *lnull nelly* ⟹
llast nellx = *lhd nelly* ⟹
P (*lhd nelly*) ⟹
lfinite nellx ⟹
∀ *b*. *nelly* ≠ *LCons b LNil* ⟹
∀ *b*. *kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly* ≠ *LCons b LNil* ⟹

x ∈ *lset nelly* ⟹
P x ⟹
xa ∈ *lset nellx* ⟹
P xa ⟹
P xb ⟹
xb ∈ *lset* (*ltl nelly*) ⟹
kfilter P k (*lappend nellx* (*ltl nelly*)) =
lappend (*kfilter P k nellx*) (*ltl* (*kfilter P* (*k* + *the-enat* (*epred* (*llength nellx*))) *nelly*))
by (*simp add*: *f2*)
qed
qed

lemma *nleast-conv*:
assumes ∃ *x* ∈ *nset nellx*. *P x*
shows *nleast P nellx* = (*LEAST na*. *na* ≤ *nlength nellx* ∧ *P* (*nnth nellx na*))
using *assms*
by *transfer*
(*auto simp add*: *min-def lleast-def*,
metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0)

lemma *nfilter-chop*:
assumes *nlast nellx* = *nfirst nelly*
P (*nlast nellx*)
nfinite nellx
shows *nfilter P* (*nfuse nellx nelly*) = *nfuse* (*nfilter P nellx*) (*nfilter P nelly*)
proof (*cases is-NNil nelly*)
case *True*
then show *?thesis* **by** (*metis nfilter-NNil nfuse-def1 nfuse-leftneutral*)
next
case *False*
then show *?thesis*
proof (*cases is-NNil* (*nfilter P nelly*))
case *True*
then show *?thesis* **unfolding** *nfuse-def1* **using** *assms nfilter-nappend2*[*of ntl nelly P nellx*]
nfilter-expand[*of nelly P*]
by (*auto simp add*: *nfirst-def nellist.case-eq-if*)
(*metis nellist.discI*(*2*) *nellist.set-sel*(*2*) *nset-nlast*)
next
case *False*
then show *?thesis*
proof −
have *1*: ∃ *x*∈*nset nellx*. *P x*
using *assms nset-nlast* **by** *blast*
have *2*: ∃ *x*∈*nset* (*ntl nelly*). *P x*
using *False assms*
by (*metis nellist.collapse*(*1*) *nellist.collapse*(*2*) *nellist.disc*(*1*) *nfilter-NCons-a*
nfilter-NNil nnth-0 ntaken-0 ntaken-nlast)
have *3*: ¬*is-NNil* (*nfilter P nelly*) ⟹
nfilter P (*nappend nellx* (*ntl nelly*)) = *nappend* (*nfilter P nellx*) (*nfilter P* (*ntl nelly*))
by (*simp add*: *1 2 assms*(*3*) *nappend-nfilter-nfinite*)

have *4*: *is-NNil* (*nfilter P nelly*) $\Longrightarrow$ *nfilter P nellx* = *nappend* (*nfilter P nellx*) (*nfilter P* (*ntl nelly*))
  by (*simp add*: *False*)
have *5*: *nfilter P* (*nfuse nellx nelly*) = *nfilter P* (*nappend nellx* (*ntl nelly*))
  unfolding *nfuse-def1*
  by (*metis* (*full-types*) *False nellist.collapse*(*1*) *nfilter-NNil*)
have *6*: (*ntl* (*nfilter P nelly*)) = (*nfilter P* (*ntl nelly*))
  using *assms 2 False nfilter-expand*[*of nelly P*]
  by (*metis in-nset-ntlD nellist.case-eq-if nellist.sel*(*5*) *nfirst-def*)
show *?thesis*
  by (*metis 3 5 6 False nfuse-def1*)
qed
qed
qed

lemma *nfilter-chop1*:
assumes $n \leq$ *nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *nfilter P nellx* = *nfuse* (*nfilter P* (*ntaken n nellx*)) (*nfilter P* (*ndropn n nellx*))
using *assms*
by (*metis nfuse-ntaken-ndropn ndropn-nfirst nfilter-chop nfinite-ntaken ntaken-nlast*)

lemma *nfilter-chop1-ntaken*:
assumes $n \leq$ *nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *ntaken* (*the-enat* (*nlength*(*nfilter P* (*ntaken n nellx*)))) (*nfilter P nellx*) =
(*nfilter P* (*ntaken n nellx*))
using *assms nfilter-nappend-ntaken* **by** (*metis nfinite-ntaken nset-nlast*)

lemma *nfilter-nlast*:
assumes $n \leq$ *nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *nlast*(*nfilter P* (*ntaken n nellx*)) = (*nlast*(*ntaken n nellx*))
using *assms*
proof (*induction n arbitrary*: *nellx*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis* **by** *auto*
next
case (*NCons x21 x22*)
then show *?thesis* **using** *Suc*
by *auto*
(*metis Suc-ile-eq iless-Suc-eq nfilter-NCons nfinite-ntaken nlast-NCons nlength-NCons*
nset-nlast ntaken-Suc-NCons)
qed

qed

lemma *nfilter-nfirst*:
assumes *P* (*nfirst*(*ndropn n nellx*))
shows *nfirst*(*nfilter P* (*ndropn n nellx*)) = (*nfirst* (*ndropn n nellx*))
using *assms*
proof (*induction n arbitrary*: *nellx*)
case *0*
then show *?case*
by (*auto simp add*: *ndropn-nfirst*)
(*metis nellist.set-intros*(*1*) *nfilter-NNil nfilter-nappend-ntaken nlast-NNil nlength-NNil ntaken-0 the-enat-0 zero-enat-def zero-le*)
next
case (*Suc n*)
then show *?case*
by (*metis ndropn-ndropn plus-1-eq-Suc*)
qed

lemma *nfilter-chop1-ndropn*:
assumes *n*≤ *nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *ndropn* (*the-enat* (*nlength*(*nfilter P* (*ntaken n nellx*)))) (*nfilter P nellx*) =
(*nfilter P* (*ndropn n nellx*))
proof −
have *1*: (*nfilter P nellx*) = *nfuse* (*nfilter P* (*ntaken n nellx*)) (*nfilter P* (*ndropn n nellx*))
by (*simp add*: *assms nfilter-chop1*)
have *2*: *nlast*(*nfilter P* (*ntaken n nellx*)) = *nfirst* (*nfilter P* (*ndropn n nellx*))
by (*metis assms*(*1*) *assms*(*2*) *ndropn-nfirst nfilter-nfirst nfilter-nlast ntaken-nlast*)
have *3*: *ndropn* (*the-enat* (*nlength*(*nfilter P* (*ntaken n nellx*))))
(*nfuse* (*nfilter P* (*ntaken n nellx*)) (*nfilter P* (*ndropn n nellx*))) =
(*nfilter P* (*ndropn n nellx*))
by (*metis 2 assms*(*1*) *enat-the-enat infinity-ileE length-nfilter-le min-absorb1 ndropn-nfuse nfinite-conv-nlength-enat ntaken-nlength*)
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *nkfilter-chop1*:
assumes (*enat n*) ≤ *nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *nkfilter P k nellx* = *nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
using *assms nfuse-ntaken-ndropn*[*of n nellx*] *nkfilter-chop*[*of* (*ntaken n nellx*) (*ndropn n nellx*) *P k*]
by (*metis min.orderE ndropn-nfirst nfinite-ntaken ntaken-nlast ntaken-nlength the-enat.simps*)

lemma *nkfilter-nlast*:
assumes *n*≤ *nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *nlast*(*nkfilter P k* (*ntaken n nellx*)) = *k*+*n*
using *assms*
proof (*induction n arbitrary*: *k nellx*)
case *0*

then show *?case* **by** *simp*
next
case (*Suc n*)
then show *?case*
 proof (*cases nellx*)
 case (*NNil x1*)
 then show *?thesis*
 using *Suc.prems*(*1*) *enat-0-iff* (*1*) **by** *auto*
 next
 case (*NCons x21 x22*)
 then show *?thesis*
   using *Suc*
   proof *auto*
   assume *a1*: P (*nlast* (*ntaken n x22*))
   assume *a2*: $\bigwedge$*nellx k*. $[\![$*enat n* $\leq$ *nlength nellx*; P (*nlast* (*ntaken n nellx*))$]\!] \Longrightarrow$
         nlast (*nkfilter P k* (*ntaken n nellx*)) = $k + n$
   assume *enat* (*Suc n*) $\leq$ *eSuc* (*nlength x22*)
   then have *enat n* $<$ *eSuc* (*nlength x22*)
   using *Suc-ile-eq* **by** *blast*
   then have *f3*: *enat n* $\leq$ *nlength x22*
   by (*meson iless-Suc-eq*)
   have $\exists$ *a*. $a \in$ *nset* (*ntaken n x22*) $\wedge$ P *a*
   using *a1 nfinite-ntaken nset-nlast* **by** *blast*
   then show *nlast* (*nkfilter P k* (*NCons x21* (*ntaken n x22*))) = *Suc* ($k + n$)
   using *f3 a2 a1* **by** (*simp add*: *Bex-def-raw*)
   qed
 qed
qed

lemma *nkfilter-nfirst*:
 assumes P (*nfirst*(*ndropn n nellx*))
 shows *nfirst*(*nkfilter P k* (*ndropn n nellx*)) = k
 using *assms*
proof (*induction n arbitrary*: *k nellx*)
case *0*
then show *?case*
 by (*metis enat-defs*(*1*) *nellist.set-intros*(*1*) *nkfilter-NNil nkfilter-nappend-ntaken nlength-NNil nnth-NNil ntaken-0 the-enat.simps zero-le*)
next
case (*Suc n*)
then show *?case*
proof (*cases nellx*)
case (*NNil x1*)
then show *?thesis* **using** *Suc*
by (*metis ndropn-ndropn plus-1-eq-Suc*)
next
case (*NCons x21 x22*)
then show *?thesis* **using** *Suc*
by *auto*
qed

qed

lemma *nkfilter-chop1-ndropn*:
assumes *n≤ nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *ndropn* (*the-enat* (*nlength*(*nkfilter P k* (*ntaken n nellx*)))) (*nkfilter P k nellx*) =
(*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
proof −
have *1*: (*nkfilter P k nellx*) = *nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*simp add*: *assms nkfilter-chop1*)
have *2*: *nlast*(*nkfilter P k* (*ntaken n nellx*)) = *nfirst* (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*metis assms*(*1*) *assms*(*2*) *ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast*)
have *3*: *ndropn* (*the-enat* (*nlength*(*nkfilter P k* (*ntaken n nellx*))))
(*nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))) =
(*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*metis 2 assms*(*2*) *enat-the-enat infinity-ileE ndropn-nfuse nfinite-conv-nlength-enat*
nfinite-ntaken nlength-nkfilter-le nset-nlast)
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *nkfilter-chop1-ntaken*:
assumes *n≤ nlength nellx*
P (*nlast* (*ntaken n nellx*))
shows *ntaken* (*the-enat* (*nlength*(*nkfilter P k* (*ntaken n nellx*)))) (*nkfilter P k nellx*) =
(*nkfilter P k* (*ntaken n nellx*))
proof −
have *1*: (*nkfilter P k nellx*) = *nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*simp add*: *assms nkfilter-chop1*)
have *2*: *nlast*(*nkfilter P k* (*ntaken n nellx*)) = *nfirst* (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))
by (*metis assms*(*1*) *assms*(*2*) *ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast*)
have *3*: *ntaken* (*the-enat* (*nlength*(*nkfilter P k* (*ntaken n nellx*))))
(*nfuse* (*nkfilter P k* (*ntaken n nellx*)) (*nkfilter P* (*k*+*n*) (*ndropn n nellx*))) =
(*nkfilter P k* (*ntaken n nellx*))
by (*metis 1 assms*(*1*) *assms*(*2*) *nfinite-ntaken nkfilter-nappend-ntaken nset-nlast*)
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *nfilter-nsubn*:
assumes *enat j* ≤ *nlength nellx*
P (*nnth nellx j*)
enat i ≤ *nlength nellx*
P (*nnth nellx i*)
i ≤ *j*
enat ni = (*nlength*(*nfilter P* (*ntaken i nellx*)))
enat nj = (*nlength*(*nfilter P* (*ntaken j nellx*)))
shows (*nfilter P* (*nsubn nellx i j*)) = (*nsubn* (*nfilter P nellx*) *ni nj*)
proof −
have *1*: (*nfilter P* (*nsubn nellx i j*)) = (*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*)))
by (*simp add*: *nsubn-def1*)
have *2*: (*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*))) =

ntaken (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*)))))
(*nfilter P* (*ndropn i nellx*))
by (*metis assms*(*2*) *assms*(*5*) *enat-ile le-add-diff-inverse2 linorder-le-cases nfilter-chop1-ntaken ntaken-all ntaken-ndropn-nlast*)
have *3*: *ntaken* (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*)))))
(*nfilter P* (*ndropn i nellx*)) =
ntaken (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*)))))
(*ndropn* (*the-enat* (*nlength*(*nfilter P* (*ntaken i nellx*)))) (*nfilter P nellx*))
by (*simp add*: *assms*(*3*) *assms*(*4*) *nfilter-chop1-ndropn ntaken-nlast*)
have *4*: *ntaken* (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*)))))
(*ndropn* (*the-enat* (*nlength*(*nfilter P* (*ntaken i nellx*)))) (*nfilter P nellx*)) =
nsubn (*nfilter P nellx*)
(*the-enat* (*nlength*(*nfilter P* (*ntaken i nellx*))))
((*the-enat* (*nlength*(*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*)))))+
(*the-enat* (*nlength*(*nfilter P* (*ntaken i nellx*)))))
using *ntaken-ndropn* **by** *blast*
have *5*: ((*the-enat* (*nlength*(*nfilter P* (*ntaken* (*j*−*i*) (*ndropn i nellx*)))))+
(*the-enat* (*nlength*(*nfilter P* (*ntaken i nellx*))))) =
((*the-enat* (*nlength*(*nfilter P* (*nsubn nellx i j*))))+
(*the-enat* (*nlength*(*nfilter P* (*ntaken i nellx*)))))
using *1* **by** *auto*
have *6*: *ntaken j nellx* = *nfuse* (*ntaken i nellx*) (*nsubn nellx i j*)
using *nsubn-nfuse*[*of 0 i j nellx*]
by (*simp add*: *assms*(*1*) *assms*(*5*) *nsubn-def1*)
have *7*: *nlength* (*nfilter P* (*nfuse* (*ntaken i nellx*) (*nsubn nellx i j*))) =
(*nlength* (*nfilter P* (*ntaken i nellx*)) + *nlength* (*nfilter P* (*nsubn nellx i j*)))
by (*simp add*: *assms*(*4*) *ndropn-nfirst nfilter-chop nfuse-nlength nsubn-def1 ntaken-nfirst ntaken-nlast*)
have *70*: *nfinite* (*nfilter P* (*nfuse* (*ntaken i nellx*) (*nsubn nellx i j*)))
by (*metis 6 assms*(*1*) *assms*(*2*) *nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast*)
have *71*: *nfinite* (*nfilter P* (*ntaken i nellx*))
by (*metis assms*(*3*) *assms*(*4*) *nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast*)
have *72*: *nfinite* (*nsubn nellx i j*)
by (*simp add*: *nsubn-def1*)
have *8*: *nj* =
((*the-enat* (*nlength*(*nfilter P* (*nsubn nellx i j*))))+ *ni*)
by (*metis 6 7 add.commute assms*(*6*) *assms*(*7*) *enat-le-plus-same*(*2*) *enat-the-enat infinity-ileE plus-enat-simps*(*1*) *the-enat.simps*)
show *?thesis*
using *1 2 3 4 8* **by** (*metis assms*(*6*) *the-enat.simps*)
qed

lemma *nfilter-nsubn-zero*:
assumes *enat j* ≤ *nlength nellx*
P (*nnth nellx j*)
shows (*nfilter P* (*nsubn nellx 0 j*)) =
(*nsubn* (*nfilter P nellx*)
0
(*the-enat* (*nlength*(*nfilter P* (*ntaken j nellx*))))
)

using *assms*
by (*simp add*: *nfilter-chop1-ntaken nsubn-def1 ntaken-nlast*)

end

context includes *lifting-syntax*
begin
lemma *ndropns-transfer2* [*transfer-rule*]:
(*nellist-all2 A* ===> *nellist-all2* (*nellist-all2 A*)) *ndropns ndropns*
unfolding *rel-fun-def*
by (*auto intro*: *nellist-all2-ndropnsI*)

end

end

4 Finite and Infinite ITL Semantics

theory *Semantics*
imports *NELList-Extras HOL−TLA.Intensional*
begin

This theory mechanises a *shallow* embedding of Finite and Infinite ITL using the *NELList* and *Intensional* theories.

4.1 Types of Formulas

To mechanise the Finite and Infinite ITL semantics, the following type abbreviations are used:
type-synonym *'a intervals* = *'a nellist*

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

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

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

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

instance
nellist :: (*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.

4.2 Semantics of ITL

The semantics of ITL is defined.

definition *skip-d* :: ('*a* ::*world*) *formula*
where *skip-d* ≡ (λ*s*. *nlength s* = (*enat* (*Suc 0*)))

definition *chop-d* :: ('*a* ::*world*) *formula* ⇒ ('*a* ::*world*) *formula* ⇒ ('*a* ::*world*) *formula*
where *chop-d F1 F2* ≡
    (λ*s*.
        (∃ *n*≤*nlength s*. ((*ntaken n s*) ⊨ *F1*) ∧ ((*ndropn n s*) ⊨ *F2*))
        ∨ (¬ *nfinite s* ∧ (*s* ⊨ *F1*))
)

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

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

definition *fin-val-d* :: ('*a*::*world*,'*b*) *stfun* ⇒ ('*a*,'*b*) *formfun*
where *fin-val-d f* ≡ λ *s*. ((*if nfinite s then* ((*nlast s*) ⊨ *f*) *else* (ϵ (*x*::'*b*). *x*=*x*)))

definition *penult-val-d* :: ('*a*::*world*,'*b*) *stfun* ⇒ ('*a*,'*b*) *formfun*
where *penult-val-d f* ≡
  (λ *s*.
   (*if nfinite s*
    then (*if nlength s* ≠ (*enat 0*)
        then ((*nnth s* (*the-enat* (*epred*(*nlength s*)))) ⊨ *f*)
        else (ϵ (*x*::'*b*). *x*=*x*))
    else (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*) ⇒ -)

4.3 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* $\equiv$ *LIFT*(*finite*;(*F*;#*True*))

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

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

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

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

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

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

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

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

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

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

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

where *wnext-d F* $\equiv$ *LIFT*$(\neg(\bigcirc(\neg F)))$

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

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

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

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

translations
-*df-d* $\rightleftharpoons$ *CONST df-d*
-*sda-d* $\rightleftharpoons$ *CONST sda-d*
-*always-d* $\rightleftharpoons$ *CONST always-d*
-*bi-d* $\rightleftharpoons$ *CONST bi-d*
-*ba-d* $\rightleftharpoons$ *CONST ba-d*
-*wnext-d* $\rightleftharpoons$ *CONST wnext-d*
-*wprev-d* $\rightleftharpoons$ *CONST wprev-d*
-*more-d* $\rightleftharpoons$ *CONST more-d*

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

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

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

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* ⇒ $'a$ *formula*
where *initonly-d F* ≡ *LIFT*(*bi*(*empty* = *F*))

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

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

definition *syields-d* :: (*'a::world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *syields-d F1 F2* ≡ *LIFT*(¬(*F1*⌢(¬*F2*)))

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

syntax
-bm-d :: *lift* ⇒ *lift* ((*bm* -) [*88*] *87*)
-init-d :: *lift* ⇒ *lift* ((*init* -) [*88*] *87*)
-fin-d :: *lift* ⇒ *lift* ((*fin* -) [*88*] *87*)
-halt-d :: *lift* ⇒ *lift* ((*halt* -) [*88*] *87*)
-initonly-d :: *lift* ⇒ *lift* ((*initonly* -) [*88*] *87*)
-keep-d :: *lift* ⇒ *lift* ((*keep* -) [*88*] *87*)
-yields-d :: [*lift*,*lift*] ⇒ *lift* ((- *yields* -) [*88*,*88*] *87*)
-syields-d :: [*lift*,*lift*] ⇒ *lift* ((- *syields* -) [*88*,*88*] *87*)
-ifthenelse-d :: [*lift*,*lift*,*lift*] ⇒ *lift* ((if_i - *then* - *else* -) [*88*,*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*)
-syields-d :: [*lift*,*lift*] ⇒ *lift* ((- *syields* -) [*88*,*88*] *87*)
-ifthenelse-d :: [*lift*,*lift*,*lift*] ⇒ *lift* ((if_i - *then* - *else* -) [*88*,*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*

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-d F G* ≡ *LIFT*(if_i *F then G else #True*)

syntax

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

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

-sfin-d :: *lift* ⇒ *lift* ((*sfin* -) [*88*] *87*)

translations
-ifthen-d ⇌ *CONST ifthen-d*
-sfin-d ⇌ *CONST sfin-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*(*finite* ⟶ *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* ⊨ *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
-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*

lemmas *itl-def* = *skip-d-def chop-d-def current-val-d-def next-val-d-def fin-val-d-def penult-val-d-def infinite-d-def finite-d-def schop-d-def sometimes-d-def di-d-def da-d-def next-d-def prev-d-def df-d-def sda-d-def always-d-def bi-d-def ba-d-def wnext-d-def wprev-d-def more-d-def bf-d-def sba-d-def empty-d-def fmore-d-def dm-d-def bm-d-def init-d-def fin-d-def halt-d-def initonly-d-def keep-d-def yields-d-def syields-d-def ifthenelse-d-def sfin-d-def ifthen-d-def next-assign-d-def prev-assign-d-def always-eq-d-def temporal-assign-d-def gets-d-def stable-d-def padded-d-def padded-temp-assign-d-def*

4.4 Properties of Operators

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

lemma *skip-defs* :
$(w \models skip) = (nlength\ w = (enat\ 1))$
by (*simp add*: *itl-def*)

lemma *chop-defs* :
$(w \models F1 ; F2) =$
(
$(\exists\, n \le nlength\ w.\ ((ntaken\ n\ w) \models F1)\ \wedge\ ((ndropn\ n\ w) \models F2))$
$\vee (\neg\ nfinite\ w \wedge (w \models F1))$
)
by (*simp add*: *itl-def*)

lemma *yields-defs* :
$(w \models F1\ yields\ F2) =$
$((\forall\ n.\ ((ntaken\ n\ w) \models F1) \longrightarrow enat\ n \le nlength\ w \longrightarrow (ndropn\ n\ w \models F2)) \wedge (nfinite\ w \vee (w \models \neg F1)))$

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

lemma *infinite-defs*:
$(w \models inf) = (\neg\ nfinite\ w)$
by (*simp add*: *itl-def*)

lemma *finite-defs* :
$(w \models \mathit{finite}) = (\mathit{nfinite}\ w)$
by (*simp add*: *itl-def*)

lemma *schop-defs* :
$(w \models F1 \frown F2) =$
(
$(\exists n \le \mathit{nlength}\ w.\ ((\mathit{ntaken}\ n\ w) \models F1\ )\ \wedge ((\mathit{ndropn}\ n\ w) \models F2\ )\ )$
)

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

lemma *syields-defs* :
$(w \models F1\ \mathit{syields}\ F2) =$
$(\forall\ n. ((\mathit{ntaken}\ n\ w) \models F1) \longrightarrow \mathit{enat}\ n \le \mathit{nlength}\ w \longrightarrow ((\mathit{ndropn}\ n\ w \models F2)))$

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

lemma *sometimes-defs* :
$(w \models \diamond\ F) = (\exists\ n \le \mathit{nlength}\ w.\ ((\mathit{ndropn}\ n\ w)) \models F)$
by (*simp add*: *itl-def finite-defs chop-defs*)

lemma *always-defs* :
$(w \models \square\ F) =$
$(\forall\ n \le \mathit{nlength}\ w.\ ((\mathit{ndropn}\ n\ w) \models F))$
by (*simp add*: *itl-def sometimes-defs*)

lemma *di-defs* :
$(w \models \mathit{di}\ F) =$
$((\exists\ n \le \mathit{nlength}\ w.\ ((\mathit{ntaken}\ n\ w) \models F)) \vee (\neg \mathit{nfinite}\ w \wedge (w \models F)))$
by (*simp add*: *itl-def*)

lemma *df-defs* :
$(w \models \mathit{df}\ F) =$
$(\exists\ n \le \mathit{nlength}\ w.\ ((\mathit{ntaken}\ n\ w) \models F))$
by (*simp add*: *df-d-def schop-defs*)

lemma *bi-defs* :
$(w \models \mathit{bi}\ F) =$
$((\forall\ n \le \mathit{nlength}\ w\ .\ ((\mathit{ntaken}\ n\ w) \models F)) \wedge (\mathit{nfinite}\ w \vee (w \models F)))$
by (*simp add*: *itl-def di-defs*)

lemma *bf-defs* :
$(w \models \mathit{bf}\ F) =$
$(\forall\ n \le \mathit{nlength}\ w. ((\mathit{ntaken}\ n\ w) \models F))$
by (*simp add*: *bf-d-def df-defs*)

lemma *da-defs* :
$(w \models \mathit{da}\ F) =$

(($\exists$ *n na*. (*n*+*na* $\leq$ *nlength w* $\wedge$ ((*nsubn w n* (*n*+ *na*)) $\models$ *F*)) $\vee$ ($\neg$*nfinite w* $\wedge$ ((*ndropn n w*) $\models$ *F*)))
)
proof
(*auto simp add*: *itl-def chop-defs nsubn-def1*)
show $\bigwedge$*n na*.
enat n $\leq$ *nlength w* $\Longrightarrow$
enat na $\leq$ *nlength w* $-$ *enat n* $\Longrightarrow$
F (*ntaken na* (*ndropn n w*)) $\Longrightarrow$ $\exists$ *n*. ($\exists$ *na*. *enat* (*n* + *na*) $\leq$ *nlength w* $\wedge$ *F* (*ntaken na* (*ndropn n w*)))
$\vee$ $\neg$ *nfinite w* $\wedge$ *F* (*ndropn n w*)
by (*metis add-left-mono enat.simps*(*3*) *enat-add-sub-same le-iff-add plus-enat-simps*(*1*))
next
fix *n* :: *nat* **and** *na* :: *nat*
assume *a1*: *enat* (*n* + *na*) $\leq$ *nlength w*
assume *a2*: *F* (*ntaken na* (*ndropn n w*))
have *enat n* $\leq$ *nlength w*
using *a1* **by** (*meson enat-ord-simps*(*1*) *le-iff-add order-subst2*)
then show $\exists$ *n*. *enat n* $\leq$ *nlength w* $\wedge$
(($\exists$ *na*. *enat na* $\leq$ *nlength w* $-$ *enat n* $\wedge$ *F* (*ntaken na* (*ndropn n w*)))
$\vee$ $\neg$ *nfinite w* $\wedge$ *F* (*ndropn n w*))
using *a2 a1* **by** (*metis* (*no-types*) *add-diff-cancel-left*$'$ *enat-minus-mono1 idiff-enat-enat*)
next
show $\bigwedge$*n*. $\neg$ *nfinite w* $\Longrightarrow$
F (*ndropn n w*) $\Longrightarrow$
$\exists$ *n*. *enat n* $\leq$ *nlength w* $\wedge$
(($\exists$ *na*. *enat na* $\leq$ *nlength w* $-$ *enat n* $\wedge$ *F* (*ntaken na* (*ndropn n w*))) $\vee$ *F* (*ndropn n w*))
by (*meson enat-ile le-cases nfinite-conv-nlength-enat*)
qed

lemma *ba-defs* :
(*w* $\models$ *ba F*) =
($\forall$ *n na*. (*enat* (*n* + *na*) $\leq$ *nlength w* $\longrightarrow$ ((*nsubn w n* (*n*+*na*)) $\models$ *F*))
$\wedge$ (*nfinite w* $\vee$ ((*ndropn n w*) $\models$ *F*)))
by (*simp add*: *ba-d-def da-defs*)

lemma *sda-defs* :
(*w* $\models$ *sda F*) =
($\exists$ *n na*. (*n*+*na* $\leq$ *nlength w* $\wedge$ ((*nsubn w n* (*n*+ *na*)) $\models$ *F*)))
proof
(*auto simp add*: *sda-d-def schop-defs nsubn-def1*)
show $\bigwedge$*n na*.
enat n $\leq$ *nlength w* $\Longrightarrow$
enat na $\leq$ *nlength w* $-$ *enat n* $\Longrightarrow$ *F* (*ntaken na* (*ndropn n w*)) $\Longrightarrow$
$\exists$ *n na*. *enat* (*n* + *na*) $\leq$ *nlength w* $\wedge$ *F* (*ntaken na* (*ndropn n w*))
by (*metis add-le-cancel-left enat-ord-simps*(*1*) *idiff-enat-enat le-add-diff-inverse le-cases nfinite-nlength-enat nfinite-ntaken ntaken-all*)
next
fix *n* :: *nat* **and** *na* :: *nat*
assume *a1*: *F* (*ntaken na* (*ndropn n w*))
assume *a2*: *enat* (*n* + *na*) $\leq$ *nlength w*

have *enat na* $\leq$ *nlength w* $-$ *enat n*
by (*metis a2 add-diff-cancel-left′ enat-minus-mono1 idiff-enat-enat*)
then show $\exists$ *n. enat n* $\leq$ *nlength w* $\wedge$
($\exists$ *na. enat na* $\leq$ *nlength w* $-$ *enat n* $\wedge$ *F* (*ntaken na* (*ndropn n w*)))
using *a2 a1*
by (*metis add.right-neutral le-cases le-zero-eq ndropn-all ndropn-nlength nlength-NNil the-enat.simps the-enat-0*)
qed

lemma *sba-defs* :
(*w* $\models$ *sba F*) =
($\forall$ *n na. n+na* $\leq$ *nlength w* $\longrightarrow$ ((*nsubn w n* (*n+na*)) $\models$ *F*))
by (*simp add*: *sba-d-def sda-defs*)

lemma *next-defs* :
(*w* $\models$ $\bigcirc$ *F*) =
(*nlength w* $\neq$ (*enat 0*) $\wedge$ ((*ndropn 1 w*)$\models$ *F*))
by (*simp add*: *itl-def chop-defs*)
(*metis One-nat-def Suc-ile-eq dual-order.order-iff-strict enat-le-plus-same*(*1*) *gen-nlength-def min.orderE min-enat-simps*(*2*) *nlength-code nlength-eq-enat-nfiniteD one-enat-def the-enat.simps zero-enat-def zero-one-enat-neq*(*1*))

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

lemma *prev-defs* :
(*w* $\models$ *prev F*) =
((*nlength w* $\neq$ (*enat 0*) $\wedge$ *nfinite w* $\wedge$
((*ntaken* (*the-enat*((*epred*(*nlength w*)))) *w*) $\models$ *F*)) $\vee$ ($\neg$*nfinite w* $\wedge$ (*w* $\models$ *F*)))
proof (*cases nfinite w*)
case *True*
then show *?thesis*
by (*auto simp add*: *prev-d-def chop-defs skip-defs*)
(*metis One-nat-def diff-diff-cancel enat-ord-simps*(*1*) *epred-enat idiff-enat-enat nfinite-nlength-enat the-enat.simps*,
metis One-nat-def diff-le-self eSuc-epred enat.simps(*3*) *enat-add-sub-same enat-ord-simps*(*1*) *epred-enat nfinite-nlength-enat one-enat-def plus-1-eSuc*(*2*) *the-enat.simps zero-enat-def*)
next
case *False*
then show *?thesis*
by (*auto simp add*: *prev-d-def chop-defs skip-defs*)
(*metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD*)
qed

lemma *wprev-defs* :
(*w* $\models$ *wprev F*) =
((*nfinite w* $\longrightarrow$ (*nlength w* = (*enat 0*) $\vee$ ((*ntaken* (*the-enat*(*epred*(*nlength w*))) *w*) $\models$ *F*)))
$\wedge$ (*nfinite w* $\vee$ (*w* $\models$ *F*)))

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

lemma *more-defs* :
  $(w \models more) = (\ 0 < nlength\ w)$
using *zero-enat-def* **by** (*auto simp add*: *more-d-def next-defs*)

lemma *fmore-defs* :
  $(w \models fmore) = (nfinite\ w \wedge (0 < nlength\ w))$
by (*auto simp add*: *fmore-d-def more-defs finite-defs*)

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

lemma *init-defs* :
  $(w \models init\ F) = (\ (ntaken\ 0\ w) \models F\ )$
by (*simp add*: *init-d-def chop-defs empty-defs min-def*)
 (*metis ntaken-0 ntaken-all the-enat.simps zero-enat-def zero-le*)

lemma *initalt-defs* :
  $(w \models bi(\ empty \longrightarrow F)) = (\ (ntaken\ 0\ w) \models F\ )$
by (*simp add*: *bi-defs empty-defs min-def*)
  (*metis enat-0-iff*(*2*) *nlength-eq-enat-nfiniteD ntaken-0 zero-le*)

lemma *fin-defs* :
  $(w \models fin\ F) =$
   $(\ \ (nfinite\ w \wedge (\ (ndropn\ (the\text{-}enat(nlength\ w))\ w) \models F)\ ) \vee (\neg nfinite\ w)\ )$
by (*simp add*: *fin-d-def empty-defs always-defs*)
  (*metis add.right-neutral enat.distinct*(*2*) *enat-add-sub-same le-iff-add nfinite-nlength-enat*
   nlength-eq-enat-nfiniteD the-enat.simps)

lemma *finalt-defs* :
  $(w \models \#True;(F \wedge empty)) =$
   $(\ \ (nfinite\ w \wedge (\ (ndropn\ (the\text{-}enat(nlength\ w))\ w) \models F)\ ) \vee (\neg nfinite\ w)\ )$
by (*simp add*: *chop-defs empty-defs*)
  (*metis add.right-neutral enat.distinct*(*2*) *enat-add-sub-same le-iff-add nfinite-nlength-enat*
   the-enat.simps)

lemma *sfin-defs* :
  $(w \models sfin\ F) = (nfinite\ w \wedge (\ (ndropn\ (the\text{-}enat(nlength\ w))\ w) \models F)\ )$
by (*auto simp add*: *sfin-d-def fin-defs*)

lemma *halt-defs* :
  $(w \models halt(F)) = (\forall n.\ n \le nlength\ w \longrightarrow (nlength\ w = n) = (\ \ (ndropn\ n\ w) \models F))$

by (*simp add*: *halt-d-def empty-defs always-defs*)
  (*metis add.right-neutral dual-order.strict-iff-order enat-add-sub-same enat-ord-code*(*4*)
   le-iff-add)

lemma *initonly-defs* :
  $(w \models initonly(F)) =$
   $((\forall n.\ n \le nlength\ w \longrightarrow (n = 0) = ((ntaken\ \ n\ w) \models F\ )) \wedge$
     $(nfinite\ w \vee (nlength\ w = 0) = F\ w)$
   $)$
  by (*simp add*: *initonly-d-def bi-defs empty-defs zero-enat-def*)

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

lemma *currentval-defs* :
  $(s \models \$v) = (v\ (nfirst\ s))$
by (*simp add*: *itl-def*)

lemma *nextval-defs* :
 $(s \models v\$) =$
   $(if\ nlength\ s \neq (enat\ 0)\ then\ (v\ (nnth\ s\ 1))\ else\ (\epsilon\ \ x.\ x{=}x))$
by (*simp add*: *itl-def*)

lemma *finval-defs* :
  $(s \models !v) =$
   $((if\ nfinite\ s\ then\ (v\ (nlast\ s))\ else\ (\epsilon\ \ x.\ x{=}x))$
   $)$
by (*simp add*: *itl-def*)

lemma *penultval-defs* :
  $(s \models v!) =$
   $(if\ nfinite\ s\ then$
     $(if\ nlength\ s \neq (enat\ 0)\ then\ (v\ (nnth\ s\ (the\text{-}enat(epred(nlength\ s)))))\ else\ (\epsilon\ \ x.\ x{=}x))$
    $else\ (\epsilon\ \ x.\ x{=}x)$
   $)$
by (*simp add*: *itl-def*)

lemma *next-assign-defs* :
  $(s \models v := e) = ((if\ nlength\ s \neq (enat\ 0)\ then\ v\ (nnth\ s\ 1)\ else\ (\epsilon\ \ x.\ x{=}x)) = e\ s)$
by (*auto simp*: *itl-def*)

lemma *prev-assign-defs* :
  $(s \models v =: e) =$
  $(if\ nfinite\ s\ then$
    $(if\ nlength\ s \neq (enat\ 0)\ then\ (v\ (nnth\ s\ (the\text{-}enat(epred(nlength\ s))))) = e\ s$
                              $else\ ((\epsilon\ \ x.\ x{=}x) = e\ s))$
   $else\ \ True$
   $)$
by (*simp add*: *itl-def finite-defs*)

lemma *always-eqv-defs* :
  $(s \models v \approx e) =$
   $((\forall\ \ i.\ i \leq nlength\ s \longrightarrow v\ (nnth\ s\ i) = e\ (ndropn\ i\ s))$
   $)$
by (*simp add*: *always-eq-d-def always-defs current-val-d-def ndropn-nfirst*)

lemma *temporal-assign-defs* :
  $(s \models v \leftarrow e) =$
   (*if nfinite s then* $(v\ (nlast\ s)) = e\ s$
                else True
)
by (*simp add*: *itl-def finite-defs*)

lemma *gets-defs* :
  $(s \models v\ gets\ e) =$
  $((\forall\ \ i.\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = e\ ((nsubn\ s\ i\ (Suc\ i)))\ )$

)
proof (*auto simp add*: *min-def finite-defs gets-d-def keep-d-def ba-defs skip-defs temporal-assign-d-def fin-val-d-def nsubn-nlength Suc-ile-eq nsubn-def1 ntaken-ndropn-nlast*)
show $\bigwedge i.\ \forall n\ na.\ (enat\ na \leq nlength\ s - enat\ n \longrightarrow$
                $enat\ (n + na) \leq nlength\ s \longrightarrow$
                $na = Suc\ 0 \longrightarrow v\ (nnth\ s\ (Suc\ n)) = e\ (ntaken\ (Suc\ 0)\ (ndropn\ n\ s))) \land$
              $(\neg\ enat\ na \leq nlength\ s - enat\ n \longrightarrow$
               $enat\ (n + na) \leq nlength\ s \longrightarrow$
              $nlength\ s - enat\ n = enat\ (Suc\ 0) \longrightarrow$
              $v\ (nnth\ s\ (na + n)) = e\ (ntaken\ na\ (ndropn\ n\ s))) \Longrightarrow$
        $enat\ i < nlength\ s \Longrightarrow$
        $v\ (nnth\ s\ (Suc\ i)) = e\ (ntaken\ (Suc\ 0)\ (ndropn\ i\ s))$
  by (*metis One-nat-def add.commute eSuc-enat i0-less ileI1 iless-Suc-eq ndropn-Suc-conv-ndropn ndropn-nlength nlength-NCons one-eSuc one-enat-def plus-1-eq-Suc zero-le*)
show $\bigwedge n\ na.$
     $\forall i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = e\ (ntaken\ (Suc\ 0)\ (ndropn\ i\ s)) \Longrightarrow$
     $\neg\ na \leq Suc\ 0 \Longrightarrow enat\ (n + na) \leq nlength\ s \Longrightarrow nlength\ s - enat\ n = enat\ (Suc\ 0) \Longrightarrow$
      $v\ (nnth\ s\ (na + n)) = e\ (ntaken\ na\ (ndropn\ n\ s))$
  by (*metis* (*no-types*) *add-diff-cancel-left′ enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat*)
qed

lemma *stable-defs-helpa*:
assumes $(\forall i.\ i<nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i))$
        $i \leq nlength\ s$
 shows $(v\ (nnth\ s\ i) = v\ (nfirst\ s))$
using *assms*
proof (*induct i arbitrary*:*s*)
case *0*
then show *?case* **by** (*metis ndropn-0 ndropn-nfirst*)
next
case (*Suc i*)
then show *?case*
by (*simp add*: *Suc-ile-eq*)

qed

lemma *stable-defs-helpb*:
assumes $(\forall\, i.\ i \le nlength\ s \longrightarrow v\ (nnth\ s\ i) = v\ (nfirst\ s))$
$i < nlength\ s$
shows $v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)$
using *assms*
proof (*induct i arbitrary*:*s*)
case *0*
then show *?case* **using** *Suc-ile-eq* **by** *auto*
next
case (*Suc i*)
then show *?case* **by** (*metis eSuc-enat ileI1 less-imp-le*)
qed

lemma *stable-defs-help*:
$(\forall\, i.\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)) =$
$(\forall\, i.\ i \le nlength\ s \longrightarrow v\ (nnth\ s\ i) = v\ (nfirst\ s))$
using *stable-defs-helpa*[*of s v*] *stable-defs-helpb*[*of s v*]
by *blast*

lemma *stable-defs*:
$(s \models stable\ v) =$
$(\forall\ \ i.\ i \le nlength\ s \longrightarrow (v\ (nnth\ s\ i)) = (v\ (nfirst\ s)))$
proof (*simp add*: *stable-d-def gets-defs current-val-d-def*)
have *1*: $\bigwedge i.\ i < nlength\ s \longrightarrow v\ (nfirst\ \ (nsubn\ s\ i\ (Suc\ i))) = v\ (nnth\ s\ i)$
by (*simp add*: *nsubn-def1 ntaken-ndropn-nfirst*)
have *2*: $(\forall\, i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nfirst\ (nsubn\ s\ i\ (Suc\ i)))) =$
$(\forall\, i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i))$
by (*simp add*: *1*)
have *3*: $(\forall\, i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)) =$
$(\forall\ \ i.\ i \le nlength\ s \longrightarrow (v\ (nnth\ s\ i)) = (v\ (nfirst\ s)))$
using *stable-defs-help*[*of s v*] **by** *blast*
show $(\forall\, i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nfirst\ (nsubn\ s\ i\ (Suc\ i)))) =$
$(\forall\, i.\ enat\ i \le nlength\ s \longrightarrow v\ (nnth\ s\ i) = v\ (nfirst\ s))$
by (*simp add*: *2 3*)
qed

lemma *padded-defs* :
$(s \models padded\ v) =$
$(\ ((\forall\, i.\ \ i < nlength\ s \longrightarrow (v\ (nnth\ s\ i)) = (v\ (nfirst\ s)))) \lor nlength\ s = (enat\ 0))$
proof (*cases s*)
case (*NNil x1*)
then show *?thesis*
by (*auto simp add*: *padded-d-def stable-defs chop-d-def skip-defs empty-defs ntaken-nfirst ntaken-nnth zero-enat-def*)
next
case (*NCons x21 x22*)
then show *?thesis*
by (*auto simp add*: *padded-d-def stable-defs chop-d-def skip-defs empty-defs ntaken-nfirst*

ntaken-nnth zero-enat-def)
(*metis One-nat-def enat.simps*(*3*) *enat-add-sub-same enat-ord-simps*(*1*) *iless-Suc-eq le-iff-add less-imp-le one-enat-def plus-1-eSuc*(*2*),
meson iless-Suc-eq less-imp-le,
metis One-nat-def enat.simps(*3*) *enat-add-sub-same enat-ord-simps*(*1*) *ile-eSuc nfinite-nlength-enat one-enat-def plus-1-eSuc*(*2*),
metis One-nat-def antisym-conv2 co.enat.sel(*2*) *diff-le-self enat-add-sub-same enat-ord-code*(*4*) *enat-ord-simps*(*1*) *epred-enat iless-Suc-eq one-enat-def plus-1-eSuc*(*2*))
qed

lemma *padded-temporal-assign-defs* :
$(s \models v <\sim e) =$
$((s \models \mathit{padded}\ v) \wedge$
$(\mathit{if\ nfinite}\ s\ \mathit{then}\ (v\ (\mathit{nlast}\ s\ )) = e\ s\ \mathit{else\ True})$
$)$
by (*auto simp add*: *padded-temp-assign-d-def padded-defs temporal-assign-defs*)

lemma *chop-nfuse-1* :
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = \mathit{nfuse}\ \sigma 1\ \sigma 2 \wedge \mathit{nfinite}\ \sigma 1 \wedge$
$(\sigma 1\ \models f) \wedge (\sigma 2\ \models g) \wedge$
$(\mathit{nlast}\ \sigma 1 = \mathit{nfirst}\ \sigma 2)) =$
$((\exists\ i.\ 0 \le i \wedge i \le \mathit{nlength}\ \sigma \wedge (\ \mathit{ntaken}\ i\ \sigma\ \models f) \wedge (\mathit{ndropn}\ i\ \sigma\ \models g)))$
by *auto*
(*metis enat-le-plus-same*(*1*) *nfuse-nlength ndropn-nfuse nfinite-conv-nlength-enat ntaken-nfuse the-enat.simps*,
metis nfuse-ntaken-ndropn ndropn-nfirst nfinite-ntaken ntaken-nlast)

lemma *chop-nfuse-2* :
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = \mathit{nfuse}\ \sigma 1\ \sigma 2 \wedge \mathit{nfinite}\ \sigma 1 \wedge$
$(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge$
$(\mathit{nlast}\ \sigma 1 = \mathit{nfirst}\ \sigma 2)) =$
$(\exists\ i.\ i \le \mathit{nlength}\ \sigma \wedge (\mathit{ntaken}\ i\ \sigma) \in X \wedge (\mathit{ndropn}\ i\ \sigma) \in Y)$
by *auto*
(*metis enat-le-plus-same*(*1*) *ndropn-nfuse nfinite-nlength-enat nfuse-nlength ntaken-nfuse the-enat.simps*,
metis ndropn-nfirst nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast)

lemma *chop-nfuse*:
$(\sigma\ \models f;g) = ($
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = \mathit{nfuse}\ \sigma 1\ \sigma 2 \wedge \mathit{nfinite}\ \sigma 1 \wedge$
$(\sigma 1\ \models f) \wedge (\ \sigma 2\ \models g)\ \wedge (\mathit{nlast}\ \sigma 1 = \mathit{nfirst}\ \sigma 2))$
$\vee (\neg\ \mathit{nfinite}\ \sigma \wedge (\sigma \models f))$
$)$

by (*simp add*: *chop-defs chop-nfuse-1*)

lemmas *itl-defs* = *skip-defs chop-defs yields-defs infinite-defs finite-defs schop-defs syields-defs sometimes-defs always-defs di-defs df-defs bi-defs bf-defs da-defs ba-defs sda-defs sba-defs next-defs wnext-defs prev-defs wprev-defs more-defs fmore-defs empty-defs init-defs initalt-defs fin-defs finalt-defs sfin-defs halt-defs initonly-defs ifthenelse-defs* *currentval-defs nextval-defs finval-defs penultval-defs next-assign-defs prev-assign-defs always-eqv-defs temporal-assign-defs gets-defs stable-defs padded-defs padded-temporal-assign-defs*

4.5 Soundness Axioms

4.5.1 ChopAssoc

lemma *ChopAssocSemHelpa*:
assumes $((\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
shows $((\exists n.\ enat\ n \le nlength\ \sigma \wedge (\exists na{\le}n.\ enat\ na \le nlength\ \sigma \wedge$
$f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma))) \wedge h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists n.\ enat\ n \le nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
proof −
have *1*: $((\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
using *assms* **by** *auto*
have *2*: $\neg\ nfinite\ \sigma \wedge f\ \sigma \Longrightarrow$
$((\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$(\exists na{\le}n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists n.\ enat\ n \le nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
by *simp*
have *3*: $(\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \Longrightarrow$
$((\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$(\exists na{\le}n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists n.\ enat\ n \le nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
proof −
assume *4*: $(\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)))$
obtain n **where** *5*: $enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$

$\neg$ *nfinite* (*ndropn n* σ) $\wedge$ *g* (*ndropn n* σ))
using *4* **by** *auto*
have *6*: *enat n* $\leq$ *nlength* σ $\wedge$ *f* (*ntaken n* σ)
using *5* **by** *auto*
have *7*: $\neg$ *nfinite* (*ndropn n* σ) $\wedge$ *g* (*ndropn n* σ) $\Longrightarrow$
(($\exists$ *n*. *enat n* $\leq$ *nlength* σ $\wedge$
($\exists$ *na*$\leq$*n*. *enat na* $\leq$ *nlength* σ $\wedge$ *f* (*ntaken* (*min na n*) σ) $\wedge$ *g* (*ndropn na* (*ntaken n* σ)))
$\wedge$ *h* (*ndropn n* σ)) $\vee$
$\neg$ *nfinite* σ $\wedge$ (($\exists$ *n*. *enat n* $\leq$ *nlength* σ $\wedge$ *f* (*ntaken n* σ) $\wedge$ *g* (*ndropn n* σ)) $\vee$ $\neg$ *nfinite* σ $\wedge$ *f* σ))
using *6 nfinite-ndropn-a* **by** *blast*
have *8*: ($\exists$ *na*. *enat na* $\leq$ *nlength* σ $-$ *enat n* $\wedge$ *g* (*ntaken na* (*ndropn n* σ)) $\wedge$ *h* (*ndropn na* (*ndropn n* σ))) $\Longrightarrow$
(($\exists$ *n*. *enat n* $\leq$ *nlength* σ $\wedge$
($\exists$ *na*$\leq$*n*. *enat na* $\leq$ *nlength* σ $\wedge$ *f* (*ntaken* (*min na n*) σ) $\wedge$ *g* (*ndropn na* (*ntaken n* σ)))
$\wedge$ *h* (*ndropn n* σ)) $\vee$
$\neg$ *nfinite* σ $\wedge$ (($\exists$ *n*. *enat n* $\leq$ *nlength* σ $\wedge$ *f* (*ntaken n* σ) $\wedge$ *g* (*ndropn n* σ)) $\vee$ $\neg$ *nfinite* σ $\wedge$ *f* σ))
proof $-$
assume *9*: ($\exists$ *na*. *enat na* $\leq$ *nlength* σ $-$ *enat n* $\wedge$ *g* (*ntaken na* (*ndropn n* σ)) $\wedge$ *h* (*ndropn na* (*ndropn n* σ)))
show (($\exists$ *n*. *enat n* $\leq$ *nlength* σ $\wedge$
($\exists$ *na*$\leq$*n*. *enat na* $\leq$ *nlength* σ $\wedge$ *f* (*ntaken* (*min na n*) σ) $\wedge$ *g* (*ndropn na* (*ntaken n* σ)))
$\wedge$ *h* (*ndropn n* σ)) $\vee$
$\neg$ *nfinite* σ $\wedge$ (($\exists$ *n*. *enat n* $\leq$ *nlength* σ $\wedge$ *f* (*ntaken n* σ) $\wedge$ *g* (*ndropn n* σ)) $\vee$ $\neg$ *nfinite* σ $\wedge$ *f* σ))
proof $-$
obtain *na* **where** *10*: *enat na* $\leq$ *nlength* σ $-$ *enat n* $\wedge$ *g* (*ntaken na* (*ndropn n* σ)) $\wedge$
h (*ndropn na* (*ndropn n* σ))
using *9* **by** *auto*
have *11*: *h* (*ndropn* (*na*+*n*) σ)
by (*metis 10 add.commute ndropn-ndropn*)
have *12*: *na*+*n* $\leq$ *nlength* σ
by (*metis 10 6 Groups.add-ac*(*2*) *dual-order.strict-implies-order enat.simps*(*3*)
enat-add-sub-same enat-less-enat-plusI2 le-iff-add not-le-imp-less
order.not-eq-order-implies-strict plus-enat-simps(*1*))
have *13*: *g* (*ndropn n* (*ntaken* (*n*+*na*) σ))
by (*metis 10 12 add.commute ntaken-ndropn-swap plus-enat-simps*(*1*))
have *14*: *f* (*ntaken* (*min n* (*n*+*na*)) σ)
using *6* **by** *linarith*
show *?thesis*
by (*metis 10 12 13 14 6 add.commute le-add1 ndropn-ndropn*)
qed
qed
show *?thesis*
using *5 7 8* **by** *blast*
qed
show *?thesis*
using *3 assms* **by** *blast*
qed

lemma *ChopAssocSemHelpb*:
assumes (($\exists$ *n*. *enat n* $\leq$ *nlength* σ $\wedge$

$(\exists\, na \le n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists\, n.\ enat\ n \le nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
shows $((\exists\, n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists\, na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
proof −
have *1*: $((\exists\, n.\ enat\ n \le nlength\ \sigma \wedge$
$(\exists\, na \le n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists\, n.\ enat\ n \le nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
using *assms* **by** *auto*
have *2*: $\neg\ nfinite\ \sigma \wedge ((\exists\, n.\ enat\ n \le nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma)$
$\Longrightarrow$
$((\exists\, n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists\, na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
using *nfinite-ndropn* **by** *blast*
have *3*: $(\exists\, n.\ enat\ n \le nlength\ \sigma \wedge$
$(\exists\, na \le n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)) \Longrightarrow$
$((\exists\, n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists\, na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
proof −
assume *4*: $(\exists\, n.\ enat\ n \le nlength\ \sigma \wedge$
$(\exists\, na \le n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma))$
show $((\exists\, n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists\, na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
proof −
obtain n **where** *5*: $enat\ n \le nlength\ \sigma \wedge$
$(\exists\, na \le n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)$
using *4* **by** *auto*
have *6*: $(\exists\, na \le n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
using *5* **by** *auto*

obtain *na* **where** *7*: $na \le n \wedge enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma))$
using *6* **by** *auto*
have *8*: $na \le nlength\ \sigma$
by (*simp add*: *7*)
have *9*: $n - na \le nlength\ \sigma - na$
by (*metis 5 enat-minus-mono1 idiff-enat-enat*)
have *10*: $f\ (ntaken\ na\ \sigma)$
using *7* **by** *linarith*
have *11*: $g\ (ntaken\ (n - na)\ (ndropn\ na\ \sigma))$
by (*simp add*: *5 7 ntaken-ndropn-swap*)
have *12*: $h\ (ndropn\ ((n - na) + na)\ \sigma)$
by (*simp add*: *5 7*)
have *13*: $h\ (ndropn\ (n - na)\ (ndropn\ na\ \sigma))$
by (*metis 12 add.commute ndropn-ndropn*)
show *?thesis*
using *10 11 13 7 9* **by** *auto*
qed
qed
show *?thesis*
using *2 3 assms* **by** *blast*
qed

lemma *ChopAssocSemHelp*:
$((\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma) =$
$((\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$(\exists na \le n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)) \vee$
$\neg\ nfinite\ \sigma \wedge ((\exists n.\ enat\ n \le nlength\ \sigma \wedge f\ (ntaken\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma)) \vee \neg\ nfinite\ \sigma \wedge f\ \sigma))$
using *ChopAssocSemHelpa*[*of* σ *f g h*] *ChopAssocSemHelpb*[*of* σ *f g h*] **by** *blast*

lemma *ChopAssocSemHelp1*:
$((\ \sigma) \models\ f\ ;\ (g\ ;\ h)) = ((\ \sigma) \models (f;g);h)$
proof –
have $(\sigma \models\ f\ ;\ (g\ ;\ h)) = ((\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$((\exists na.\ enat\ na \le nlength\ \sigma - enat\ n \wedge g\ (ntaken\ na\ (ndropn\ n\ \sigma)) \wedge h\ (ndropn\ na\ (ndropn\ n\ \sigma)))$
$\vee$
$\neg\ nfinite\ (ndropn\ n\ \sigma) \wedge g\ (ndropn\ n\ \sigma))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma)$
by (*simp add*: *chop-defs*)
also have ... =
$((\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$(\exists na \le n.\ enat\ na \le nlength\ \sigma \wedge f\ (ntaken\ (min\ na\ n)\ \sigma) \wedge g\ (ndropn\ na\ (ntaken\ n\ \sigma)))$
$\wedge\ h\ (ndropn\ n\ \sigma)) \vee$

$\neg$ *nfinite* $\sigma \wedge ((\exists n.$ *enat* $n \leq$ *nlength* $\sigma \wedge f$ (*ntaken* n σ) $\wedge$ g (*ndropn* n σ)) $\vee$ $\neg$ *nfinite* $\sigma \wedge f$ σ))
using *ChopAssocSemHelp*[*of* σ *f g h*] **by** *blast*
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 *ChopAssocSemHelp1*[*of f g h* σ] **by** *auto*

4.5.2 OrChopImp

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

4.5.3 ChopOrImp

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

4.5.4 EmptyChop

lemma *EmptyChopSem*:
($\sigma \models$ *empty* ; $f = f$)
by (*simp add*: *chop-defs empty-defs min-def*)
(*metis enat-0-iff*(*1*) *ndropn-0 nlength-eq-enat-nfiniteD zero-le*)

4.5.5 ChopEmpty

lemma *ChopEmptySem*:
($\sigma \models f$;*empty* $= f$)
by (*simp add*: *chop-defs empty-defs min-def*)
(*metis cancel-comm-monoid-add-class.diff-cancel enat-diff-cancel-left idiff-enat-enat nfinite-nlength-enat ntaken-all order-refl zero-enat-def*)

4.5.6 StateImpBi

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

4.5.7 NextImpNotNextNot

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

4.5.8 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

4.5.9 BoxInduct

lemma *box-induct-help-1* :
$\bigwedge j.\ \forall n.\ enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow$
$\quad nlength\ \sigma - enat\ n = (enat\ 0) \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \Longrightarrow$
$\quad f\ \sigma \Longrightarrow$
$\quad enat\ j \leq nlength\ \sigma \Longrightarrow$
$\quad f\ (ndropn\ j\ \sigma)$
proof −
  fix *j*
  show $\forall n.\ enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow$
$\quad nlength\ \sigma - enat\ n = (enat\ 0) \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \Longrightarrow$
$\quad f\ \sigma \Longrightarrow$
$\quad enat\ j \leq nlength\ \sigma \Longrightarrow$
$\quad f\ (ndropn\ j\ \sigma)$
   proof (*induct j arbitrary*: σ)
   case *0*
   then show *?case* **by** *simp*
   next
   case (*Suc j*)
   then show *?case* **by** (*simp add*: *ndropn-ndropn*)
    (*metis Suc-ile-eq add.right-neutral enat.simps*(*3*) *enat-add-sub-same le-cases*
    le-iff-add not-less zero-enat-def)
   qed
qed

lemma *BoxInductSem*:
  $(\sigma \models \Box\ (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box\ f)$
proof
 (*auto simp add*: *always-defs wnext-defs*)
 show $\bigwedge n.\ \forall n.\ enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow$
$\quad nlength\ \sigma - enat\ n = enat\ 0 \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \Longrightarrow$
$\quad f\ \sigma \Longrightarrow$
$\quad enat\ n \leq nlength\ \sigma \Longrightarrow$
$\quad f\ (ndropn\ n\ \sigma)$
 using *box-induct-help-1* **by** *blast*
qed

4.6 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*

4.7 Temporal Quantifiers

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

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

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

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

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

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

end

5 Finite and Infinite ITL: Axioms and Rules

theory *ITL*
imports
Semantics
begin

The Finite and Infinite ITL axiom and proof rules are introduced (taken from [9]). The soundness of the rules and axioms are checked using the lemmas of Semantics.thy.

5.1 Rules

lemma *MP* :
assumes ⊢ *f* ⟶ *g*
⊢ *f*
shows ⊢ *g*
using *assms* **by** *fastforce*

lemma *BoxGen* :

assumes $\vdash f$
shows $\vdash \Box f$
using *assms*
by (*auto simp add*: *itl-defs Valid-def*)

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

5.2 Axioms

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

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

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

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

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

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

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

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

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

5.3 Additional Lemmas

The following is again from [4, 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*

5.4 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*:
$((\ w) \models (\exists\exists\ x.\ F\ x)) =$
$(\exists\ x\ (val :: 'a\ nellist).\ (\ (\ val = (nmap\ x\ w) \wedge\ ((\ w) \models F\ x))))$

by (*simp add*: *exist-state-d-def*)

lemma *AAxDef*:
⊢ (∀∀ *x*. *F x*) = (¬(∃∃ *x*. ¬(*F x*)))
by (*simp add*: *Valid-def forall-state-d-def exist-state-d-def*)

lemma *ExEqvRule*:
assumes ⋀ *x*. ⊢ (*f x*) = (*g x*)
shows ⊢ (∃ *x*. *f x*) = (∃ *x*. *g x*)
using *assms* **by** *fastforce*

5.5 Lemmas about *current-val*

lemma *current-const*: ⊢ \$(#*c*) = #*c*
by (*simp add*: *current-val-d-def intI*)

lemma *current-fun1*: ⊢ \$(*f*<*x*>) = *f* <\$*x*>
by (*simp add*: *current-val-d-def intI*)

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

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

lemma *current-forall*: ⊢ \$(∀ *x*. *P x*) = (∀ *x*. \$(*P x*))
by (*auto simp*: *current-val-d-def intI*)

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

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

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*]

5.6 Lemmas about *next-val*

lemma *next-const*: ⊢ *more* ⟶ (#*c*)\$ = #*c*
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun1*: ⊢ *more* ⟶ *f*<*x*>\$ = *f*<*x*\$>
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun2*: ⊢ *more* ⟶ *f*<*x*,*y*>\$ = *f* <*x*\$,*y*\$>
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun3*: ⊢ *more* ⟶ *f*<*x*,*y*,*z*>\$ = *f* <*x*\$,*y*\$,*z*\$>
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

lemma *next-forall*: ⊢ *more* ⟶ (∀ *x*. *P x*)\$ = (∀ *x*. (*P x*)\$)
by (*auto simp*: *next-val-d-def intI*)

lemma *next-exists*: ⊢ *more* ⟶ (∃ *x*. *P x*)\$ = (∃ *x*. (*P x*)\$)
by (*auto simp*: *next-val-d-def intI*)

lemma *next-exists1*: ⊢ *more* ⟶ (∃! *x*. *P x*)\$ = (∃! *x*. (*P x*)\$)
by (*auto simp*: *next-val-d-def more-defs zero-enat-def intI*)

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*]

5.7 Lemmas about *fin-val*

lemma *fin-const*: ⊢ *finite* ⟶ !(#*c*) = #*c*
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-fun1*: ⊢ *finite* ⟶!(*f*<*x*>) = *f* <!*x*>
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-fun2*: ⊢ *finite* ⟶ !(*f*<*x*,*y*>) = *f* <!*x*, !*y*>
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-fun3*: ⊢ *finite* ⟶ !(*f*<*x*,*y*,*z*>) = *f* <!*x*,!*y*,!*z*>
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-forall*: ⊢ *finite* ⟶ !(∀ *x*. *P x*) = (∀ *x*. !(*P x*))
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-exists*: ⊢ *finite* ⟶ !(∃ *x*. *P x*) = (∃ *x*. !(*P x*))
by (*auto simp*: *fin-val-d-def finite-defs intI*)

lemma *fin-exists1*: ⊢ *finite* ⟶ !(∃! *x*. *P x*) = (∃! *x*. !(*P x*))
by (*auto simp*: *fin-val-d-def finite-defs intI*)

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*]

5.8 Lemmas about *penult-val*

lemma *penult-const*: ⊢ *more* ∧ *finite* ⟶ (#*c*)! = #*c*
by (*auto simp*: *penult-val-d-def more-defs finite-defs zero-enat-def intI*)

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

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

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

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

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

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

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*]

5.9 Basic temporal variables properties

lemma *empty-imp-fin-eqv-curr*:
$\vdash$ *empty* $\longrightarrow$ $!v = \$v$
by (*simp add*: *Valid-def itl-defs nlength-eq-enat-nfiniteD*)
(*metis nlast-NNil nlength-eq-enat-nfiniteD nnth-nlast ntaken-0 ntaken-nlast the-enat-0 zero-enat-def*)

lemma *skip-imp-fin-eqv-next*:
$\vdash$ *skip* $\longrightarrow$ $!v = v\$$
by (*simp add*: *Valid-def itl-defs*)
(*metis One-nat-def le-numeral-extra*(*4*) *ndropn-0 nlength-eq-enat-nfiniteD*
ntaken-all ntaken-ndropn-nlast one-enat-def plus-1-eq-Suc)

lemma *skip-imp-penult-eqv-curr*:
$\vdash$ *skip* $\longrightarrow$ $v! = \$v$
by (*simp add*: *Valid-def itl-defs current-val-d-def nlength-eq-enat-nfiniteD*)
(*metis ndropn-0 ndropn-nfirst*)

end

6 Finite and Infinite ITL theorems using Weak Chop

theory *Theorems*
imports
ITL
begin

We give the proofs of a list of Finite and Infinite ITL theorems. These proofs and theorems are from [8] but adapted for infinite and finite intervals.

6.1 Propositional reasoning

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

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

6.2 State formulas

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

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

lemma *Finprop* :
$\vdash ((\#\mathit{True};(f \wedge \mathit{empty})) \wedge (\#\mathit{True};(g \wedge \mathit{empty}))) = (\#\mathit{True};((f \wedge g) \wedge \mathit{empty}))$
$\vdash ((\#\mathit{True};(f \wedge \mathit{empty})) \vee (\#\mathit{True};(g \wedge \mathit{empty}))) = (\#\mathit{True};((f \vee g) \wedge \mathit{empty}))$
$\vdash (\#\mathit{True};((\#\mathit{True}) \wedge \mathit{empty}))$
$\vdash \mathit{finite} \longrightarrow (\neg\ (\#\mathit{True};(f \wedge \mathit{empty}))) = (\#\mathit{True};(\neg f \wedge \mathit{empty}))$

⊢ (¬ (#True;(f ∧ empty))) = ((#True;(¬f ∧ empty)) ∧ finite)
using *nfinite-nlength-enat*
by (*auto simp*: *finalt-defs finite-defs zero-enat-def*,
auto simp add: *itl-defs nfinite-nlength-enat zero-enat-def*, *force*)

6.3 finite and inf properties

lemma *EmptyImpFinite*:
⊢ *empty* ⟶ *finite*
using *nlength-eq-enat-nfiniteD* **by** (*auto simp add*: *itl-defs zero-enat-def*)

lemma *SkipChopFiniteImpFinite*:
⊢ *skip*;*finite* ⟶ *finite*
using *nfinite-ndropn nlength-eq-enat-nfiniteD*
by (*simp add*: *Valid-def itl-defs*,*force*)

lemma *FiniteChopSkipImpFinite*:
⊢ *finite*;*skip* ⟶ *finite*
using *nlength-eq-enat-nfiniteD*
by (*simp add*: *Valid-def itl-defs*, *force*)

lemma *FiniteChopSkipImpMore*:
⊢ *finite*;*skip* ⟶ *more*
using *nlength-eq-enat-nfiniteD one-enat-def*
by (*simp add*: *Valid-def itl-defs*, *force*)

lemma *FiniteAndMoreImpFiniteChopSkip*:
⊢ *finite* ∧ *more* ⟶ *finite*;*skip*
by (*auto simp add*: *Valid-def itl-defs zero-enat-def*)
(*metis Suc-ile-eq diff-diff-cancel diff-le-self enat-ord-simps*(*1*) *idiff-enat-enat nfinite-nlength-enat*)

lemma *FiniteChopSkipEqvFiniteAndMore*:
⊢ *finite*;*skip* = (*finite* ∧ *more*)
by (*simp add*: *FiniteAndMoreImpFiniteChopSkip FiniteChopSkipImpFinite FiniteChopSkipImpMore Prop12 int-iffI*)

lemma *FiniteChopSkipEqvSkipChopFinite*:
⊢ *finite*;*skip* = *skip*;*finite*
by (*auto simp add*: *Valid-def itl-defs*)
(*metis enat.distinct*(*1*) *enat-add-sub-same enat-le-plus-same*(*2*) *le-iff-add* ,
metis eSuc-enat enat.simps(*3*) *enat-add-sub-same idiff-enat-0-right iless-Suc-eq le-zero-eq less-eqE min.orderE nlength-eq-enat-nfiniteD not-le one-eSuc plus-1-eSuc*(*2*),
metis add.commute enat.simps(*3*) *enat-add-sub-same idiff-enat-enat le-iff-add nfinite-nlength-enat*)

lemma *FiniteAndEmptyEqvEmpty*:
⊢ (*finite* ∧ *empty*) = *empty*
by (*auto simp add*: *Valid-def itl-defs nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *FiniteChopFiniteEqvFinite*:
$\vdash$ *finite*;*finite* = *finite*
by (*auto simp add*: *Valid-def itl-defs*) (*metis zero-enat-def zero-le*)

lemma *InfChopInfEqvInf*:
$\vdash$ *inf*;*inf* = *inf*
by (*simp add*: *Valid-def itl-defs*)

lemma *InfChopFiniteEqvInf*:
$\vdash$ *inf*;*finite* = *inf*
by (*simp add*: *Valid-def itl-defs*)

lemma *FiniteChopInfEqvInf*:
$\vdash$ *finite*;*inf* = *inf*
by (*auto simp add*: *Valid-def itl-defs*) (*metis zero-enat-def zero-le*)

lemma *InfEqvNotFinite*:
$\vdash$ *inf* = ($\neg$ *finite*)
by (*simp add*: *Valid-def itl-defs*)

lemma *FiniteEqvNotInf*:
$\vdash$ *finite* = ($\neg$ *inf*)
by (*simp add*: *Valid-def itl-defs*)

lemma *ChopTrueAndFiniteEqvAndFiniteChopFinite*:
$\vdash$ ((*f*;#*True*) $\wedge$ *finite*) = (*f* $\wedge$ *finite*);*finite*
by (*auto simp add*: *Valid-def itl-defs*)

lemma *TrueChopAndFiniteEqvAndFiniteChopFinite*:
$\vdash$ ((#*True*;*f*) $\wedge$ *finite*) = *finite*;(*f* $\wedge$ *finite*)
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteChopMoreEqvMore*:
$\vdash$ *finite*;*more* = *more*
by (*auto simp add*: *Valid-def itl-defs*)
(*metis idiff-0-right zero-enat-def zero-le*)

lemma *ChopAndFiniteDist*:
$\vdash$ ((*f*;*g*) $\wedge$ *finite*) = (*f* $\wedge$ *finite*);(*g* $\wedge$ *finite*)
by (*auto simp add*: *Valid-def itl-defs*)

lemma *FiniteOrInfinite*:
$\vdash$ *finite* $\vee$ *inf*
by (*simp add*: *Valid-def itl-defs*)

lemma *FiniteImpAnd*:
assumes $\vdash$ *finite* $\longrightarrow$ *f* = *g*
shows $\vdash$ (*f* $\wedge$ *finite*) = (*g* $\wedge$ *finite*)
using *assms* **by** (*auto simp add*: *Valid-def itl-defs*)

lemma *FmoreEqvSkipChopFinite*:
$\vdash$ *fmore* = *skip*;*finite*
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite fmore-d-def inteq-reflection lift-and-com*)

lemma *FiniteImp*:
$\vdash (f \wedge \mathit{finite} \longrightarrow g) = (f \wedge \mathit{finite} \longrightarrow g \wedge \mathit{finite})$
by (*simp add*: *itl-defs Valid-def*)

lemma *ChopAndInf*:
$\vdash ((f;g) \wedge \mathit{inf}) = (f;(g \wedge \mathit{inf}))$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *ChopAndInfB*:
$\vdash ((f;g) \wedge \mathit{inf}) = ((f \wedge \mathit{inf}) \vee (f \wedge \mathit{finite});(g \wedge \mathit{inf})\ )$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *MoreAndInfEqvInf*:
$\vdash (\mathit{more} \wedge \mathit{inf}) = \mathit{inf}$
by (*metis ChopAndInf EmptyImpFinite FiniteChopMoreEqvMore InfEqvNotFinite Prop11 Prop12 empty-d-def finite-d-def int-simps*(*32*) *inteq-reflection*)

lemma *AndInfChopAndInfEqvAndInf*:
$\vdash (f \wedge \mathit{inf});(f \wedge \mathit{inf}) = (f \wedge \mathit{inf})$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *AndInfChopEqvAndInf*:
$\vdash (f \wedge \mathit{inf});g = (f \wedge \mathit{inf})$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *AndMoreAndInfEqvAndInf*:
$\vdash ((f \wedge \mathit{more}) \wedge \mathit{inf}) = (f \wedge \mathit{inf})$
by (*auto simp add*: *Valid-def itl-defs nlength-eq-enat-nfiniteD zero-enat-def*)
(*metis gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *AndMoreAndFiniteEqvAndFmore*:
$\vdash ((f \wedge \mathit{more}) \wedge \mathit{finite}) = (f \wedge \mathit{fmore})$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *NotFmoreAndEmpty*:
$\vdash \neg\ (\mathit{empty} \wedge \mathit{fmore})$
by (*auto simp add*: *fmore-d-def empty-d-def*)

lemma *NotFmoreAndInf*:
$\vdash \neg\ ((f \wedge \mathit{inf}) \wedge \mathit{fmore})$
by (*auto simp add*: *fmore-d-def finite-d-def infinite-d-def*)

lemma *FmoreChopAnd*:
$\vdash (((f \wedge \mathit{more});g) \wedge \mathit{fmore}) = ((f \wedge \mathit{fmore});(g \wedge \mathit{finite}))$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *NotEmptyAndInf*:
$\vdash \neg(empty \wedge inf)$
by (*auto simp add*: *Valid-def itl-defs nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *OrFiniteInf*:
$\vdash\ f = ((\ f \wedge finite) \vee (\ f \wedge inf))$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *AndInfEqvChopFalse*:
$\vdash (f \wedge inf) = f;\#False$
by (*auto simp add*: *Valid-def itl-defs*)

6.4 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 *InfRightChopEqvChop*:
assumes $\vdash$ *inf* $\longrightarrow g = g1$
shows $\vdash$ *inf* $\longrightarrow (f;g) = (f;g1)$
using *assms*
by (*auto simp add*: *Valid-def itl-defs*)

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 *FiniteChopImpDiamond*:
$\vdash$ (*f* $\wedge$ *finite*);*g* $\longrightarrow$ $\diamond$ *g*
proof –
have *1*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *finite* **by** *auto*
hence *2*: $\vdash$ (*f* $\wedge$ *finite*);*g* $\longrightarrow$ *finite*;*g* **by** (*rule LeftChopImpChop*)
from *2* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

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

lemma *BoxElim*:
$\vdash$ $\Box$ *f* $\longrightarrow$ *f*
proof –

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

lemma *NextDiamondImpDiamond*:
$\vdash \bigcirc (\diamond f) \longrightarrow \diamond f$
proof –
have *1*: $\vdash$ *skip*;(*finite*;*f*) = ((*skip*;*finite*);*f*) **by** (*rule ChopAssoc*)
hence *2*: $\vdash$ (*skip*;*finite*);*f* = *skip*;(*finite*;*f*) **by** *auto*
hence *3*: $\vdash$ (*skip*;*finite*);*f* = $\bigcirc(\diamond f)$ **by** (*simp add*: *next-d-def sometimes-d-def*)
have *4*: $\vdash$ (*skip*;*finite*);*f* $\longrightarrow \diamond f$
by (*simp add*: *SkipChopFiniteImpFinite LeftChopImpChop sometimes-d-def*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

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

lemma *BoxImpBoxRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \Box f \longrightarrow \Box g$
proof –
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*
hence *3*: $\vdash \Box(\neg g \longrightarrow \neg f)$ **by** (*rule BoxGen*)
have *4*: $\vdash \Box(\neg g \longrightarrow \neg f) \longrightarrow$ (*finite*;$(\neg g)) \longrightarrow$ (*finite*;$(\neg f))$ **by** (*rule BoxChopImpChop*)
have *5*: $\vdash$ (*finite*;$(\neg g)) \longrightarrow$ (*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*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \mathit{skip};f = \mathit{skip};g$ **by** (*rule RightChopEqvChop*)
from *1* **show** *?thesis* **by** (*metis 2 next-d-def*)
qed

lemma *NextAndNextImpNextRule*:
assumes $\vdash (f \land g) \longrightarrow h$
shows $\vdash (\bigcirc f \land \bigcirc g) \longrightarrow \bigcirc h$
using *assms*
by (*simp add*: *Valid-def itl-defs*)

lemma *NextAndNextEqvNextRule*:
assumes $\vdash (f \land g) = h$
shows $\vdash (\bigcirc f \land \bigcirc g) = \bigcirc h$
using *assms*
by (*simp add*: *NextAndNextImpNextRule NextImpNext Prop11 Prop12*)

lemma *WeakNextEqvWeakNext*:
assumes $\vdash f = g$
shows $\vdash \mathit{wnext}\ f = \mathit{wnext}\ g$
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*: *itl-defs Valid-def*)

lemma *BoxAndBoxEqvBoxRule*:
 assumes $\vdash (f \wedge g) = h$
 shows $\vdash (\Box f \wedge \Box g) = \Box h$
using *assms BoxAndBoxImpBoxRule BoxImpBoxRule* **by** (*metis int-iffD1 int-iffD2 int-iffI Prop12*)

lemma *ImpBoxRule*:
 assumes $\vdash f \longrightarrow g$
 shows $\vdash \Box f \longrightarrow \Box g$
using *assms* **by** (*simp add*: *BoxImpBoxRule*)

lemma *WnextEqvEmptyOrNext*:
$\vdash$ *wnext* $f = (empty \vee \bigcirc f)$
by (*auto simp*: *Valid-def itl-defs zero-enat-def*)

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*: $\vdash f \longrightarrow (empty \vee \bigcirc f)$
  unfolding *empty-d-def* **by** *fastforce*
 hence *3*: $\vdash f \longrightarrow$ *wnext* f
  by (*metis WnextEqvEmptyOrNext inteq-reflection*)
 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 $\vdash$ *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*)
unfolding *more-d-def wnext-d-def*
by (*metis NextImpNext NextImpNotNextNot Prop05 Prop10 Prop12 int-iffD1 int-simps*(*29*) *lift-imp-trans*)
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*: $\vdash$ *finite* = ($\neg$($\Box$ *more*))
by (*metis DiamondEmptyEqvFinite InfEqvNotFinite always-d-def empty-d-def finite-d-def int-eq*)
from *7 9 11 12* **show** *?thesis*
by *fastforce*
qed

lemma *NotEmptyAndNext*:
$\vdash \neg$(*empty* $\wedge \bigcirc f$)
by (*auto simp*: *Valid-def itl-defs zero-enat-def*)

lemma *BoxEqvAndWnextBox*:

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

lemma *BoxEqvAndEmptyOrNextBox*:
$\vdash \Box f = (f \wedge (\mathit{empty}\ \ \vee\ \ \bigcirc(\Box\ f)))$
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 (*simp add*: *BoxEqvBoxBox int-iffD1*)

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*: $\vdash \Box\ (\neg\ g) = ((\neg\ g) \wedge wnext(\Box\ (\neg\ g)))$
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*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow more \wedge wnext\ f \wedge wnext(\Box\ (\neg\ g))$
using *zero-enat-def* **by** (*auto simp*: *Valid-def itl-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*: *Valid-def itl-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*: $\vdash finite = (\neg(\Box\ more))$
by (*metis DiamondEmptyEqvFinite InfEqvNotFinite always-d-def empty-d-def finite-d-def int-eq*)
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*: *itl-defs Valid-def*)
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
using *assms* **by** (*auto simp add*: *itl-defs Valid-def*)

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

6.5 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*: ⊢ (¬ (*init w*)) ⟶ ¬ (*di* (¬ ¬ (*init w*))) **by** (*simp add*: *bi-d-def*)
have *3*: ⊢ (¬ (*init w*) ⟶¬ (*di* (¬ ¬ (*init w*)))) ⟶
(*di* (¬ ¬ (*init w*)) ⟶ (*init w*)) **by** *auto*
have *4*: ⊢ *di* (¬ ¬ (*init w*)) ⟶ (*init w*) **using** *2 3 MP* **by** *blast*
have *5*: ⊢ (*init w*) ⟶ ¬ ¬ (*init w*) **by** *auto*
hence *6*: ⊢ *di* (*init w*) ⟶ *di* (¬ ¬ (*init w*)) **by** (*rule DiImpDi*)
have *7*: ⊢ *di* (*init w*) ⟶ (*init w*) **using** *6 4* **using** *lift-imp-trans* **by** *metis*
have *8*: ⊢ (*init w*) ⟶ *di* (*init w*) **by** (*rule ImpDi*)
from *7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *StateChop*:
⊢ (*init w*); *f* ⟶ (*init w*)
by (*metis ChopAssoc Prop12 RightChopImpChop TrueW init-d-def int-simps*(*13*) *int-simps*(*17*) *inteq-reflection*)

lemma *StateChopExportA*:
⊢ ((*init w*) ∧ *f*); *g* ⟶ (*init w*)
using *DiState AndChopA StateChop* **by** *fastforce*

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

lemma *StateAndChopImpChopRule*:
assumes ⊢ (*init w*) ∧ *f* ⟶ *f1*
shows ⊢ (*init w*) ∧ (*f*; *g*) ⟶ (*f1*; *g*)
proof –
have *1*: ⊢ (*init w*) ∧ *f* ⟶ *f1* **using** *assms* **by** *auto*
hence *2*: ⊢ ((*init w*) ∧ *f*); *g* ⟶ *f1*; *g* **by** (*rule LeftChopImpChop*)
have *3*: ⊢ ((*init w*) ∧ *f*); *g* = ((*init w*) ∧ (*f*; *g*)) **by** (*rule StateAndChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpChopEqvChop* :
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 StateAndChopImpChopRule*)
have *4*: ⊢ (*init w*) ∧ *f1* ⟶ *f* **using** *1* **by** *auto*
hence *5*: ⊢ (*init w*) ∧ (*f1*; *g*) ⟶ (*f*; *g*) **by** (*rule StateAndChopImpChopRule*)
from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvStateAndChop*:
assumes ⊢ *f* = (*init w*) ∧ *f1*
shows ⊢ (*f*; *g*) = ((*init w*) ∧ (*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*: $\vdash f = (((init\ w) \wedge f1) \vee \ (\ (init\ (\neg\ w)) \wedge f2))$
by (*metis Initprop*(*2*) *ifthenelse-d-def inteq-reflection*)
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 (*metis Initprop*(*2*) *ifthenelse-d-def inteq-reflection*)
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*: $\vdash f;\ g \longrightarrow (\ empty\ \lor\ f1);\ g$ **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*: $\vdash f;\ g \longrightarrow (\ empty\ \lor\ \bigcirc f1);\ g$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash (\ empty\ \lor\ \bigcirc f1);\ g = (g \lor\ \bigcirc(f1;\ g))$ **by** (*rule EmptyOrNextChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

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

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

lemma *StateAndEmptyImpBoxState*:
$\vdash (init\ w) \land empty \longrightarrow \Box\ (init\ w)$
using *BoxEqvAndEmptyOrNextBox* **by** *fastforce*

lemma *BoxEqvAndBox*:

$\vdash \Box\, f = (f \wedge \Box\, f)$
using *BoxElim* **by** *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 *BoxStateChopBoxAndInfImpBox*:
$\vdash \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w) \wedge \mathit{inf} \longrightarrow \Box\ (\mathit{init}\ w)$
by (*auto simp add*: *Valid-def itl-defs nfirst-eq-nnth-zero*)
(*metis enat-ile le-add-diff-inverse linorder-le-cases min-def ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD ntaken-nnth*)

lemma *BoxStateChopBoxEqvBox*:
$\vdash\ \ \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w) = \Box\ (\mathit{init}\ w)$
proof −
have *1*: $\vdash (\Box\ (\mathit{init}\ w)) = ((\mathit{init}\ w) \wedge (\ \mathit{empty}\ \vee\ \bigcirc(\Box\ (\mathit{init}\ w))))$
  by (*rule BoxEqvAndEmptyOrNextBox*)
hence *2*: $\vdash (\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)) =$
  $((\mathit{init}\ w) \wedge ((\ \mathit{empty}\ \vee\ \bigcirc(\Box\ (\mathit{init}\ w)));\ \Box\ (\mathit{init}\ w)))$
  by (*metis StateAndChop inteq-reflection*)
have *3*: $\vdash ((\ \mathit{empty}\ \vee\ \bigcirc(\Box\ (\mathit{init}\ w)));\ \Box\ (\mathit{init}\ w)) =$
  $(\Box\ (\mathit{init}\ w) \vee\ \bigcirc(\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)))$
  by (*rule EmptyOrNextChopEqv*)
have *4*: $\vdash (\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)) =$
  $((\mathit{init}\ w) \wedge (\Box\ (\mathit{init}\ w) \vee\ \bigcirc(\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w))))$
  using *2 3* **by** *fastforce*
have *5*: $\vdash \neg\ (\Box\ (\mathit{init}\ w)) \longrightarrow \neg\ \ (\mathit{init}\ w) \vee\ \neg(\ \bigcirc(\Box\ (\mathit{init}\ w)))$
  by (*rule NotBoxImpNotOrNotNextBox*)
have *6*: $\vdash (\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)) \wedge \neg(\ \Box\ (\mathit{init}\ w)) \longrightarrow$
  $\bigcirc(\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)) \wedge \neg(\ \bigcirc(\Box\ (\mathit{init}\ w)))$
  using *4 5* **by** *fastforce*
hence *7*: $\vdash \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w) \wedge \mathit{finite} \longrightarrow \Box\ (\mathit{init}\ w)$
  by (*rule NextContra*)
have *8*: $\vdash \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w) \wedge \mathit{inf} \longrightarrow \Box\ (\mathit{init}\ w)$
  by (*rule BoxStateChopBoxAndInfImpBox*)
have *9*: $\vdash \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w) \wedge (\mathit{finite} \vee \mathit{inf}) \longrightarrow \Box\ (\mathit{init}\ w)$
  using *7 8* **by** *fastforce*
hence *10*: $\vdash\ \ \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w) \longrightarrow \Box\ (\mathit{init}\ w)$
  using *FiniteOrInfinite* **by** *fastforce*
have *11*: $\vdash \Box\ (\mathit{init}\ w) = ((\mathit{init}\ w) \wedge \Box\ (\mathit{init}\ w))$
  by (*rule BoxEqvAndBox*)
have *12*: $\vdash \mathit{empty}\ ;\ \Box\ (\mathit{init}\ w) = \Box\ (\mathit{init}\ w)$

by (*rule EmptyChop*)
have *13*: $\vdash$ ((*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*)

6.6 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*:
$\vdash$ *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*:
$\vdash \ \ \square\ (bi\ \ f) = bi(\ \square\ f)$
proof −
have *1*: $\vdash \ \ \ \ ba\ \ f = \square\ (bi\ \ f)$ **by** (*rule BaEqvBtBi*)
have *2*: $\vdash \ \ \ \ ba\ \ f = bi(\ \square\ f)$ **by** (*rule BaEqvBiBt*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateEqvBaBoxState*:
$\vdash \ \ \square\ (init\ w) = \ ba\ (\square\ (init\ w))$
proof −
have *1*: $\vdash (init\ w) = bi\ \ (init\ w)$ **by** (*rule StateEqvBi*)
hence *2*: $\vdash \square\ (init\ w) = \square\ (bi\ \ (init\ w))$ **by** (*rule BoxEqvBox*)
have *3*: $\vdash \square\ (bi\ \ (init\ w)) = bi(\ \square\ (init\ w))$ **by** (*rule BtBiEqvBiBt*)
have *4*: $\vdash \square\ (init\ w) = \square(\square\ (init\ w))$ **by** (*rule BoxEqvBoxBox*)
hence *5*: $\vdash bi(\ \square\ (init\ w)) = bi\ (\square(\square\ (init\ w)))$ **by** (*rule BiEqvBi*)
have *6*: $\vdash \ \ ba(\ \square\ (init\ w)) = bi(\ \square(\square\ (init\ w)))$ **by** (*rule BaEqvBiBt*)
from *2 3 5 6* **show** *?thesis* **by** *fastforce*
qed

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

lemma *BaImpBt*:
$\vdash \ \ \ \ ba\ \ f \longrightarrow \square\ f$
proof −
have *1*: $\vdash \ \ ba\ \ f = bi(\ \square\ f)$ **by** (*rule BaEqvBiBt*)
have *2*: $\vdash bi(\ \square\ f) \longrightarrow \square\ f$ **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*:
$\vdash \ \ \square\ h \wedge f;\ g \longrightarrow f;\ (h \wedge\ g)$
proof −
have *1*: $\vdash h \longrightarrow g \longrightarrow (h\wedge\ g)$ **by** *auto*
hence *2*: $\vdash \square\ h \longrightarrow \square(g \longrightarrow (h\wedge\ g))$ **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*: ⊢ (□ (*init w*) ∧ *f*); (□ (*init w*) ∧ *g*) ⟶ *f*; (□ (*init w*) ∧ *g*)
  by (*rule AndChopB*)
have *15*: ⊢ *f*; (□ (*init w*) ∧ *g*) ⟶ *f*; *g*
  by (*rule ChopAndB*)
have *16*: ⊢ (□ (*init w*) ∧ *f*); (□ (*init w*) ∧ *g*) ⟶ □ (*init w*) ∧ (*f*; *g*)
  using *11 12 13 14 15* **by** *fastforce*
from *3 16* **show** *?thesis* **by** *fastforce*
qed

lemma *DiEqvNotBiNot*:
⊢ *di f* = (¬(*bi* (¬ *f*)))
proof −
have *1*: ⊢ *bi* (¬ *f*) = (¬ (*di* (¬ ¬ *f*))) **by** (*simp add*: *bi-d-def*)
hence *2*: ⊢ *di* (¬ ¬ *f*) = (¬(*bi* (¬ *f*))) **by** *auto*
have *3*: ⊢ *f* = (¬ ¬ *f*) **by** *auto*
hence *4*: ⊢ *di f* = *di* (¬ ¬ *f*) **by** (*rule DiEqvDi*)
from *2 4* **show** *?thesis* **by** *auto*
qed

lemma *ChopAndBoxImport*:
⊢ *f*; *g* ∧ □ *h* ⟶ *f*; (*g* ∧ *h*)
proof −
have *1*: ⊢ □ *h* ∧ *f*; *g* ⟶ *f*; (*h* ∧ *g*) **by** (*rule BoxAndChopImport*)
have *2*: ⊢ *f*; (*h* ∧ *g*) = *f*; (*g* ∧ *h*) **by** (*rule ChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

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

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

lemma *ChopEqvChop*:
assumes ⊢ *f* = *f1*
⊢ *g* = *g1*

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

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

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

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

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

lemma *StateAndEmptyChop*:
$\vdash\ \ ((init\ w) \wedge\ \ empty\ );\ f = ((init\ w) \wedge f)$

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

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

lemma *NextAndEqvNextAndNext*:
$\vdash \bigcirc\ (f \wedge g) = (\bigcirc\ f \wedge \bigcirc\ g)$
by (*metis NextAndNextEqvNextRule int-eq lift-and-com*)

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*: $\vdash$ *di* (*init* ($\neg$ *w*)) = (*init* ($\neg$ *w*)) **by** (*rule DiState*)
have *6*: $\vdash$ *di* ($\neg$ *f*) $\longrightarrow$ $\neg$ (*init w*) **using** *4 5* **using** *Initprop*(*2*) **by** *fastforce*
hence *7*: $\vdash$ (*init w*) $\longrightarrow$ $\neg$ (*di* ($\neg$ *f*)) **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *ChopAndNotChopImp*:
$\vdash$ *f*; *g* $\wedge$ $\neg$ (*f*; *g1*) $\longrightarrow$ *f*; (*g* $\wedge$ $\neg$ *g1*)
proof −
have *1*: $\vdash$ *g* $\longrightarrow$ (*g*$\wedge$ $\neg$ *g1*) $\vee$ *g1* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ *f*; ((*g*$\wedge$ $\neg$ *g1*) $\vee$ *g1*) **by** (*rule RightChopImpChop*)
have *3*: $\vdash$ *f*; ((*g*$\wedge$ $\neg$ *g1*) $\vee$ *g1*) $\longrightarrow$ (*f*; (*g*$\wedge$ $\neg$ *g1*)) $\vee$ (*f*; *g1*) **by** (*rule ChopOrImp*)
have *4*: $\vdash$ *f*; *g* $\longrightarrow$ *f*; (*g*$\wedge$ $\neg$ *g1*) $\vee$ *f*; *g1* **using** *2 3 MP* **by** *fastforce*
from *4* **show** *?thesis* **by** *auto*
qed

lemma *ChopAndYieldsImp*:
$\vdash$ *f*; *g* $\wedge$ *f yields g1* $\longrightarrow$ *f*; (*g* $\wedge$ *g1*)
proof −
have *1*: $\vdash$ *g* $\longrightarrow$ (*g*$\wedge$ *g1*) $\vee$ $\neg$ *g1* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ *f*; ((*g*$\wedge$ *g1*) $\vee$ $\neg$ *g1*) **by** (*rule RightChopImpChop*)
have *3*: $\vdash$ *f*; ((*g*$\wedge$ *g1*) $\vee$ $\neg$ *g1*) $\longrightarrow$ (*f*; (*g*$\wedge$ *g1*)) $\vee$ (*f*; ($\neg$ *g1*)) **by** (*rule ChopOrImp*)
have *4*: $\vdash$ *f*; *g* $\longrightarrow$ *f*; (*g*$\wedge$ *g1*) $\vee$ *f*; ($\neg$ *g1*) **using** *2 3 MP* **by** *fastforce*
hence *5*: $\vdash$ *f*; *g* $\wedge$ $\neg$ (*f*; ($\neg$ *g1*)) $\longrightarrow$ *f*; (*g* $\wedge$ *g1*) **by** *auto*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ChopAndYieldsMP*:
$\vdash$ *f*; *g* $\wedge$ *f yields* (*g*$\longrightarrow$ *g1*) $\longrightarrow$ *f*; *g1*
proof −
have *1*: $\vdash$ *f*; *g* $\wedge$ *f yields* (*g*$\longrightarrow$ *g1*) $\longrightarrow$ *f*; (*g* $\wedge$ (*g* $\longrightarrow$ *g1*)) **by** (*rule ChopAndYieldsImp*)
have *2*: $\vdash$ *g* $\wedge$ (*g* $\longrightarrow$ *g1*) $\longrightarrow$ *g1* **by** *auto*
hence *3*: $\vdash$ *f*; (*g* $\wedge$ (*g* $\longrightarrow$ *g1*)) $\longrightarrow$ *f*; *g1* **by** (*rule RightChopImpChop*)
from *1 3* **show** *?thesis* **by** *fastforce*

qed

lemma *OrYieldsImp*:
 $\vdash$ (*f* $\vee$ *f1*) *yields* *g* = ((*f* *yields* *g*) $\wedge$ (*f1* *yields* *g*))
proof –
 have *1*: $\vdash$ ((*f*$\vee$ *f1*); ($\neg$ *g*)) = ((*f*; ($\neg$ *g*)) $\vee$ (*f1*; ($\neg$ *g*))) **by** (*rule OrChopEqv*)
 hence *2*: $\vdash$ ($\neg$ ((*f*$\vee$ *f1*); ($\neg$ *g*))) = ($\neg$ (*f*; ($\neg$ *g*)) $\wedge$ $\neg$(*f1*; ($\neg$ *g*))) **by** *auto*
 from *2* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

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

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

6.7 Properties of Fin

lemma *FinEqvTrueChopAndEmpty*:
 $\vdash$ *fin f* = *#True*;(*f* $\wedge$ *empty*)
proof –
 have *1*: $\vdash$ *fin f* = $\Box$(*empty* $\longrightarrow$ *f*)
  by (*simp add*: *fin-d-def*)
 have *2*: $\vdash$ $\Box$(*empty* $\longrightarrow$ *f*) = ($\neg$($\Diamond$($\neg$(*empty* $\longrightarrow$ *f*))))
  by (*simp add*: *always-d-def*)
 have *3*: $\vdash$ ($\neg$(*empty* $\longrightarrow$ *f*)) = ($\neg$ *f* $\wedge$ *empty*)
  by *auto*
 hence *4*: $\vdash$ $\Diamond$($\neg$(*empty* $\longrightarrow$ *f*)) = $\Diamond$($\neg$ *f* $\wedge$ *empty*)
  using *DiamondEqvDiamond* **by** *blast*
 hence *5*: $\vdash$ $\neg$($\Diamond$($\neg$(*empty* $\longrightarrow$ *f*))) = ($\neg$($\Diamond$($\neg$ *f* $\wedge$ *empty*)))
  by *auto*
 have *51*: $\vdash$ *finite*;(($\neg$ *f* $\wedge$ *empty*) $\wedge$ *finite*) $\longrightarrow$ *finite* ; ($\neg$ *f* $\wedge$ *empty*)
  by (*simp add*: *ChopAndA*)
 have *52*: $\vdash$ (*#True*;($\neg$ *f* $\wedge$ *empty*) $\wedge$ *finite*) $\longrightarrow$ *finite* ; ($\neg$ *f* $\wedge$ *empty*)
  by (*metis 51 TrueChopAndFiniteEqvAndFiniteChopFinite int-eq*)
 have *53*: $\vdash$ $\neg$ (*#True*;(*f* $\wedge$ *empty*)) $\longrightarrow$ *finite* ; ($\neg$ *f* $\wedge$ *empty*)

by (*metis 52 Finprop*(*5*) *int-eq*)
have *54*: $\vdash \neg$ *finite*;($\neg$ *f* $\wedge$ *empty*) $\longrightarrow$ *#True*;(*f* $\wedge$ *empty*)
using *53* **by** *auto*
have *6*: $\vdash$ ($\neg$($\diamond$($\neg$ *f* $\wedge$ *empty*))) $\longrightarrow$ *#True*;(*f* $\wedge$ *empty*)
unfolding *sometimes-d-def* **using** *54* **by** *auto*
have *61*: $\vdash \neg$ *f* $\wedge$ *empty* $\longrightarrow$ *finite*
by (*metis ChopAndB DiamondEmptyEqvFinite NowImpDiamond inteq-reflection lift-imp-trans sometimes-d-def*)
have *62*: $\vdash$($\neg$ *#True*;(*f* $\wedge$ *empty*)) = *finite*;($\neg$ *f* $\wedge$ *empty*)
using *61*
by (*metis* (*no-types*) *Finprop*(*5*) *Prop10 TrueChopAndFiniteEqvAndFiniteChopFinite inteq-reflection*)
have *7*: $\vdash$ *#True*;(*f* $\wedge$ *empty*) $\longrightarrow$ ($\neg$($\diamond$($\neg$ *f* $\wedge$ *empty*)))
unfolding *sometimes-d-def* **using** *TrueChopAndFiniteEqvAndFiniteChopFinite*[*of LIFT*(*f* $\wedge$ *empty*)]
using *62* **by** *auto*
have *8*: $\vdash$ ($\neg$($\diamond$($\neg$ *f* $\wedge$ *empty*))) = *#True*;(*f* $\wedge$ *empty*)
by (*simp add*: *6 7 int-iffI*)
from *1 2 5 8* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondFin*:
$\vdash \diamond$(*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*:
$\vdash$ (*f* $\wedge$ *finite*);(*g* $\wedge$ *fin w*) $\longrightarrow$ *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*: *itl-defs min-absorb1*)

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*: *itl-defs Valid-def*)

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*: ⊢ $(f \wedge \mathit{finite}); (g \wedge (\mathit{init}\ w)) \longrightarrow (f \wedge \mathit{finite}); ((\mathit{init}\ w) \wedge g)$
  by (*rule RightChopImpChop*)
have *10*: ⊢ $((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w)); g \longrightarrow (f \wedge \mathit{finite}); ((\mathit{init}\ w) \wedge g)$
  using *6 7 9* **by** *fastforce*
have *11*: ⊢ $(f \wedge \mathit{finite}) \longrightarrow ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ w)) \vee ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))$
  using *ImpAndFinStateOrFinNotState* **by** *blast*
hence *12*: ⊢ $(f \wedge \mathit{finite}); ((\mathit{init}\ w) \wedge g) \longrightarrow$
$(((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w)) \vee$
$((\mathit{finite} \wedge f) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))\ )); ((\mathit{init}\ w) \wedge g)$
  using *LeftChopImpChop*
  by (*metis inteq-reflection lift-and-com*)
have *13*: ⊢ $(((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w)) \vee ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w)))); ((\mathit{init}\ w) \wedge g)$
$=$
$(((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w));$
$((\mathit{init}\ w) \wedge g) \vee\ ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))); ((\mathit{init}\ w) \wedge g))$
  by (*rule OrChopEqv*)
have *14*: ⊢ $(\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ (\neg\ w))); ((\mathit{init}\ w) \wedge g) \longrightarrow$
$\Diamond(\ (\mathit{init}\ (\neg\ w)) \wedge ((\mathit{init}\ w) \wedge g))$
  using *FinChopEqvDiamond*
  by (*metis AndFinEqvChopAndEmpty ChopEmpty FiniteChopImpDiamond LeftChopImpChop int-eq*)
have *141*: ⊢ $\neg(\ \Diamond(\ (\mathit{init}\ (\neg\ w)) \wedge ((\mathit{init}\ w) \wedge g))) \longrightarrow$
$\neg\ (\ (\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ (\neg\ w))); ((\mathit{init}\ w) \wedge g))$
  using *14* **by** *fastforce*
have *142*: ⊢ $((\mathit{init}\ (\neg\ w)) \wedge ((\mathit{init}\ w) \wedge g)) = \#\mathit{False}$
  using *Initprop*(*2*) **by** *fastforce*
have *15*: ⊢ $\neg(\ \Diamond(\ (\mathit{init}\ (\neg\ w)) \wedge ((\mathit{init}\ w) \wedge g)))$
  by (*metis 142 NotDiamondAndNot int-simps*(*21*) *inteq-reflection*)
have *151*: ⊢ $\neg\ (\ (\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ (\neg\ w))); ((\mathit{init}\ w) \wedge g))$
  using *15 141* **by** *fastforce*
have *1511*: ⊢ $(\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))); ((\mathit{init}\ w) \wedge g) \longrightarrow \#\mathit{False}$
  using *151* **by** (*metis Initprop*(*2*) *int-simps*(*14*) *inteq-reflection*)
have *152*: ⊢ $((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w));$
$((\mathit{init}\ w) \wedge g) \vee\ (\ (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ (\mathit{init}\ w))); ((\mathit{init}\ w) \wedge g) \longrightarrow$
$((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w)); ((\mathit{init}\ w) \wedge g)$
  using *1511* **by** *fastforce*
have *16*: ⊢ $(f \wedge \mathit{finite}); ((\mathit{init}\ w) \wedge g) \longrightarrow ((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w)); ((\mathit{init}\ w) \wedge g)$
  using *12 13 152*
  proof –
   have ⊢ $(f \wedge \mathit{finite});(\mathit{init}\ w \wedge g) \longrightarrow$
$((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ w) \vee (f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\neg\ \mathit{init}\ w));(\mathit{init}\ w \wedge g)$
    by (*metis 12 inteq-reflection lift-and-com*)
   then show *?thesis*
    using *13 152* **by** *fastforce*
  qed
have *17*: ⊢ $((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w)); ((\mathit{init}\ w) \wedge g) \longrightarrow ((f \wedge \mathit{finite}) \wedge \mathit{fin}\ (\mathit{init}\ w)); g$
  by (*rule ChopAndB*)
have *18*: ⊢ $(f \wedge \mathit{finite}); ((\mathit{init}\ w) \wedge g) \longrightarrow ((f \wedge \mathit{finite}) \wedge\ \mathit{fin}\ (\mathit{init}\ w)); g$
  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*: ⊢ *bi* (*f* ∧ *finite* ⟶ *fin* (*init w1*)) = (*f* ∧ *finite*)*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*:
⊢ *f yields g* ⟶ (*f* ∧ *f1*) *yields g*
proof −
have *1*: ⊢ *f* ∧ *f1* ⟶ *f* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftYieldsImpYields*)
qed

lemma *AndYieldsB*:
⊢ *f1 yields g* ⟶ (*f* ∧ *f1*) *yields g*
proof −
have *1*: ⊢ *f* ∧ *f1* ⟶ *f1* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftYieldsImpYields*)
qed

lemma *RightYieldsImpYields*:
assumes ⊢ *g*⟶ *g1*
shows ⊢ (*f yields g*) ⟶ (*f yields g1*)
proof −
have *1*: ⊢ *g* ⟶ *g1* **using** *assms* **by** *auto*
hence *2*: ⊢ ¬ *g1* ⟶ ¬ *g* **by** *auto*
hence *3*: ⊢ *f*; (¬ *g1*) ⟶ *f*; (¬ *g*) **by** (*rule RightChopImpChop*)
hence *4*: ⊢ ¬ (*f*; (¬ *g*)) ⟶ ¬ (*f*; (¬ *g1*)) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *RightYieldsEqvYields*:
assumes ⊢ *g* = *g1*
shows ⊢ (*f yields g*) = (*f yields g1*)
proof −
have *1*: ⊢ *g* = *g1* **using** *assms* **by** *auto*
hence *2*: ⊢ (¬ *g*) = (¬ *g1*) **by** *auto*
hence *3*: ⊢ *f*; (¬ *g*) = *f*; (¬ *g1*) **by** (*rule RightChopEqvChop*)
hence *4*: ⊢ (¬ (*f*; (¬ *g*))) = (¬ (*f*; (¬ *g1*))) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)

qed

lemma *BoxImpYields*:
$\vdash \quad \Box\, g \longrightarrow (f \wedge \mathit{finite})\ \mathit{yields}\ \ g$
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*:
$\vdash \quad \Box\, f = \mathit{finite}\ \mathit{yields}\ \ f$
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 $\vdash \quad g$
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*:
$\vdash \quad ((f\ \mathit{yields}\ \ g) \wedge (f\ \ \mathit{yields}\ \ g1)) = f\ \mathit{yields}\ (g \wedge g1)$
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 (*metis 1 3 int-simps*(*14*) *int-simps*(*33*) *inteq-reflection*)
from *6* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *YieldsAndYieldsImpAndYieldsAnd*:
$\vdash \quad (f\ \mathit{yields}\ \ g) \wedge (f1\ \mathit{yields}\ \ g1) \longrightarrow (f \wedge\ f1)\ \mathit{yields}\ (g \wedge\ g1)$

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** (*simp add*: *NextImpNext more-d-def*)
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 (*metis ChopAndB FiniteChopFiniteEqvFinite FmoreEqvSkipChopFinite RightChopImpChop inteq-reflection next-d-def*)
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 (*metis AndInfChopEqvAndInf MoreAndInfEqvInf Prop11 Prop12 lift-imp-trans*)
have *2*: $\vdash \Diamond$ *more* $\longrightarrow$ *more*
by (*metis FiniteChopMoreEqvMore NowImpDiamond inteq-reflection sometimes-d-def*)
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*: *OrFiniteInf*)
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*: *FmoreEqvSkipChopFinite LeftChopEqvChop next-d-def*)
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*:
$\vdash \quad di\ \ f = \ di\ (\ \ di\ \ f)$
proof –
have *1*: $\vdash \#True = \#True;\ \#True$ **by** (*rule TrueEqvTrueChopTrue*)
hence *2*: $\vdash f;\ \#True = f;\ (\#True;\ \#True)$ **by** (*rule RightChopEqvChop*)
have *3*: $\vdash f;\ (\#True;\ \#True) = (f;\ \#True);\ \#True$ **by** (*rule ChopAssoc*)
have *4*: $\vdash f;\ \#True = (f;\ \#True);\ \#True$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis di-d-def*)
qed

lemma *BiEqvBiBi*:
$\vdash \quad bi\ \ f = bi(\ bi\ \ f)$
proof –
have *1*: $\vdash \ \ di\ (\neg\ \ f) = \ \ di(\ \ di\ (\neg\ \ f))$ **by** (*rule DiEqvDiDi*)
have *2*: $\vdash \ \ di\ (\neg\ \ f) = (\neg\ (\ bi\ \ f))$ **by** (*rule DiNotEqvNotBi*)
hence *3*: $\vdash \ \ di\ (\ di\ (\neg\ \ f)) = \ \ di\ (\neg\ (\ bi\ \ f))$ **by** (*rule DiEqvDi*)
have *4*: $\vdash \ \ di\ (\neg\ \ f) = \ \ di\ (\neg(\ \ bi\ \ f))$ **using** *1 3* **by** *fastforce*
hence *5*: $\vdash (\neg\ (\ di\ (\neg\ \ f))) = (\neg\ (\ di\ (\neg\ (\ bi\ \ f))))$ **by** *fastforce*
from *5* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *DiOrEqv*:
$\vdash \quad di\ (f \vee \ \ g) = \ (di\ \ f \vee \ \ di\ \ g)$
proof –
have *1*: $\vdash (f\vee \ \ g);\ \#True = (f;\ \#True \vee \ \ g;\ \#True)$ **by** (*rule OrChopEqv*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

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

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

lemma *DiAndImpAnd*:
$\vdash \quad di\ (f \wedge g) \longrightarrow \ \ di\ \ f \wedge \ \ di\ \ g$
proof –
have *1*: $\vdash \ \ di\ (f \wedge g) \longrightarrow \ \ di\ \ f$ **by** (*rule DiAndA*)
have *2*: $\vdash \ \ di\ (f \wedge g) \longrightarrow \ \ di\ \ g$ **by** (*rule DiAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*

qed

lemma *DiSkipEqvMore*:
$\vdash$ *di skip* = *more*
proof –
have *1*: $\vdash$ *skip* ; *#True* = $\bigcirc$*#True* **by** (*rule SkipChopEqvNext*)
have *2*: $\vdash$ $\bigcirc$*#True* = *more* **by** (*auto simp*: *more-d-def*)
have *3*: $\vdash$ *skip* ; *#True* = *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiMoreEqvMore*:
$\vdash$ *di more* = *more*
proof –
have *1*: $\vdash$ *di* ($\bigcirc$ *#True*) = $\bigcirc$(*di #True*)
by (*rule DiNext*)
have *2*: $\vdash$ $\bigcirc$(*di #True*) $\longrightarrow$ *more*
by (*metis 1 ChopImpDi TrueEqvTrueChopTrue di-d-def int-eq more-d-def*)
have *3*: $\vdash$ *di*($\bigcirc$ *#True*) $\longrightarrow$ *more*
using *1 2* **by** *fastforce*
hence *4*: $\vdash$ *di more* $\longrightarrow$ *more*
by (*simp add*: *more-d-def*)
have *5*: $\vdash$ *more* $\longrightarrow$ *di more*
by (*rule ImpDi*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

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

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

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

qed

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

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

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

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

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

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

lemma *BiAndEqv*:
$\vdash$ *bi* $(f \wedge g) = $ (*bi* $f \wedge$ *bi* g)
proof –
have *1*: $\vdash$ *di* $(\neg f \vee \neg g) = $ (*di* $(\neg f) \vee$ *di* $(\neg g)$)
by (*simp add*: *DiOrEqv*)
have *2*: $\vdash$ ($\neg$ (*di* $(\neg f \vee \neg g)$)) $=$ ($\neg$ *di* $(\neg f) \wedge \neg$ *di* $(\neg g)$)
using *1* **by** *auto*
have *3*: $\vdash (f \wedge g) = (\neg (\neg f \vee \neg g))$
by *fastforce*
have *4*: $\vdash$ *bi* $(f \wedge g) = $ ($\neg$ (*di* $(\neg f \vee \neg g)$))
unfolding *bi-d-def* **using** *3* **by** (*metis int-simps*(*4*) *inteq-reflection*)
from *2 4* **show** *?thesis* **unfolding** *bi-d-def* **by** (*metis inteq-reflection*)
qed

lemma *BaAndEqv*:
$\vdash$ *ba* $(f \wedge g) = $ (*ba* $f \wedge$ *ba* g)
proof –
have *1*: $\vdash$ *ba* $(f \wedge g) = \Box$(*bi* $(f \wedge g)$)
by (*rule BaEqvBtBi*)
have *2*: $\vdash$ *bi* $(f \wedge g) = $ (*bi* $f \wedge$ *bi* g)
by (*simp add*: *BiAndEqv*)
hence *3*: $\vdash \Box$(*bi* $(f \wedge g)$) $= \Box$(*bi* $f \wedge$ *bi* g)
using *BoxEqvBox* **by** *blast*
have *4*: $\vdash \Box$(*bi* $f \wedge$ *bi* g)$=$ ($\Box$(*bi* f) $\wedge \Box$(*bi* g))
by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
have *5*: $\vdash$*ba* $f = \Box$(*bi* f)
by (*rule BaEqvBtBi*)
have *6*: $\vdash$ *ba* $g = \Box$(*bi* g)

by (*rule BaEqvBtBi*)
from *1 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBaEqvBa*:
$\vdash$ *ba* $(f = g) \longrightarrow (ba\ f = ba\ g)$
proof –
have *1*: $\vdash$ *ba* $(f \longrightarrow g) \longrightarrow ba\ f \longrightarrow ba\ g$ **by** (*rule BaImpDist*)
have *2*: $\vdash$ *ba* $(g \longrightarrow f) \longrightarrow ba\ g \longrightarrow ba\ f$ **by** (*rule BaImpDist*)
have *25*: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** *fastforce*
have *3*: $\vdash ba\ (f = g) = ba\ ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (*metis 25 BaAndEqv inteq-reflection*)
have *4*: $\vdash ba\ ((f \longrightarrow g) \wedge (g \longrightarrow f)) = (ba((f \longrightarrow g)) \wedge ba((g \longrightarrow f)))$ **by** (*rule BaAndEqv*)
have *5*: $\vdash ((ba\ f \longrightarrow ba\ g) \wedge (ba\ g \longrightarrow ba\ f)) = (ba\ f = ba\ g)$ **by** *auto*
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBa*:
assumes $\vdash f \longrightarrow g$
shows $\vdash ba\ f \longrightarrow ba\ g$
using *BaGen BaImpDist MP assms* **by** *metis*

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

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

lemma *DiamondEqvDiamondDiamond*:
$\vdash \Diamond f = \Diamond (\Diamond f)$
proof –
have *1*: $\vdash \Diamond (\Diamond f) = finite;(finite;f)$
by (*simp add*: *sometimes-d-def*)
have *2*: $\vdash finite;(finite;f) = (finite;finite);f$
by (*rule ChopAssoc*)
have *3*: $\vdash (finite;finite);f = finite;f$
by (*simp add*: *LeftChopEqvChop FiniteChopFiniteEqvFinite*)
have *4*: $\vdash finite;f = \Diamond f$
by (*simp add*: *sometimes-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DaEqvDaDa*:
$\vdash da\ f = da\ (da\ f)$
proof –
have *1*: $\vdash da\ f = \Diamond(di\ f)$
by (*rule DaEqvDtDi*)

have *2*: $\vdash$ *di f* = (*di* (*di f*))
  by (*rule DiEqvDiDi*)
 hence *3*: $\vdash$ $\diamond$ (*di f*) = $\diamond$ (*di* (*di f*))
  by (*rule DiamondEqvDiamond*)
 have *4*: $\vdash$ $\diamond$ (*di f*) = $\diamond$($\diamond$ (*di* (*di f*)))
  using *DiamondEqvDiamondDiamond DiEqvDiDi* **using** *3* **by** *fastforce*
 have *5*: $\vdash$ $\diamond$ (*di* (*di f*)) = *di* ($\diamond$ (*di f*))
  by (*rule DtDiEqvDiDt*)
 hence *6*: $\vdash$ $\diamond$($\diamond$ (*di* (*di f*))) = $\diamond$ (*di* ($\diamond$ (*di f*)))
  by (*rule DiamondEqvDiamond*)
 have *7*: $\vdash$ *da f* = $\diamond$ (*di*($\diamond$ (*di f*)))
  using *1 3 4 6* **by** *fastforce*
 have *8*: $\vdash$ *da* ($\diamond$ (*di f*)) = $\diamond$(*di* ($\diamond$ (*di f*)))
  by (*rule DaEqvDtDi*)
 have *9*: $\vdash$ *da* (*da f*) = *da* ($\diamond$ (*di f*))
  using *1* **by** (*rule DaEqvDa*)
 from *7 8 9* **show** *?thesis* **by** *fastforce*
qed

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

lemma *BaLeftChopImpChop*:
$\vdash$ *ba* (*f* $\longrightarrow$ *f1*) $\longrightarrow$ *f*; *g* $\longrightarrow$ *f1*; *g*
proof –
 have *1*: $\vdash$ *ba* (*f* $\longrightarrow$ *f1*) $\longrightarrow$ *bi* (*f* $\longrightarrow$ *f1*) **by** (*rule BaImpBi*)
 have *2*: $\vdash$ *bi* (*f* $\longrightarrow$ *f1*) $\longrightarrow$ *f*; *g* $\longrightarrow$ *f1*; *g* **by** (*rule BiChopImpChop*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

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

lemma *ChopAndBaImport*:
$\vdash$ (*f*; *f1*) $\wedge$ *ba g* $\longrightarrow$ (*f* $\wedge$ *g*); (*f1* $\wedge$ *g*)
proof –
 have *1*: $\vdash$ *ba g* $\wedge$ (*f*; *f1*) $\longrightarrow$ (*g* $\wedge$ *f*); (*g* $\wedge$ *f1*) **by** (*rule BaAndChopImport*)
 have *2*: $\vdash$ (*g* $\wedge$ *f*); (*g* $\wedge$ *f1*) = (*f* $\wedge$ *g*); (*f1* $\wedge$ *g*) **by** (*rule AndChopAndCommute*)

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

lemma *BaAndChopImportA*:
$\vdash$ *ba f* $\wedge$ *g;g1* $\longrightarrow$ (*f* $\wedge$ *g*);*g1*
by (*meson BaAndChopImport ChopAndB lift-imp-trans*)

lemma *BaAndChopImportB*:
$\vdash$ *ba f* $\wedge$ *g;g1* $\longrightarrow$ (*f* $\wedge$ *g*);(*ba f* $\wedge$ *g1*)
proof –
have *1*: $\vdash$ *ba f* = *ba* (*ba f*)
by (*simp add*: *BaEqvBaBa*)
have *2*: $\vdash$ *ba* (*ba f*) $\wedge$ *g;g1* $\longrightarrow$ *g*;(*ba f* $\wedge$ *g1*)
by (*metis AndChopB BaAndChopImport lift-imp-trans*)
have *3*: $\vdash$ *ba f* $\wedge$ *g*;(*ba f* $\wedge$ *g1*) $\longrightarrow$ (*f* $\wedge$ *g*);(*ba f* $\wedge$ *g1*)
by (*simp add*: *BaAndChopImportA*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *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*: ⊢ *di* (¬ (*ba* *f*)) ⟶ ¬ (*ba* *f*) **by** (*rule DiNotBaImpNotBa*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

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

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

lemma *FiniteRightChopEqvChop*:
assumes ⊢ *finite* ⟶ *g* = *g1*
shows ⊢ *finite* ⟶ *f*;*g* = *f*;*g1*
using *assms* **by** (*auto simp add*: *Valid-def itl-defs*)

lemma *FinImpYieldsFin*:
⊢ *fin* (*init w*) ∧ *finite* ⟶ (*f* ∧ *finite*) *yields* (*fin* (*init w*) ∧ *finite*)
proof −
have *1*: ⊢ (*f* ∧ *finite*); (*fin* (*init* (¬ *w*)) ∧ *finite*) ⟶ (*fin* (*init* (¬ *w*)) ∧ *finite*)
by (*metis* (*no-types*, *lifting*) *ChopAndB FiniteChopEqvDiamond FiniteChopFinExportA FiniteChopFiniteEqvFinite FiniteChopImpDiamond Prop12 inteq-reflection lift-and-com lift-imp-trans*)
have *2*: ⊢ *finite* ⟶ (¬ (*fin* (*init w*) ∧ *finite*)) = (*fin* (*init* (¬ *w*)) ∧ *finite*)
using *FinNotStateEqvNotFinState* **by** *fastforce*
hence *3*: ⊢ *finite* ⟶ (*f* ∧ *finite*); (¬ (*fin* (*init w*) ∧ *finite*)) =
(*f* ∧ *finite*); (*fin* (*init* (¬ *w*)) ∧ *finite*)
using *FiniteRightChopEqvChop*[*of LIFT*(¬ (*fin* (*init w*) ∧ *finite*))
LIFT(*fin* (*init* (¬ *w*)) ∧ *finite*) *LIFT*(*f* ∧ *finite*)]
by *blast*
have *4*: ⊢ (*f* ∧ *finite*); (¬ (*fin* (*init w*) ∧ *finite*)) ⟶ (¬ (*fin* (*init w*) ∧ *finite*))
using *1 2 3* **by** *fastforce*
hence *5*: ⊢ *fin* (*init w*) ∧ *finite* ⟶ ¬ ((*f* ∧ *finite*); (¬ (*fin* (*init w*) ∧ *finite*)))
by *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)

qed

lemma *ChopAndFin*:
 $\vdash$ $((f;\ g) \wedge\ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})) = (f \wedge \mathit{finite});\ (g \wedge\ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}))$
proof –
 have *1*: $\vdash$ $\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite} \longrightarrow (f \wedge \mathit{finite})\ \mathit{yields}\ (\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})$
  by (*rule FinImpYieldsFin*)
 have *10*: $\vdash ((f;\ g) \wedge\ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})) =$
   $(((f \wedge \mathit{finite});\ (g \wedge \mathit{finite})) \wedge\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})$
  using *ChopAndFiniteDist*[*of f g*] **by** *auto*
 have *2*: $\vdash (f;\ g) \wedge\ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}) \longrightarrow$
   $((f \wedge \mathit{finite});\ (g \wedge \mathit{finite})) \wedge (f \wedge \mathit{finite})\ \mathit{yields}\ \ (\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})$
  using *1 10* **by** *fastforce*
 have *3*: $\vdash ((f \wedge \mathit{finite});\ (g \wedge \mathit{finite})) \wedge (f \wedge \mathit{finite})\ \mathit{yields}\ \ (\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}) \longrightarrow$
   $(f \wedge \mathit{finite});\ ((g \wedge \mathit{finite}) \wedge\ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}))$
  using *ChopAndYieldsImp* **by** *blast*
 have *30*: $\vdash ((g \wedge \mathit{finite}) \wedge\ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})) = (g \wedge \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})$
  by *auto*
 have *4*: $\vdash (f;\ g) \wedge\ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}) \longrightarrow (f \wedge \mathit{finite});\ (g \wedge\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})$
  using *2 3 30*
  by (*metis* (*mono-tags*, *lifting*) *inteq-reflection lift-imp-trans*)
 have *11*: $\vdash (f \wedge \mathit{finite});\ (g \wedge\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}) \longrightarrow (f \wedge \mathit{finite});\ (g \wedge \mathit{finite})$
  using *ChopAndA* **by** (*metis 30 inteq-reflection*)
 have *12*: $\vdash (f \wedge \mathit{finite});\ (g \wedge\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}) \longrightarrow$
   $(f \wedge \mathit{finite});\ \ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})$
  by (*rule ChopAndB*)
 have *13*: $\vdash (f \wedge \mathit{finite});\ \ (\mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}) \longrightarrow \Diamond\ (\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite})$
  using *FiniteChopImpDiamond* **by** *blast*
 have *14*: $\vdash \Diamond(\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}) \longrightarrow\ \mathit{fin}\ \ (\mathit{init}\ w)$
  by (*metis ChopAndA DiamondFin inteq-reflection sometimes-d-def*)
 have *15*: $\vdash (f \wedge \mathit{finite});\ (g \wedge\ \mathit{fin}\ \ (\mathit{init}\ w) \wedge \mathit{finite}) \longrightarrow$
   $((f \wedge \mathit{finite});\ (g \wedge \mathit{finite})) \wedge\ \mathit{fin}\ \ (\mathit{init}\ w)$
  using *11 12 13 14* **by** *fastforce*
 from *4 15* **show** *?thesis* **by** (*metis ChopAndFiniteDist Prop12 int-iffI inteq-reflection*)
qed

lemma *ChopAndNotFin*:
 $\vdash$ $(f;\ g \wedge \neg\ (\ \mathit{fin}\ \ (\mathit{init}\ w)) \wedge \mathit{finite}) = (f \wedge \mathit{finite});\ (g \wedge \neg\ (\ \mathit{fin}\ \ (\mathit{init}\ w)) \wedge \mathit{finite})$
proof –
 have *1*: $\vdash (f;\ g \wedge\ \mathit{fin}\ \ \ (\mathit{init}\ (\neg\ w)) \wedge \mathit{finite}) =$
   $(f \wedge \mathit{finite});\ (g \wedge\ \mathit{fin}\ \ \ (\mathit{init}\ (\neg\ w)) \wedge \mathit{finite})$
  by (*rule ChopAndFin*)
 have *2*: $\vdash$ $(\mathit{fin}\ \ \ (\mathit{init}\ (\neg\ w)) \wedge \mathit{finite}) = (\ (\neg\ \ (\ \mathit{fin}\ \ (\mathit{init}\ w)\ )) \wedge \mathit{finite})$
  using *FinNotStateEqvNotFinState* **by** *fastforce*
 show *?thesis* **by** (*metis 1 2 int-eq*)
qed

lemma *FinChopChain*:
$\vdash$ $(((\mathit{init}\ w\ )\ \ \wedge \mathit{finite} \longrightarrow\ \ \mathit{fin}\ \ (\mathit{init}\ w1)));$
  $(((\mathit{init}\ w1)\ \ \wedge \mathit{finite} \longrightarrow\ \ \mathit{fin}\ \ (\mathit{init}\ w2)))$

$\wedge$ *finite*
$\longrightarrow$ (((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)))
proof –
have *1*: $\vdash$ (*init w*) $\wedge$ *finite* $\wedge$
((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)); ((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*))
$\longrightarrow$
((*init w*) $\wedge$ *finite* $\wedge$ ((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)));
(((*init w1*) $\wedge$ *finite*$\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*)
using *ChopAndFiniteDist StateAndChopImport*
by (*metis* (*no-types*, *opaque-lifting*) *inteq-reflection lift-and-com*)
have *2*: $\vdash$ (*init w*) $\wedge$ *finite* $\wedge$ ((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin*(*init w1*)) $\longrightarrow$ *fin* (*init w1*) $\wedge$ *finite*
by *auto*
have *3*: $\vdash$ ((*init w*) $\wedge$ *finite* $\wedge$ ((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)));
(((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*)
$\longrightarrow$
(*fin* (*init w1*) $\wedge$ *finite*); (((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*)
using *2 LeftChopImpChop* **by** *blast*
have *4*: $\vdash$ (*fin* (*init w1*) $\wedge$ *finite*); (((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*) =
$\Diamond$((*init w1*) $\wedge$ ((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)) $\wedge$ *finite*)
using *FinChopEqvDiamond* **by** *blast*
have *41*: $\vdash$ ((*init w1*) $\wedge$ *finite* $\wedge$ ((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*))) $\longrightarrow$ *fin* (*init w2*)
by *auto*
have *42*: $\vdash$ $\Diamond$((*init w1*) $\wedge$ *finite* $\wedge$ ((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*))) $\longrightarrow$ $\Diamond$(*fin* (*init w2*))
using *41 DiamondImpDiamond* **by** *blast*
have *5*: $\vdash$ $\Diamond$(*fin*(*init w2*)) $\longrightarrow$ *fin* (*init w2*)
using *DiamondFinImpFin* **by** *blast*
have *6*: $\vdash$ (*init w*) $\wedge$ *finite* $\wedge$ ((*init w*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*));
((*init w1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*))
$\longrightarrow$ *fin* (*init w2*)
using *1 3 4 5 42*
using *ChopAndCommute FinChopEqvDiamond* **by** *fastforce*
from *6* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopRule*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*)
$\vdash$ (*init w1*)$\wedge$ *f1* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)
shows $\vdash$ (*init w*) $\wedge$ (*f*; *f1*) $\wedge$ *finite* $\longrightarrow$ *fin* (*init w2*)
proof –
have *1*: $\vdash$ (*init w*) $\wedge$ (*f*; *f1*) $\wedge$ *finite* $\longrightarrow$ ((*init w*) $\wedge$ *f* $\wedge$ *finite*); (*f1* $\wedge$ *finite*)
using *StateAndChopImport*
by (*metis ChopAndFiniteDist inteq-reflection*)
have *2*: $\vdash$ (*init w*) $\wedge$ *f* $\wedge$ *finite* $\longrightarrow$ *fin* (*init w1*) $\wedge$ *finite*
using *assms* **by** *auto*
hence *3*: $\vdash$ ((*init w*) $\wedge$ *f* $\wedge$ *finite*); (*f1* $\wedge$ *finite*) $\longrightarrow$ (*fin* (*init w1*) $\wedge$ *finite*); (*f1* $\wedge$ *finite*)
by (*rule LeftChopImpChop*)
have *4*: $\vdash$ (*fin* (*init w1*) $\wedge$ *finite*); (*f1* $\wedge$ *finite*) = $\Diamond$((*init w1*) $\wedge$ *f1* $\wedge$ *finite*)
by (*rule FinChopEqvDiamond*)
have *5*: $\vdash$ (*init w1*) $\wedge$ *f1* $\wedge$ *finite* $\longrightarrow$ *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* **by** (*meson Prop12 lift-imp-trans*)
qed

lemma *TrueChopMoreEqvMore*:
⊢ #*True* ; *more* = *more*
by (*metis ChopMoreImpMore EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteChopMoreEqvMore*

LeftChopImpChop Prop09 int-eq-true int-iffI inteq-reflection)

lemma *FiniteChopFmoreEqvFmore*:
$\vdash$ *finite*;*fmore* = *fmore*
by (*metis TrueChopAndFiniteEqvAndFiniteChopFinite TrueChopMoreEqvMore fmore-d-def inteq-reflection*)

lemma *MoreChopLoop*:
assumes $\vdash$ *f* $\longrightarrow$ *fmore* ; *f*
shows $\vdash$ *finite* $\longrightarrow$ $\neg$ *f*
proof –
have *1*: $\vdash$ *f* $\longrightarrow$ *fmore* ; *f*
using *assms* **by** *auto*
hence *11*: $\vdash$ $\diamond$ (*f*) $\longrightarrow$ $\diamond$ (*fmore*;*f*)
using *DiamondImpDiamond* **by** *blast*
have *12*: $\vdash$ $\diamond$ (*fmore*;*f*) = *finite*;(*fmore*;*f*)
by (*simp add*: *sometimes-d-def*)
have *13*: $\vdash$ *finite*;(*fmore*;*f*) = (*finite*;*fmore*);*f*
by (*rule ChopAssoc*)
have *14*: $\vdash$ $\diamond$ (*fmore*;*f*) = *fmore*;*f*
using *FiniteChopFmoreEqvFmore 12 13* **by** (*metis int-eq*)
have *2*: $\vdash$ *fmore* ; *f* = $\bigcirc$($\diamond$ *f*)
using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: $\vdash$ $\diamond$ (*f*) $\longrightarrow$ $\bigcirc$($\diamond$ *f*)
using *11 14 2* **by** *fastforce*
hence *4*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ($\diamond$ *f*)
using *NextLoop* **by** *blast*
have *5*: $\vdash$ $\neg$ ($\diamond$ *f*) $\longrightarrow$ $\neg$ *f*
using *NowImpDiamond* **by** *fastforce*
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *MoreChopContra*:
assumes $\vdash$ *f* $\wedge$ $\neg$ *g* $\longrightarrow$ (*fmore* ; (*f* $\wedge$ $\neg$ *g*))
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *g*
proof –
have *1*: $\vdash$ *f* $\wedge$ $\neg$ *g* $\longrightarrow$ (*fmore* ; (*f* $\wedge$ $\neg$ *g*)) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *finite* $\longrightarrow$ $\neg$ (*f* $\wedge$ $\neg$ *g*) **by** (*rule MoreChopLoop*)
from *2* **show** *?thesis* **by** *auto*
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*: ⊢ *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*: ⊢ *fmore* ; *f* = ○(◇ *f*)
  using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: ⊢ ◇ (*f* ∧ *finite*) ⟶ ○(◇ *f*)
  using *11 14 2* **by** *fastforce*
have *31*: ⊢ ◇ (*f* ∧ *finite*) = ((◇ *f*) ∧ *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*: ⊢ (◇ *f*) ∧ *finite* ⟶ ○(◇ *f*)
  using *3 31* **by** *fastforce*
hence *4*: ⊢ *finite* ⟶ ¬ (◇ *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*: ⊢ ¬ (◇ *f*) ⟶ ¬ *f*
  by (*simp add*: *NowImpDiamond*)
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *MoreChopEqvFmoreOrInf*:
⊢ *more* ; *f* = ((*fmore*;*f*) ∨ *inf*)
by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv OrFiniteInf fmore-d-def int-eq*)

lemma *MoreChopLoopFiniteB*:
assumes ⊢ *f* ⟶ *more* ; *f*
shows ⊢ *finite* ⟶ ¬ *f*
proof –
have *1*: ⊢ *f* ⟶ *more* ; *f*
  using *assms* **by** *auto*
have *10*: ⊢ *f* ⟶ (*fmore*;*f*) ∨ *inf*
  using *MoreChopEqvFmoreOrInf assms* **by** *fastforce*
hence *100*: ⊢ *f* ∧ *finite* ⟶ (*fmore*;*f*)
  by (*simp add*: *Prop13 finite-d-def*)
hence *11*: ⊢ ◇ (*f* ∧ *finite*) ⟶ ◇ (*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*: ⊢ *fmore* ; *f* = ○(◇ *f*)
  using *MoreChopEqvNextDiamond* **by** *blast*
have *3*: ⊢ ◇ (*f* ∧ *finite*) ⟶ ○(◇ *f*)
  using *11 14 2* **by** *fastforce*
have *31*: ⊢ ◇ (*f* ∧ *finite*) = ((◇ *f*) ∧ *finite*)
  by (*metis* (*no-types*, *opaque-lifting*) *ChopAndA ChopAndB ChopAndNotChopImp FiniteChopFiniteEqvFi-*

nite
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*: $\vdash$ *finite* $\longrightarrow$ $\neg$ (*f* $\wedge$ $\neg$ *g*) **using** *MoreChopLoopFinite* **by** (*simp add*: *MoreChopLoopFinite*)
from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *MoreChopContraFiniteB*:
assumes $\vdash$ (*f* $\wedge$ $\neg$ *g*) $\longrightarrow$ (*more* ; (*f* $\wedge$ $\neg$ *g*))
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *g*
proof –
have *1*: $\vdash$ (*f* $\wedge$ $\neg$ *g*) $\longrightarrow$ (*more* ; (*f* $\wedge$ $\neg$ *g*)) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *finite* $\longrightarrow$ $\neg$ (*f* $\wedge$ $\neg$ *g*) **using** *MoreChopLoopFinite* **by** (*simp add*: *MoreChopLoopFiniteB*)
from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

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

lemma *ChopLoopB*:
assumes $\vdash$ *f* $\longrightarrow$ *g*;*f*
$\vdash$ *g* $\longrightarrow$ *more*
shows $\vdash$ *finite* $\longrightarrow$ $\neg$ *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** *MoreChopLoopFiniteB* **by** *auto*
qed

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

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

6.8 Properties of Halt

lemma *WnextAndMoreEqvNext*:
$\vdash$ (*wnext f* $\wedge$ *more*) $= \bigcirc f$
proof –
have *1*: $\vdash$ *wnext f* = (*empty* $\vee \bigcirc f$)
by (*simp add*: *WnextEqvEmptyOrNext*)
have *2*: $\vdash \bigcirc f \longrightarrow$ *more*
by (*metis DiIntro LeftChopImpMoreRule di-d-def more-d-def next-d-def*)
have *3*: $\vdash$ ((*empty* $\vee \bigcirc f$) $\wedge$ *more*) $= \bigcirc f$
unfolding *empty-d-def* **using** *2* **by** *auto*
show *?thesis* **by** (*metis 1 3 int-eq*)
qed

lemma *BoxStateAndEmptyEqvStateAndEmpty*:
$\vdash$ ($\Box$(*empty* = (*init w*)) $\wedge$ *empty*) = ((*init w*) $\wedge$ *empty*)
proof –
have *1*: $\vdash$ ((*empty* = (*init w*)) $\wedge$ *empty*) = ((*init w*) $\wedge$ *empty*)
by *force*
have *2*: $\vdash$ ($\Box$(*empty* = (*init w*)) $\wedge$ *empty*) $\longrightarrow$ ((*init w*) $\wedge$ *empty*)

using *BoxElim* **by** *fastforce*
have *3*: $\vdash ((init\ w) \wedge empty) \longrightarrow (\Box(empty = (init\ w)) \wedge empty)$
using *BoxEqvAndEmptyOrNextBox* **by** *fastforce*
show *?thesis*
by (*simp add*: *2 3 int-iffI*)
qed

lemma *BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*:
$\vdash \Box(empty = (init\ w)) = ((empty \wedge init\ w) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w)))))$
proof –
have *1*: $\vdash \Box(empty = (init\ w)) =$
$((\Box(empty = (init\ w)) \wedge empty) \vee (\Box(empty = (init\ w)) \wedge more))$
by (*auto simp*: *empty-d-def*)
have *2*: $\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$
using *BoxStateAndEmptyEqvStateAndEmpty* **by** *blast*
have *3*: $\vdash \Box(empty = (init\ w)) = ((empty = (init\ w)) \wedge wnext(\Box(empty = (init\ w))))$
using *BoxEqvAndWnextBox* **by** *blast*
hence *4*: $\vdash (\Box(empty = (init\ w)) \wedge more) =$
$(((empty = (init\ w)) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$
by *auto*
have *5*: $\vdash ((empty = (init\ w)) \wedge more) = (\neg(init\ w) \wedge more)$
by (*auto simp*: *empty-d-def*)
have *6*: $\vdash (wnext(\Box(empty = (init\ w))) \wedge more) = \bigcirc(\Box(empty = (init\ w)))$
using *WnextAndMoreEqvNext* **by** *metis*
have *7*: $\vdash (\Box(empty = (init\ w)) \wedge more) =$
$((\neg (init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$
using *4 5* **by** *fastforce*
have *8*: $\vdash ((\neg (init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)) =$
$((\neg (init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$
by *auto*
have *9*: $\vdash ((\neg (init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)) =$
$((\neg (init\ w)) \wedge \bigcirc(\Box(empty = (init\ w))))$
using *8 6* **by** *auto*
have *10*: $\vdash \Box(empty = (init\ w)) = (((init\ w) \wedge empty) \vee (\Box(empty = (init\ w)) \wedge more))$
using *1 2* **by** *fastforce*
show *?thesis*
using *10 7 9* **by** *fastforce*
qed

lemma *HaltStateEqvIfStateThenEmptyElseNext*:
$\vdash\ halt(\ init\ w) =\ if_i\ \ (init\ w)\ \ then\ \ \ empty\ \ \ else\ (\ \bigcirc(\ halt\ (\ init\ w)))$
by (*metis BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext halt-d-def ifthenelse-d-def inteq-reflection lift-and-com*)

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 *FiniteChopAndEmptyEqvChopAndEmpty*:
$\vdash$ ((*finite*;(*f* $\wedge$ *empty*)) $\wedge$ *g*) = ((*g* $\wedge$ *finite*);(*f*$\wedge$ *empty*))
proof –
have *1*: $\vdash$ *g* $\wedge$ *finite*;(*f* $\wedge$ *empty*) $\longrightarrow$ *fin f*
by (*metis ChopAndA DiamondFin FinAndEmpty Prop01 Prop05 inteq-reflection sometimes-d-def*)
have *2*: $\vdash$ *g* $\wedge$ *finite*;(*f* $\wedge$ *empty*) $\longrightarrow$ (*finite* $\wedge$ *g*) $\wedge$ *fin f*
using *1* **by** (*metis* (*no-types*, *lifting*) *ChopAndB ChopEmpty Prop10 Prop12 int-iffD1*
inteq-reflection)
have *3*: $\vdash$ ((*finite*;(*f* $\wedge$ *empty*)) $\wedge$ *g*) $\longrightarrow$ ((*g* $\wedge$ *finite*);(*f*$\wedge$ *empty*))

using *2* **using** *AndFinEqvChopAndEmpty* **by** *fastforce*
have *4*: $\vdash ((g \wedge \mathit{finite});(f\wedge\ \mathit{empty})) \longrightarrow ((\mathit{finite};(f\ \wedge\ \mathit{empty}))\ \wedge\ g)$
by (*metis AndChopB ChopAndB ChopEmpty Prop12 inteq-reflection*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *WprevEqvEmptyOrPrev*:
$\vdash \mathit{wprev}\ f = (\mathit{empty} \vee \mathit{prev}\ f)$
using *nlength-eq-enat-nfiniteD* **by** (*auto simp add*: *Valid-def itl-defs zero-enat-def*)

lemma *NotChopSkipEqvMoreAndNotChopSkip*:
$\vdash (\neg\ f);\mathit{skip} = (\mathit{more} \wedge \neg(f;\mathit{skip}))$
proof –
have *1*: $\vdash \mathit{wprev}\ f = (\mathit{empty} \vee \mathit{prev}\ f)$ **using** *WprevEqvEmptyOrPrev* **by** *auto*
hence *2*: $\vdash (\neg(\mathit{wprev}\ f)) = (\neg(\mathit{empty} \vee \mathit{prev}\ f))$ **by** *auto*
have *3*: $\vdash \neg(\mathit{wprev}\ f) = ((\neg\ f);\mathit{skip})$ **by** (*simp add*: *wprev-d-def prev-d-def*)
have *31*: $\vdash (\mathit{empty} \vee \mathit{prev}\ f) = (\mathit{empty} \vee (f;\mathit{skip}))$ **by** (*simp add*: *prev-d-def*)
have *32*: $\vdash (\mathit{empty} \vee (f;\mathit{skip})) = (\neg \mathit{more} \vee \neg\neg(f;\mathit{skip}))$ **by** (*simp add*: *empty-d-def*)
have *33*: $\vdash (\neg \mathit{more} \vee \neg\neg(f;\mathit{skip})) = (\neg(\mathit{more} \wedge \neg(f;\mathit{skip})))$ **by** *fastforce*
have *34*: $\vdash (\mathit{empty} \vee \mathit{prev}\ f) = (\neg(\mathit{more} \wedge \neg(f;\mathit{skip})))$ **using** *31 32 33* **by** (*metis int-eq*)
have *4*: $\vdash \neg(\mathit{empty} \vee \mathit{prev}\ f) = (\mathit{more} \wedge \neg(f;\mathit{skip}))$ **using** *34* **by** *fastforce*
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltChopImpNotHaltChopNot*:
$\vdash\ \ \mathit{halt}\ (\ \mathit{init}\ w);\ f \wedge \mathit{finite} \longrightarrow \neg\ (\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg\ \ f))$
proof –
have *1*: $\vdash\ \mathit{halt}\ \ (\mathit{init}\ w);\ f =\ \ \mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ f\ \ \mathit{else}\ (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f))$
by (*rule HaltChopEqv*)
have *2*: $\vdash \mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ f\ \ \mathit{else}\ (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)) \longrightarrow$
$(\ ((\mathit{init}\ w) \longrightarrow f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f))))$
by (*rule IfThenElseImp*)
have *3*: $\vdash\ \mathit{halt}\ \ (\mathit{init}\ w);\ (\neg f) =$
$\mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ (\neg f)\ \ \mathit{else}\ (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))$
by (*rule HaltChopEqv*)
have *4*: $\vdash \mathit{if}_i\ \ (\mathit{init}\ w)\ \ \mathit{then}\ \ (\neg f)\ \ \mathit{else}\ (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f))) \longrightarrow$
$(\ ((\mathit{init}\ w) \longrightarrow \neg f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))))$
by (*rule IfThenElseImp*)
have *5*: $\vdash \mathit{halt}\ \ (\mathit{init}\ w);\ f \wedge \mathit{halt}\ \ (\mathit{init}\ w);\ (\neg f) \longrightarrow$
$(\ ((\mathit{init}\ w) \longrightarrow f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)))) \wedge$
$(\ ((\mathit{init}\ w) \longrightarrow \neg f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))))$
using *1 2 3 4* **by** *fastforce*
have *6*: $\vdash (\ ((\mathit{init}\ w) \longrightarrow f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)))) \wedge$
$(\ ((\mathit{init}\ w) \longrightarrow \neg f) \wedge (\ \neg(\mathit{init}\ w) \longrightarrow (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f))))) \longrightarrow$
$(\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)) \wedge (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))$
by *auto*
have *7*: $\vdash \mathit{halt}\ \ (\mathit{init}\ w);\ f \wedge \mathit{halt}\ \ (\mathit{init}\ w);\ (\neg f) \longrightarrow$
$(\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)) \wedge (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))$
using *5 6 lift-imp-trans* **by** *blast*
have *8*: $\vdash ((\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ f)) \wedge (\ \bigcirc(\ \mathit{halt}\ (\ \mathit{init}\ w);\ (\neg f)))) =$

$\bigcirc$ (*halt* (*init w*); *f* ∧ *halt* (*init w*); (¬*f*))
using *NextAndEqvNextAndNext* **by** *fastforce*
have *9*: ⊢ *halt* (*init w*); *f* ∧ *halt* (*init w*); (¬*f*) ⟶
$\bigcirc$ (*halt* (*init w*); *f* ∧ *halt* (*init w*); (¬*f*))
using *7 8* **by** *fastforce*
hence *10*: ⊢ *finite* ⟶ ¬(*halt* (*init w*); *f* ∧ *halt* (*init w*); (¬*f*))
using *NextLoop* **by** *blast*
from *10* **show** *?thesis* **by** *auto*
qed

lemma *HaltChopImpHaltYields*:
⊢ *halt* (*init w*); *f* ∧ *finite* ⟶ (*halt* (*init w*)) *yields f*
proof –
have *1*: ⊢ *halt* (*init w*); *f* ∧ *finite* ⟶ ¬ (*halt* (*init w*); (¬ *f*))
by (*rule HaltChopImpNotHaltChopNot*)
from *1* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *HaltChopAnd*:
⊢ (*halt* (*init w*)); *f* ∧ (*halt* (*init w*)); *g* ∧ *finite*⟶ (*halt* (*init w*)); (*f* ∧ *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* ⟶

$\neg($ *halt* $($*init* $w); g) \vee (($ *halt* $($*init* $w) \wedge f); (f1 \wedge g))$
using *31 41* **by** *fastforce*
from *42* **show** *?thesis* **by** *auto*
qed

lemma *HaltImpBoxYields*:
$\vdash$ (*halt* (*init* w)); $f \wedge$ *finite* $\longrightarrow (\Box(\neg ($ *init* $w)))$ *yields* ((*halt* (*init* w)); f)
proof –
have *1*: $\vdash (\Box (\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); $f)) \longrightarrow$ *di* $(\Box (\neg$ (*init* w)))
by (*rule ChopImpDi*)
have *2*: $\vdash \Box (\neg$ (*init* w)) $\longrightarrow \neg$ (*init* w)
by (*rule BoxElim*)
hence *3*: $\vdash$ *di* $(\Box (\neg$ (*init* w))) $\longrightarrow$ *di* $(\neg$ (*init* w))
by (*rule DiImpDi*)
have *4*: $\vdash$ *di* (*init* $(\neg w)$) = (*init* $(\neg w)$)
by (*rule DiState*)
have *41*: $\vdash$ (*init* $(\neg w)$) = $(\neg$ (*init* w))
using *Initprop*(*2*) **by** *fastforce*
have *42*: $\vdash$ *di* $(\neg$ (*init* w)) = $(\neg$(*init* w))
using *4 41* **by** (*metis inteq-reflection*)
have *5*: $\vdash ((\Box(\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); $f))) \longrightarrow \neg ($ *init* w)
using *1 2 42* **using** *3* **by** *fastforce*
hence *51*: $\vdash$ (*halt* (*init* w); $f) \wedge ((\Box(\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); $f))) \longrightarrow$
(*halt* (*init* w); $f) \wedge \neg ($ *init* w)
by *fastforce*
have *6*: $\vdash$ *halt* (*init* w); f = if_i (*init* w) *then* f *else* $(\bigcirc($ *halt* (*init* w); f))
by (*rule HaltChopEqv*)
hence *61*: $\vdash$ (*halt* (*init* w); $f \wedge \neg ($ *init* w)) =
((if_i (*init* w) *then* f *else* $(\bigcirc($ *halt* (*init* w); f))) $\wedge \neg ($ *init* w))
using *6* **by** *auto*
have *62*: $\vdash$ (if_i (*init* w) *then* f *else* $(\bigcirc($ *halt* (*init* w); f))) $\wedge$
$\neg ($ *init* $w) \longrightarrow (\bigcirc($ *halt* (*init* w); f))
by (*auto simp*: *ifthenelse-d-def*)
have *63*: $\vdash$ *halt* (*init* w); $f \wedge \neg ($ *init* $w) \longrightarrow (\bigcirc($ *halt* (*init* w); f))
using *61 62* **by** *fastforce*
have *7*: $\vdash$ (*halt* (*init* w); $f) \wedge (\Box(\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); $f)) \longrightarrow$
$\bigcirc(($ *halt* (*init* w)); f)
using *51 63* **using** *lift-imp-trans* **by** *blast*
have *8*: $\vdash \Box (\neg$ (*init* w)) $\longrightarrow$ *empty* $\vee \bigcirc(\Box(\neg($ *init* w)))
by (*metis BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq*)
hence *9*: $\vdash ((\Box (\neg ($ *init* w))); $(\neg ($ *halt* (*init* w); $f))) \longrightarrow$
$\neg ($ *halt* (*init* w); $f) \vee \bigcirc((\Box(\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); f)))
by (*rule EmptyOrNextChopImpRule*)
hence *10*: $\vdash (($ *halt* (*init* w)); $f) \wedge (\Box (\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); $f)) \longrightarrow$
$\bigcirc((\Box(\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); f)))
by *fastforce*
have *11*: $\vdash$ (*halt* (*init* w)); $f \wedge (\Box (\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); $f)) \longrightarrow$
$\bigcirc(($ *halt* (*init* w)); $f) \wedge \bigcirc((\Box(\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); f)))
using *7 10* **by** *fastforce*
have *12*: $\vdash \bigcirc(($ *halt* (*init* w)); $f) \wedge \bigcirc((\Box(\neg$ (*init* w))); $(\neg ($ *halt* (*init* w); f)))

$\longrightarrow$ $\bigcirc$(((*halt* (*init w*)); *f*) $\wedge$ (($\Box$($\neg$ (*init w*))); ($\neg$ (*halt* (*init w*); *f*))))
using *NextAndEqvNextAndNext* **by** *fastforce*
have *13*: $\vdash$ (*halt* (*init w*)); *f* $\wedge$ ($\Box$ ($\neg$ (*init w*))); ($\neg$ (*halt* (*init w*); *f*)) $\longrightarrow$
$\bigcirc$(((*halt* (*init w*)); *f*) $\wedge$ (($\Box$($\neg$ (*init w*))); ($\neg$ (*halt* (*init w*); *f*))))
using *11 12* **by** *fastforce*
hence *14*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ((*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* $\wedge$ *finite* $\longrightarrow$ $\neg$ (($\Box$ ($\neg$ (*init w*))); ($\neg$ (*halt* (*init w*); *f*)))
by *auto*
from *15* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

6.9 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

7 Finite and Infinite ITL theorems using strong chop

theory *SChopTheorems*
imports
Theorems
begin

We give the proofs of a list of Finite and Infinite ITL theorems but now using the strong chop.

7.1 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* $\wedge$ *finite*);((*g* $\wedge$ *finite*);*h*)
by (*simp add*: *schop-d-def*)
have *2*: $\vdash$ (*f* $\wedge$ *finite*);((*g* $\wedge$ *finite*);*h*) = ((*f* $\wedge$ *finite*);(*g* $\wedge$ *finite*));*h*
using *ChopAssoc* **by** *blast*
have *3*: $\vdash$ ((*f* $\wedge$ *finite*);(*g* $\wedge$ *finite*));*h* = ($f \frown$ (*g* $\wedge$ *finite*));*h*
by (*simp add*: *schop-d-def*)
have *4*: $\vdash$ $f \frown$ (*g* $\wedge$ *finite*) = ($f \frown g$ $\wedge$ *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 *FiniteOr*:
$\vdash ((f \vee g) \wedge \mathit{finite}) = ((f \wedge \mathit{finite}) \vee (g \wedge \mathit{finite}))$
by *auto*

lemma *OrSChopImp* :
$\vdash (f \vee g) \frown h \longrightarrow f \frown h \vee g \frown h$
unfolding *schop-d-def*
by (*simp add*: *FiniteOr OrChopImpRule int-iffD1*)

lemma *SChopOrImp* :
$\vdash f \frown (g \vee h) \longrightarrow f \frown g \vee f \frown h$
unfolding *schop-d-def* **by** (*simp add*: *ChopOrImp*)

lemma *EmptySChop* :
$\vdash \mathit{empty} \frown f = f$
by (*metis EmptyChopSem FiniteAndEmptyEqvEmpty intI inteq-reflection lift-and-com schop-d-def*)

lemma *SChopEmpty* :
$\vdash \mathit{finite} \longrightarrow f \frown \mathit{empty} = f$
unfolding *schop-d-def*
proof –
have *f1*: $\vdash (f \wedge \mathit{finite});\mathit{empty} = (f \wedge \mathit{finite})$
by (*simp add*: *ChopEmpty int-eq*)
then show $\vdash \mathit{finite} \longrightarrow (f \wedge \mathit{finite});\mathit{empty} = f$
by *fastforce*
qed

lemma *StateImpBf* :
$\vdash \mathit{init}\ f \longrightarrow \mathit{bf}\ (\mathit{init}\ f)$
unfolding *bf-d-def df-d-def schop-d-def*
by (*metis* (*no-types*) *AndChopA StateImpBi bi-d-def di-d-def lift-imp-neg lift-imp-trans*)

lemma *BfBoxSChopImpSChop* :
$\vdash \mathit{bf}\ (f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f1 \frown g1$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *AndMoreSChopEqvAndFmoreChop*:
$\vdash (f \wedge \mathit{more}) \frown g = (f \wedge \mathit{fmore});g$
by (*simp add*: *LeftChopEqvChop AndMoreAndFiniteEqvAndFmore schop-d-def*)

lemma *FiniteBfGen*:
 assumes $\vdash$ *finite* $\longrightarrow$ *f*
 shows $\vdash$ *bf f*
using *assms*
by (*simp add*: *Valid-def itl-defs*)

lemma *BfGen*:
 assumes $\vdash$ *f*
 shows $\vdash$ *bf f*
using *assms*
by (*metis EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteBfGen Prop09 int-eq-true inteq-reflection*)

7.2 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*:
 $\vdash f;g = ((f \frown g) \lor (f \land inf))$
by (*metis AndInfChopEqvAndInf OrChopEqv OrFiniteInf inteq-reflection schop-d-def*)

lemma *SFinprop* :
 $\vdash ((\#True \frown (f \land empty)) \land (\#True \frown (g \land empty))) = (\#True \frown ((f \land g) \land empty))$
 $\vdash ((\#True \frown (f \land empty)) \lor (\#True \frown (g \land empty))) = (\#True \frown ((f \lor g) \land empty))$
 $\vdash\ finite \longrightarrow (\neg\ (\#True \frown (f \land empty))) = (\#True \frown (\neg f \land empty))$
 $\vdash (\neg\ (\#True \frown (f \land empty))) = ((\#True \frown (\neg f \land empty)) \lor inf)$
by (*auto simp add*: *Valid-def itl-defs zero-enat-def*)
 (*metis add.right-neutral enat.distinct*(*2*) *enat-add-sub-same less-eqE the-enat.simps zero-enat-def*,
 metis ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat nlength-NNil the-enat.simps zero-enat-def,
 metis add.right-neutral enat.distinct(*2*) *enat-add-sub-same le-iff-add the-enat.simps zero-enat-def*,
 metis ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat nlength-NNil the-enat.simps zero-enat-def,
 metis add.right-neutral enat.distinct(*2*) *enat-add-sub-same le-iff-add the-enat.simps zero-enat-def*,

metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD)

7.3 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*
unfolding *bi-d-def bf-d-def di-d-def df-d-def schop-d-def*
by (*simp add*: *AndChopA*)

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* :
$\vdash\ \Box\ (g \longrightarrow g1) \longrightarrow f\frown g \longrightarrow f\frown g1$
proof –
have *1*: $\vdash g \longrightarrow g$ **by** *auto*
hence *2*: $\vdash$ *bf* $(\ g \longrightarrow g)$ **by** (*rule BfGen*)
have *3*: $\vdash$ *bf* $(\ f \longrightarrow f) \wedge \Box(g \longrightarrow g1) \longrightarrow f\frown g \longrightarrow f\frown g1$ **by** (*rule BfBoxSChopImpSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftSChopImpSChop*:
assumes $\vdash f \longrightarrow f1$
shows $\vdash f\frown g \longrightarrow f1\frown g$
proof –
have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ *bf* $(f \longrightarrow f1)$ **by** (*rule BfGen*)
have *3*: $\vdash$ *bf* $(f \longrightarrow f1) \longrightarrow f\frown g \longrightarrow f1\frown g$ **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

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

lemma *RightSChopEqvSChop*:
assumes $\vdash g = g1$
shows $\vdash (f\frown g) = (f\frown g1)$
proof –
have *1*: $\vdash g = g1$ **using** *assms* **by** *auto*
have *2*: $(\vdash g \longrightarrow g1) \Longrightarrow (\vdash f\frown g \longrightarrow f\frown g1)$ **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*:
 $\vdash \Box\ (g = g1) \longrightarrow (f \frown g) = (f \frown g1)$
proof –
 have *0*: $\vdash (g = g1) = (\ (g \longrightarrow g1) \wedge\ (\ g1 \longrightarrow g))$
  by *fastforce*
 have *1*: $\vdash \Box\ (g = g1) = (\Box\ (g \longrightarrow g1) \wedge \Box\ (\ g1 \longrightarrow g))$
  by (*metis 0 BoxAndBoxEqvBoxRule inteq-reflection*)
 have *2*: $\vdash \Box\ (g \longrightarrow g1) \longrightarrow (f \frown g) \longrightarrow (f \frown g1)$
  by (*simp add*: *BoxSChopImpSChop*)
 have *3*: $\vdash \Box\ (g1 \longrightarrow g) \longrightarrow (f \frown g1) \longrightarrow (f \frown g)$
  by (*simp add*: *BoxSChopImpSChop*)
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FiniteRightSChopEqvSChop*:
 assumes $\vdash$ *finite* $\longrightarrow g = g1$
 shows $\vdash$ *finite* $\longrightarrow (f \frown g) = (f \frown g1)$
using *assms* **unfolding** *schop-d-def*
by (*simp add*: *FiniteRightChopEqvChop*)

lemma *SChopOrEqv*:
 $\vdash f \frown (g \vee g1) = (f \frown g \vee f \frown g1)$
proof –
 have *1*: $\vdash g \longrightarrow g \vee g1$ **by** *auto*
 hence *2*: $\vdash f \frown g \longrightarrow f \frown (g \vee g1)$ **by** (*rule RightSChopImpSChop*)
 have *3*: $\vdash g1 \longrightarrow g \vee g1$ **by** *auto*
 hence *4*: $\vdash f \frown g1 \longrightarrow f \frown (g \vee g1)$ **by** (*rule RightSChopImpSChop*)
 from *2 4* **show** *?thesis* **by** (*meson SChopOrImp Prop02 Prop11*)
qed

lemma *OrSChopEqv*:
 $\vdash (f \vee f1) \frown g = (f \frown g \vee f1 \frown g)$
proof –
 have *1*: $\vdash f \longrightarrow f \vee f1$ **by** *auto*
 hence *2*: $\vdash f \frown g \longrightarrow (f \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
 have *3*: $\vdash f1 \longrightarrow f \vee f1$ **by** *auto*
 hence *4*: $\vdash f1 \frown g \longrightarrow (f \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
 from *2 4* **show** *?thesis*
 by (*meson OrSChopImp int-iffI Prop02*)
qed

lemma *OrSChopImpRule*:
 assumes $\vdash f \longrightarrow f1 \vee f2$
 shows $\vdash f \frown g \longrightarrow (f1 \frown g) \vee (f2 \frown g)$
proof –
 have *1*: $\vdash f \longrightarrow f1 \vee f2$ **using** *assms* **by** *auto*
 hence *2*: $\vdash f \frown g \longrightarrow (f1 \vee f2) \frown g$ **by** (*rule LeftSChopImpSChop*)

have *3*: ⊢ (*f1* ∨ *f2*)⁀ *g* = (*f1*⁀*g* ∨ *f2*⁀*g*) **by** (*rule OrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

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

lemma *OrSChopEqvRule*:
assumes ⊢ *f* = (*f1* ∨ *f2*)
shows ⊢ *f*⁀*g* = ((*f1*⁀*g*) ∨ (*f2*⁀*g*))
proof –
have *1*: ⊢ *f* = (*f1* ∨ *f2*) **using** *assms* **by** *auto*
hence *2*: ⊢ *f*⁀*g* = ((*f1* ∨ *f2*)⁀*g*) **by** (*rule LeftSChopEqvSChop*)
have *3*: ⊢ (*f1* ∨ *f2*)⁀*g* = (*f1*⁀*g* ∨ *f2*⁀*g*) **by** (*rule OrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopOrImpRule*:
assumes ⊢ *g* ⟶ *g1* ∨ *g2*
shows ⊢ *f*⁀*g* ⟶ (*f*⁀*g1*) ∨ (*f*⁀*g2*)
proof –
have *1*: ⊢ *g* ⟶ *g1* ∨ *g2* **using** *assms* **by** *auto*
hence *2*: ⊢ *f*⁀*g* ⟶ *f*⁀(*g1* ∨ *g2*) **by** (*rule RightSChopImpSChop*)
have *3*: ⊢ *f*⁀(*g1* ∨ *g2*) = (*f*⁀*g1* ∨ *f*⁀*g2*) **by** (*rule SChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopImpDiamond*:
⊢ *f*⁀*g* ⟶ ◇ *g*
proof –
have *1*: ⊢ *f* ⟶ #*True* **by** *auto*
hence *2*: ⊢ *f*⁀*g* ⟶ #*True*⁀*g* **by** (*rule LeftSChopImpSChop*)
from *2* **show** *?thesis* **using** *DiamondSChopdef* **by** *fastforce*
qed

lemma *BfImpDfImpDf*:
⊢ *bf* (*f* ⟶ *g*) ⟶ *df* *f* ⟶ *df* *g*
proof –

have *1*: $\vdash$ *bf* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*f*$\frown$ #*True*) $\longrightarrow$ (*g*$\frown$ #*True*) **by** (*rule BfSChopImpSChop*)
from *1* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

lemma *DfImpDf*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ *df f* $\longrightarrow$ *df g*
proof –
have *1*: $\vdash$ *f* $\longrightarrow$ *g* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*$\frown$ #*True* $\longrightarrow$ *g*$\frown$ #*True* **by** (*rule LeftSChopImpSChop*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

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

lemma *DfEqvDf*:
assumes $\vdash$ *f* = *g*
shows $\vdash$ *df f* = *df g*
proof –
have *1*: $\vdash$ *f* = *g* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*$\frown$ #*True* = *g*$\frown$ #*True* **by** (*rule LeftSChopEqvSChop*)
from *2* **show** *?thesis* **by** (*simp add*: *df-d-def*)
qed

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

lemma *LeftSChopSChopImpSChopRule*:
assumes $\vdash$ (*f*$\frown$ *g*) $\longrightarrow$ *g*

shows $\vdash (f\frown\ g)\frown\ h \longrightarrow (g\frown\ h)$
proof –
have *1*: $\vdash (f\frown\ g) \longrightarrow g$ **using** *assms* **by** *blast*
hence *2*: $\vdash (f\frown\ g)\frown\ h \longrightarrow g\frown\ h$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash f\frown\ (g\frown\ h) = (f\frown\ g)\frown\ h$ **by** (*rule SChopAssoc*)
from *2 3* **show** *?thesis* **by** *auto*
qed

lemma *AndSChopCommute* :
$\vdash (f \wedge f1)\frown\ g = (f1 \wedge f)\frown\ g$
proof –
have *1*: $\vdash (f \wedge f1) = (f1 \wedge f)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftSChopEqvSChop*)
qed

lemma *BfAndSChopImport*:
$\vdash$ *bf* $f \wedge (f1\frown\ g) \longrightarrow (f \wedge f1)\frown g$
proof –
have *1*: $\vdash f \longrightarrow (f1 \longrightarrow f\wedge f1)$ **by** *auto*
hence *2*: $\vdash$ *bf* $f \longrightarrow$ *bf* $(f1 \longrightarrow f\wedge f1)$ **by** (*rule BfImpBfRule*)
have *3*: $\vdash$ *bf* $(f1 \longrightarrow (f \wedge f1)) \longrightarrow\ f1\frown\ g \longrightarrow (f \wedge f1)\frown\ g$ **by** (*rule BfSChopImpSChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

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

lemma *StateAndSChopImport*:
$\vdash$ (*init w*) $\wedge (f\frown g) \longrightarrow$ ((*init w*) $\wedge f)\frown g$
proof –
have *1*: $\vdash$ (*init w*)$\longrightarrow$ *bf* (*init w*) **by** (*rule StateImpBf*)
hence *2*: $\vdash$ (*init w*) $\wedge (f\frown\ g) \longrightarrow$ *bf* (*init w*) $\wedge (f\frown\ g)$ **by** *auto*
have *3*: $\vdash$ *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

7.4 Further Properties Df and Bf

lemma *AndFiniteImpDf*:
$\vdash f \wedge$ *finite* $\longrightarrow$ *df* f
proof –
have *1*: $\vdash$ *finite* $\longrightarrow f\frown$ *empty* $= f$ **by** (*rule SChopEmpty*)

have *2*: ⊢ *empty* ⟶ #*True* **by** *auto*
hence *3*: ⊢ *f*⌢ *empty* ⟶ *f*⌢ #*True* **by** (*rule RightSChopImpSChop*)
have *4* : ⊢ *f* ∧ *finite* ⟶ *f*⌢ #*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 ⊢ *f* = (*init w*) ∧ *f1*
shows ⊢ (*f*⁀ *g*) = ((*init w*) ∧ (*f1*⁀ *g*))
proof –
have *1*: ⊢ *f* = ((*init w*) ∧ *f1*) **using** *assms* **by** *auto*
hence *2*: ⊢ *f*⁀ *g* = (((*init w*) ∧ *f1*)⁀ *g*) **by** (*rule LeftSChopEqvSChop*)
have *3*: ⊢ ((*init w*) ∧ *f1*)⁀ *g* = ((*init w*) ∧ (*f1*⁀ *g*)) **by** (*rule StateAndSChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DfIntro*:
⊢ *f* ∧ *finite* ⟶ *df f*
proof –
have *1*: ⊢ *finite* ⟶ *f*⁀*empty* = *f* **by** (*rule SChopEmpty*)
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** (*simp add*: *RightSChopImpSChop*)
hence *6*: ⊢ *f*⁀*empty* ⟶ *df f* **by** (*simp add*: *df-d-def*)
from *1 6* **show** *?thesis* **using** *AndFiniteImpDf* **by** *blast*
qed

lemma *BfElim*:
⊢ *bf f* ∧ *finite* ⟶ *f*
proof –
have *1*: ⊢ ¬*f* ∧ *finite* ⟶ *df* (¬*f*) **by** (*rule DfIntro*)
have *2*: ⊢ (¬*f* ∧ *finite* ⟶ *df* (¬*f*)) ⟶ (¬(*df* (¬*f*)) ⟶ ¬(¬ *f* ∧ *finite*)) **by** *simp*
have *21*: ⊢ ¬(¬ *f* ∧ *finite*) = (*f* ∨ *inf*) **by** (*simp add*: *Valid-def finite-d-def*)
have *3*: ⊢ ¬ (*df* (¬*f*)) ⟶ *f* ∨ *inf* **using** *1 2 21* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *Prop13 bf-d-def finite-d-def*)
qed

lemma *BfContraPosImpDist*:
⊢ *bf* (¬*g* ⟶ ¬*f*) ⟶ (*bf f*) ⟶ (*bf g*)
proof –
have *1*: ⊢ *bf* (¬*g* ⟶ ¬*f*) ⟶ (*df* (¬*g*)) ⟶ (*df* (¬*f*)) **by** (*rule BfImpDfImpDf*)
hence *2*: ⊢ *bf* (¬*g* ⟶ ¬*f*) ⟶ (¬(*df* (¬*f*))) ⟶ (¬(*df* (¬*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*: $\vdash f = (((init\ w) \wedge f1) \vee ((init\ (\neg\ w)) \wedge f2))$
unfolding *ifthenelse-d-def* **by** (*metis Initprop*(*2*) *int-eq*)
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*: ⊢ ((*init w*) ∧ *f1*)⁀ *g* = ((*init w*) ∧ (*f1*⁀ *g*))
by (*rule StateAndSChop*)
have *5*: ⊢ ((*init* (¬ *w*)) ∧ *f2*)⁀ *g* = ((*init* (¬ *w*)) ∧ (*f2*⁀ *g*))
by (*rule StateAndSChop*)
have *6*: ⊢ *f*⁀ *g* = (((*init w*) ∧ *f1*⁀ *g*) ∨ ((*init* (¬ *w*)) ∧ *f2*⁀ *g*))
using *3 4 5* **by** *fastforce*
from *6* **show** *?thesis* **unfolding** *ifthenelse-d-def* **by** (*metis Initprop*(*2*) *inteq-reflection*)
qed

lemma *SChopOrEqvRule*:
assumes ⊢ *g* = (*g1* ∨ *g2*)
shows ⊢ *f*⁀*g* = ((*f*⁀*g1*) ∨ (*f*⁀*g2*))
proof –
have *1*: ⊢ *g* = (*g1* ∨ *g2*) **using** *assms* **by** *auto*
hence *2*: ⊢ *f*⁀ *g* = (*f*⁀ (*g1* ∨ *g2*)) **by** (*rule RightSChopEqvSChop*)
have *3*: ⊢ *f*⁀ (*g1* ∨ *g2*) = (*f*⁀ *g1* ∨ *f*⁀ *g2*) **by** (*rule SChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrSChopEqv*:
⊢ (*empty* ∨ *f*)⁀ *g* = (*g* ∨ (*f*⁀ *g*))
proof –
have *1*: ⊢ (*empty* ∨ *f*)⁀ *g* = ((*empty* ⁀ *g*) ∨ (*f*⁀ *g*)) **by** (*rule OrSChopEqv*)
have *2*: ⊢ *empty* ⁀ *g* = *g* **by** (*rule EmptySChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqv*:
⊢ (*empty* ∨ ○ *f*)⁀ *g* = (*g* ∨ ○(*f*⁀ *g*))
proof –
have *1*: ⊢ (*empty* ∨ ○ *f*)⁀ *g* = (*g* ∨ ((○ *f*)⁀ *g*)) **by** (*rule EmptyOrSChopEqv*)
have *2*: ⊢ (○ *f*)⁀ *g* = ○(*f*⁀ *g*) **by** (*rule NextSChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrSChopImpRule*:
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 LeftSChopImpSChop*)
have *3*: ⊢ (*empty* ∨ *f1*)⁀ *g* = (*g* ∨ (*f1*⁀ *g*)) **by** (*rule EmptyOrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrSChopEqvRule*:
assumes ⊢ *f* = (*empty* ∨ *f1*)
shows ⊢ *f*⁀ *g* = (*g* ∨ (*f1*⁀ *g*))
proof –

have *1*: $\vdash f = (empty \lor f1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f\frown g = ((empty \lor f1)\frown g)$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash (empty \lor f1)\frown g = (g \lor (f1\frown g))$ **by** (*rule EmptyOrSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopImpRule*:
assumes $\vdash f \longrightarrow empty \lor \bigcirc f1$
shows $\vdash f\frown g \longrightarrow g \lor \bigcirc(f1\frown g)$
proof –
have *1*: $\vdash f \longrightarrow empty \lor \bigcirc f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f\frown g \longrightarrow (empty \lor \bigcirc f1)\frown g$ **by** (*rule LeftSChopImpSChop*)
have *3*: $\vdash (empty \lor \bigcirc f1)\frown g = (g \lor \bigcirc(f1\frown g))$ **by** (*rule EmptyOrNextSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqvRule*:
assumes $\vdash f = (empty \lor \bigcirc f1)$
shows $\vdash f\frown g = (g \lor \bigcirc(f1\frown g))$
proof –
have *1*: $\vdash f = (empty \lor \bigcirc f1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f\frown g = ((empty \lor \bigcirc f1)\frown g)$ **by** (*rule LeftSChopEqvSChop*)
have *3*: $\vdash (empty \lor \bigcirc f1)\frown g = (g \lor \bigcirc(f1\frown g))$ **by** (*rule EmptyOrNextSChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopEmptyOrImpRule*:
assumes $\vdash g \longrightarrow empty \lor g1$
shows $\vdash f\frown g \land finite \longrightarrow f \lor (f\frown g1)$
proof –
have *1*: $\vdash g \longrightarrow empty \lor g1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f\frown g \longrightarrow (f\frown empty) \lor (f\frown g1)$ **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*:
$\vdash \Box (init\ w)\frown \Box (init\ w) \land inf \longrightarrow \Box (init\ w)$
by (*metis AndChopA BoxStateChopBoxEqvBox OrFiniteInf Prop03 int-eq lift-imp-trans schop-d-def*)

lemma *BoxStateSChopBoxEqvBox*:
$\vdash \Box (init\ w)\frown \Box (init\ w) = \Box (init\ w)$
proof –
have *1*: $\vdash (\Box (init\ w)) = ((init\ w) \land (empty \lor \bigcirc(\Box (init\ w))))$
by (*rule BoxEqvAndEmptyOrNextBox*)
hence *2*: $\vdash (\Box (init\ w)\frown \Box (init\ w)) =$
$((init\ w) \land ((empty \lor \bigcirc(\Box (init\ w)))\frown \Box (init\ w)))$
by (*metis StateAndSChop inteq-reflection*)
have *3*: $\vdash ((empty \lor \bigcirc(\Box (init\ w)))\frown \Box (init\ w)) =$
$(\Box (init\ w) \lor \bigcirc(\Box (init\ w)\frown \Box (init\ w)))$

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* (*no-types*) *DfState Initprop*(*2*) *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

7.5 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*($\square$ *f*)))
proof –
have *1*: $\vdash$ $\diamond$ ($\neg$*f*) = ($\neg$ ($\square$ *f*)) **by** (*simp add*: *DiamondNotEqvNotBox*)
have *2*: $\vdash$ *df*($\diamond$ ($\neg$*f*)) = *df*($\neg$ ($\square$ *f*)) **by** (*simp add*: *1 DfEqvDf*)
have *3*: $\vdash$ *df*($\neg$($\square$ *f*)) = ($\neg$(*bf* ($\square$ *f*))) **by** (*simp add*: *DfNotEqvNotBf*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DtDfNotEqvNotBtBf*:
$\vdash$ $\diamond$(*df* ($\neg$ *f*)) = ($\neg$ ($\square$(*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*)) = ($\neg$ $\square$(*bf f*)) **by** (*simp add*: *DiamondNotEqvNotBox*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaEqvBtBf*:
$\vdash$ *sba f* = $\square$ (*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*)))) = $\square$(*bf f*) **by** (*rule NotDiamondNotEqvBox*)
have *5*: $\vdash$ ($\neg$ (*sda* ($\neg$ *f*))) = $\square$(*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*:

$\vdash \quad \Box\ (bf\ \ f) = bf(\ \Box\ f)$
proof –
have *1*: $\vdash \quad sba\ \ f = \Box\ (bf\ \ f)$ **by** (*rule SBaEqvBtBf*)
have *2*: $\vdash \quad sba\ \ f = bf(\ \Box\ f)$ **by** (*rule SBaEqvBfBt*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateEqvSBaBoxState*:
$\vdash \quad \Box\ (init\ w) = \ sba\ (\Box\ (init\ w))$
proof –
have *1*: $\vdash (init\ w) = bf\ \ (init\ w)$ **by** (*rule StateEqvBf*)
hence *2*: $\vdash \Box\ (init\ w) = \Box\ (bf\ \ (init\ w))$ **by** (*rule BoxEqvBox*)
have *3*: $\vdash \Box\ (bf\ \ (init\ w)) = bf(\ \Box\ (init\ w))$ **by** (*rule BtBfEqvBfBt*)
have *4*: $\vdash \Box\ (init\ w) = \Box(\Box\ (init\ w))$ **by** (*rule BoxEqvBoxBox*)
hence *5*: $\vdash bf(\ \Box\ (init\ w)) = bf\ (\Box(\Box\ (init\ w)))$ **by** (*rule BfEqvBf*)
have *6*: $\vdash \ sba(\ \Box\ (init\ w)) = bf(\ \Box(\Box\ (init\ w)))$ **by** (*rule SBaEqvBfBt*)
from *2 3 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpBf*:
$\vdash \quad sba\ \ f \longrightarrow bf\ \ f$
proof –
have *1*: $\vdash \ sba\ \ f = \Box(bf\ \ f)$ **by** (*rule SBaEqvBtBf*)
have *2*: $\vdash \Box(bf\ \ f) \longrightarrow bf\ \ f$ **by** (*rule BoxElim*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *BaImpBf*:
$\vdash \quad ba\ \ f \longrightarrow bf\ \ f$
proof –
have *1*: $\vdash \ ba\ \ f = \Box(bi\ \ f)$ **by** (*rule BaEqvBtBi*)
have *2*: $\vdash \Box(bi\ \ f) \longrightarrow bi\ \ f$ **by** (*rule BoxElim*)
have *3*: $\vdash bi\ \ f \longrightarrow bf\ f$ **by** (*simp add*: *BiImpBf*)
from *1 2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *SBaImpBt*:
$\vdash \quad sba\ \ f \wedge finite \longrightarrow \Box\ f$
proof –
have *1*: $\vdash \ sba\ \ f = bf(\ \Box\ f)$ **by** (*rule SBaEqvBfBt*)
have *2*: $\vdash bf(\ \Box\ f) \wedge finite \longrightarrow \Box\ f$ **by** (*rule BfElim*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondImpSDa*:
$\vdash \quad \Diamond\ f \wedge finite \longrightarrow \ sda\ \ f$
by (*metis AndFiniteImpDf SDaEqvDfDt inteq-reflection*)

lemma *DfImpSDa*:
$\vdash \quad df\ \ f \longrightarrow \ sda\ \ f$

using *NowImpDiamond SDaEqvDtDf* **by** *fastforce*

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

lemma *SBaAndSChopImport*:
$\vdash\ \ \ sba\ f \wedge finite \wedge (g\frown\ g1)\ \longrightarrow (f \wedge g)\frown\ (f \wedge g1)$
proof –
 have *1*: $\vdash sba\ \ f \longrightarrow bf\ \ f$ **by** (*rule 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*:
$\vdash\ \ (\neg\ (f\frown\ g)) = f\ \mathit{syields}\ \ (\neg\ \ g)$
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*:
$\vdash\ \ \neg\ (\ \mathit{df}\ \ \#\mathit{False})$
proof –
have *1*: $\vdash (\mathit{init}\ \#\mathit{True}) \longrightarrow \mathit{bf}\ (\mathit{init}\ \#\mathit{True})$ **by** (*rule StateImpBf*)
hence *2*: $\vdash \#\mathit{True} \longrightarrow \mathit{bf}\ \#\mathit{True}$ **by** (*simp add*: *BfGen*)
have *3*: $\vdash \#\mathit{True}$ **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*: ⊢ $f \frown g \longrightarrow f \frown ((g \wedge \neg\ g1) \vee\ g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: ⊢ $f \frown ((g \wedge \neg\ g1) \vee\ g1) \longrightarrow (f \frown (g \wedge \neg\ g1)) \vee\ (f \frown g1)$ **by** (*rule SChopOrImp*)
have *4*: ⊢ $f \frown g \longrightarrow f \frown (g \wedge \neg\ g1) \vee\ f \frown g1$ **using** *2 3 MP* **by** *fastforce*
from *4* **show** *?thesis* **by** *auto*
qed

lemma *SChopAndSYieldsImp*:
⊢ $f \frown g \wedge f$ *syields* $g1 \longrightarrow f \frown (g \wedge g1)$
proof −
have *1*: ⊢ $g \longrightarrow (g \wedge g1) \vee\ \neg\ g1$ **by** *auto*
hence *2*: ⊢ $f \frown g \longrightarrow f \frown ((g \wedge\ g1) \vee\ \neg\ g1)$ **by** (*rule RightSChopImpSChop*)
have *3*: ⊢ $f \frown ((g \wedge\ g1) \vee\ \neg\ g1) \longrightarrow (f \frown (g \wedge\ g1)) \vee\ (f \frown (\neg\ g1))$ **by** (*rule SChopOrImp*)
have *4*: ⊢ $f \frown g \longrightarrow f \frown (g \wedge\ g1) \vee\ f \frown (\neg\ g1)$ **using** *2 3 MP* **by** *fastforce*
hence *5*: ⊢ $f \frown g \wedge \neg\ (f \frown (\neg\ g1)) \longrightarrow f \frown (g \wedge g1)$ **by** *auto*
from *5* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SChopAndSYieldsMP*:
⊢ $f \frown g \wedge f$ *syields* $(g \longrightarrow g1) \longrightarrow f \frown g1$
proof −
have *1*: ⊢ $f \frown g \wedge f$ *syields* $(g \longrightarrow g1) \longrightarrow f \frown (g \wedge (g \longrightarrow g1))$ **by** (*rule SChopAndSYieldsImp*)
have *2*: ⊢ $g \wedge (g \longrightarrow g1) \longrightarrow g1$ **by** *auto*
hence *3*: ⊢ $f \frown (g \wedge (g \longrightarrow g1)) \longrightarrow f \frown g1$ **by** (*rule RightSChopImpSChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *OrSYieldsImp*:
⊢ $(f \vee\ f1)$ *syields* $g = ((f$ *syields* $g) \wedge (f1$ *syields* $g))$
proof −
have *1*: ⊢ $((f \vee\ f1) \frown (\neg\ g)) = ((f \frown (\neg\ g)) \vee\ (f1 \frown (\neg\ g)))$ **by** (*rule OrSChopEqv*)
hence *2*: ⊢ $(\neg\ ((f \vee\ f1) \frown (\neg\ g))) = (\neg\ (f \frown (\neg\ g)) \wedge\ \neg(f1 \frown (\neg\ g)))$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *LeftSYieldsImpSYields*:
assumes ⊢ $f \longrightarrow f1$
shows ⊢ $(f1$ *syields* $g) \longrightarrow (f$ *syields* $g)$
proof −
have *1*: ⊢ $f \longrightarrow f1$ **using** *assms* **by** *auto*
hence *2*: ⊢ $f \frown (\neg\ g) \longrightarrow f1 \frown (\neg\ g)$ **by** (*rule LeftSChopImpSChop*)
hence *3*: ⊢ $\neg\ (f1 \frown (\neg\ g)) \longrightarrow \neg\ (f \frown (\neg\ g))$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *LeftSYieldsEqvSYields*:
assumes ⊢ $f = f1$
shows ⊢ $(f$ *syields* $g) = (f1$ *syields* $g)$
proof −
have *1*: ⊢ $f = f1$ **using** *assms* **by** *auto*
hence *2*: ⊢ $f \frown (\neg\ g) = f1 \frown (\neg\ g)$ **by** (*rule LeftSChopEqvSChop*)

hence *3*: $\vdash$ (¬ (*f*⌢ (¬ *g*))) = (¬ (*f1*⌢ (¬ *g*))) **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

7.6 Properties of SFin

lemma *SFinEqvTrueSChopAndEmpty*:
$\vdash$ *sfin f* = *#True*⌢(*f* ∧ *empty*)
proof −
have *01*: $\vdash$ *sfin f* = (¬ *fin* (¬ *f*))
by (*simp add*: *sfin-d-def*)
have *02*:$\vdash$ (¬ *fin* (¬ *f*)) = (¬ (□ (*empty* ⟶ ¬ *f*)))
by (*simp add*: *fin-d-def*)
have *03*:$\vdash$ (¬ (□ (*empty* ⟶ ¬ *f*))) = ◇(¬(*empty* ⟶ ¬ *f*))
by (*simp add*: *always-d-def*)
have *04*: $\vdash$ ¬(*empty* ⟶ ¬ *f*) = (*empty* ∧ *f*)
by *auto*
have *05*:$\vdash$ ◇(¬(*empty* ⟶ ¬ *f*)) = ◇ (*empty* ∧ *f*)
using *04* *inteq-reflection* **by** *fastforce*
from *01 02 03 05* **show** *?thesis*
by (*metis SChopAndCommute TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *DiamondSFin*:
$\vdash$ ◇(*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*⌢(*g* ∧ *sfin w*) ⟶ *sfin w*
using *DiamondSFin*
by (*metis SChopAndB SChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *SFinImpBox*:
$\vdash$ *sfin w* ⟶ □(*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*) ∧ (*f*⌢*g*) ⟶ *f*⌢((*sfin w*) ∧ *g*)
proof −
have *1*: $\vdash$ *sfin w* ⟶ □(*sfin w*) **by** (*rule SFinImpBox*)
hence *2*: $\vdash$ *sfin w* ∧ (*f*⌢*g*) ⟶ □(*sfin w*) ∧ (*f*⌢*g*) **by** *auto*
have *3*: $\vdash$ □(*sfin w*) ∧ (*f*⌢*g*) ⟶ *f*⌢((*sfin w*) ∧ *g*) **using** *BoxAndSChopImport* **by** *blast*
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SFinAndSChop*:
$\vdash$ (*f*⌢(*g* ∧ *sfin w*)) = (*sfin w* ∧ *f*⌢*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*: *itl-defs zero-enat-def*)

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*: $\vdash (\#\mathit{True} \frown (w \wedge \mathit{empty}) \wedge \mathit{empty}) = ((\#\mathit{True} \wedge \mathit{empty}) \frown (w \wedge \mathit{empty}))$
  by (*simp add*: *FiniteChopAndEmptyEqvChopAndEmpty schop-d-def*)
 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*:
$\vdash \Diamond\ f = (f\ \vee \bigcirc(\Diamond\ f))$
proof −
 have *1*: $\vdash \Box\ (\neg\ f) = ((\neg\ f) \wedge \mathit{wnext}(\Box\ (\neg\ f)))$
  by (*simp add*: *BoxEqvAndWnextBox*)
 have *2*: $\vdash (\neg\ \Diamond\ f) = ((\neg f) \wedge \mathit{wnext}(\Box\ (\neg\ f)))$

using *1* **by** (*simp add*: *always-d-def*)
have *3*: $\vdash\ \diamond f = (f \vee \neg(wnext(\Box\ (\neg f))))$
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*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*sfin* (*init w*)))
proof –
have *01*: $\vdash$ *sfin* (*init w*) = #*True*$\frown$((*init w*) $\wedge$ *empty*)
by (*simp add*: *SFinEqvTrueSChopAndEmpty*)
have *02*: $\vdash$ #*True*$\frown$((*init w*) $\wedge$ *empty*) = $\diamond$ ((*init w*) $\wedge$ *empty*)
by (*simp add*: *TrueSChopEqvDiamond*)
have *03*: $\vdash$ $\diamond$ ((*init w*) $\wedge$ *empty*) = (((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*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*) $\wedge$ *f*) $\vee$ $\bigcirc$((*sfin* (*init w*))$\frown$ *f*))
proof –
have *1*: $\vdash$ *sfin* (*init w*) = (((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*sfin* (*init w*)))
by (*rule SFinStateEqvStateAndEmptyOrNextSFinState*)
hence *2*: $\vdash$ (*sfin* (*init w*))$\frown$ *f* = (((*init w*) $\wedge$ *empty*)$\vee$ $\bigcirc$(*sfin* (*init w*)))$\frown$ *f*
by (*rule LeftSChopEqvSChop*)
have *3*: $\vdash$ (((*init w*) $\wedge$ *empty*)$\vee$ $\bigcirc$ (*sfin* (*init w*)))$\frown$ *f*
= (((*init w*) $\wedge$ *empty*)$\frown$ *f* $\vee$ ($\bigcirc$ (*sfin* (*init w*)))$\frown$ *f*)
by (*rule OrSChopEqv*)
have *4*: $\vdash$ ((*init w*) $\wedge$ *empty*)$\frown$ *f* = ((*init w*) $\wedge$ *f*)
by (*rule StateAndEmptySChop*)
have *5*: $\vdash$ ($\bigcirc$ (*sfin* (*init w*)))$\frown$ *f* = $\bigcirc$((*sfin* (*init w*))$\frown$ *f*)
by (*rule NextSChop*)
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSChopEqvDiamond*:
$\vdash$ (*sfin* (*init w*))$\frown$ *f* = $\diamond$ ((*init w*) $\wedge$ *f*)
proof –
have *1*: $\vdash$ (*sfin* (*init w*)) = (#*True*$\frown$((*init w*) $\wedge$ *empty*))
by (*simp add*: *SFinEqvTrueSChopAndEmpty*)
hence *2*: $\vdash$ (*sfin* (*init w*))$\frown$*f* = (#*True*$\frown$((*init w*) $\wedge$ *empty*))$\frown$*f*
by (*rule LeftSChopEqvSChop*)
have *3*: $\vdash$ #*True*$\frown$(((*init w*) $\wedge$ *empty*)$\frown$*f*) = (#*True*$\frown$((*init w*) $\wedge$ *empty*))$\frown$*f*
by (*rule SChopAssoc*)

have *4*: $\vdash$ *#True*$\frown$(((*init w*) $\wedge$ *empty*)$\frown$*f*)= $\diamond$ (((*init w*) $\wedge$ *empty*)$\frown$*f*)
  using *TrueSChopEqvDiamond* **by** *blast*
have *5*: $\vdash$ ((*init w*) $\wedge$ *empty*)$\frown$*f* = ((*init w*) $\wedge$ *f*)
  using *StateAndEmptySChop* **by** *blast*
hence *6*: $\vdash$ $\diamond$ (((*init w*) $\wedge$ *empty*)$\frown$*f*) = $\diamond$ ((*init w*) $\wedge$ *f*)
  by (*rule DiamondEqvDiamond*)
from *2 3 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSYields*:
$\vdash$ (*sfin* (*init w*)) *syields* (*init w*)
proof –
have *1*: $\vdash$ (*sfin* (*init w*))$\frown$ ($\neg$(*init w*)) = $\diamond$((*init w*) $\wedge$ $\neg$(*init w*))
  by (*rule SFinSChopEqvDiamond*)
have *2*: $\vdash$ $\neg$($\diamond$((*init w*) $\wedge$ $\neg$ (*init w*)))
  by (*rule NotDiamondAndNot*)
have *3*: $\vdash$ $\neg$ ((*sfin* (*init w*))$\frown$ ($\neg$ (*init w*)))
  using *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SFinEQvFinAndFinite*:
$\vdash$ (*finite* $\wedge$ *fin f*) = *sfin f*
by (*metis AndFinEqvChopAndEmpty DiamondSChopdef SFinEqvTrueSChopAndEmpty int-simps*(*20*) *inteq-reflection sometimes-d-def*)

lemma *AndFiniteImpAndSFinStateOrSFinNotState*:
$\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ (*f* $\wedge$ *sfin* (*init w*)) $\vee$ (*f* $\wedge$ *sfin* ($\neg$ (*init w*)))
proof –
have *1*: $\vdash$ ($\neg$ *fin* (*init* ($\neg$ *w*)) $\wedge$ *finite*) = (*fin* (*init* ($\neg$ $\neg$ *w*)) $\wedge$ *finite*)
  using *FinNotStateEqvNotFinState* **by** *blast*
have *2*: $\vdash$ (*fin* (*init* ($\neg$ $\neg$ *w*)) $\wedge$ *finite*) = (*fin* (*init* (*w*)) $\wedge$ *finite*)
  by *simp*
have *3*: $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ (*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*) $\vee$ (*f* $\wedge$ *finite*) $\wedge$ *fin* ($\neg$ *init w*)
  by (*simp add*: *ImpAndFinStateOrFinNotState*)
have *4*: $\vdash$ (*finite* $\wedge$ *fin* (*init w*)) = *sfin*(*init w*)
  using *SFinEQvFinAndFinite*[*of LIFT*(*init w*)] **by** *fastforce*
have *5*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* (*init w*)) = (*f* $\wedge$ *sfin* (*init w*))
  using *4* **by** *auto*
have *6*: $\vdash$ (*finite* $\wedge$ *fin* ($\neg$(*init w*))) = *sfin*($\neg$(*init w*))
  using *SFinEQvFinAndFinite*[*of LIFT*($\neg$(*init w*))] **by** *fastforce*
have *7*: $\vdash$ ((*f* $\wedge$ *finite*) $\wedge$ *fin* ($\neg$ (*init w*))) = (*f* $\wedge$ *sfin* ($\neg$ (*init w*)))
  using *6* **by** *auto*
show *?thesis*
  using *3 5 7* **by** *fastforce*
qed

lemma *AndSFinSChopEqvStateAndSChop*:
$\vdash$ (*f* $\wedge$ *sfin* (*init w*))$\frown$ *g* = *f*$\frown$ ((*init w*) $\wedge$ *g*)
proof –

have *1*: $\vdash$ (*sfin* (*init w*)) *syields* (*init w*)
by (*rule SFinSYields*)
have *2*: $\vdash$ f $\wedge$ *sfin* (*init w*) $\longrightarrow$ *sfin* (*init w*)
by *auto*
hence *3*: $\vdash$ (*sfin* (*init w*)) *syields* (*init w*) $\longrightarrow$
(f $\wedge$ *sfin* (*init w*)) *syields* (*init w*)
using *LeftSYieldsImpSYields* **by** *metis*
have *4*: $\vdash$ (f $\wedge$ *sfin* (*init w*)) *syields* (*init w*)
using *1 3 MP* **by** *fastforce*
have *5*: $\vdash$ (f $\wedge$ *sfin* (*init w*))$\frown$ g $\wedge$ (f $\wedge$ *sfin* (*init w*)) *syields* (*init w*)
$\longrightarrow$ (f $\wedge$ *sfin* (*init w*))$\frown$ (g $\wedge$ (*init w*))
by (*rule SChopAndSYieldsImp*)
have *6*: $\vdash$ (f $\wedge$ *sfin* (*init w*))$\frown$ g $\longrightarrow$ (f $\wedge$ *sfin* (*init w*))$\frown$ (g $\wedge$ (*init w*))
using *4 5* **by** *fastforce*
have *7*: $\vdash$ (f $\wedge$ *sfin* (*init w*))$\frown$ (g $\wedge$ (*init w*)) $\longrightarrow$ $f\frown$ (g $\wedge$ (*init w*))
by (*rule AndSChopA*)
have *8*: $\vdash$ g $\wedge$ (*init w*) $\longrightarrow$ (*init w*) $\wedge$ 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 *150*: $\vdash$ ((*init* ($\neg$ *w*)) $\wedge$ ((*init w*) $\wedge$ g)) = *#False*
using *Initprop*(*2*) **by** *fastforce*
have *15*: $\vdash$ $\neg$($\Diamond$((*init* ($\neg$ *w*)) $\wedge$ ((*init w*) $\wedge$ g)))
by (*metis 150 NotDiamondAndNot int-eq int-simps*(*21*))
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 Initprop*(*2*) *SFinprop*(*3*) *int-eq*)

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*: $\vdash (\neg (f \frown (\neg\ g)) \wedge \neg (f \frown (\neg\ g1))) = (\neg (f \frown (\neg (g \wedge g1))))$ **using** *1 4* **by** *fastforce*
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 (*metis DiIntro LeftChopImpMoreRule di-d-def more-d-def next-d-def*)
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 (*metis 1 FmoreChopImpFmore fmore-d-def int-eq schop-d-def*)
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** (*simp add*: *FiniteChopMoreEqvMore int-iffD1 sometimes-d-def*)
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*: *more-d-def*)
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 (*metis ChopImpDi di-d-def more-d-def next-d-def*)
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 (*simp add*: *BfAndEqvBfAndBf*)
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*:
$\vdash\ \ sba\ (g \longrightarrow g1) \wedge finite \longrightarrow f \frown g \longrightarrow f \frown g1$
proof –
have *1*: $\vdash\ sba\ (g \longrightarrow g1) \wedge finite \longrightarrow \Box(g \longrightarrow g1)$ **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*: ⊢ *sba f* = *sba*(*sba f*) **by** (*rule SBaEqvSBaSBa*)
 have *2*: ⊢ *sba* (*sba f*) ⟶ *bf* (*sba f*) **by** (*rule SBaImpBf*)
 have *3*: ⊢ *sba f* ⟶ *bf* (*sba f*) **using** *1 2* **by** *fastforce*
 hence *4*: ⊢ *sba f* ⟶ ¬ (*df* (¬ (*sba f*))) **by** (*simp add*: *bf-d-def*)
 from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *DfNotBaImpNotBa*:
 ⊢ *df* (¬ (*ba f*)) ⟶ ¬ (*ba f*)
proof −
 have *1*: ⊢ *ba f* = *ba*(*ba f*) **by** (*rule BaEqvBaBa*)
 have *2*: ⊢ *ba* (*ba f*) ⟶ *bf* (*ba f*) **by** (*rule BaImpBf*)
 have *3*: ⊢ *ba f* ⟶ *bf* (*ba f*) **using** *1 2* **by** *fastforce*
 hence *4*: ⊢ *ba f* ⟶ ¬ (*df* (¬ (*ba f*))) **by** (*simp add*: *bf-d-def*)
 from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotSBaSChopImpNotSBa*:
⊢ (¬ (*sba f*))⌢ *g* ⟶ ¬ (*sba f*)
proof −
 have *1*: ⊢ (¬ (*sba f*))⌢ *g* ⟶ *df* (¬ (*sba f*)) **by** (*rule SChopImpDf*)
 have *2*: ⊢ *df* (¬ (*sba f*)) ⟶ ¬ (*sba f*) **by** (*rule DfNotSBaImpNotSBa*)
 from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NotBaSChopImpNotSBa*:
⊢ (¬ (*ba f*))⌢ *g* ⟶ ¬ (*ba f*)
proof −
 have *1*: ⊢ (¬ (*ba f*))⌢ *g* ⟶ *df* (¬ (*ba f*)) **by** (*rule SChopImpDf*)
 have *2*: ⊢ *df* (¬ (*ba f*)) ⟶ ¬ (*ba f*) **by** (*rule DfNotBaImpNotBa*)
 from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *DiamondSFinImpSFin*:
 ⊢ ◇ (*sfin f*) ⟶ *sfin f*
proof −
 have *1*: ⊢ *sfin f* = #*True*⌢(*f* ∧ *empty*)
  by (*rule SFinEqvTrueSChopAndEmpty*)
 hence *2*: ⊢ ◇ (*sfin f*) = #*True*⌢(#*True*⌢(*f* ∧ *empty*))
  by (*metis DiamondSChopdef inteq-reflection*)
 have *3*: ⊢ #*True*⌢(#*True*⌢(*f* ∧ *empty*)) = (#*True*⌢#*True*)⌢(*f* ∧ *empty*)
  by (*rule SChopAssoc*)
 have *4*: ⊢ (#*True*⌢#*True*)⌢(*f* ∧ *empty*) ⟶ #*True*⌢(*f* ∧ *empty*)
  using *1 2 3*
  by (*metis SChopImpDiamond TrueEqvTrueSChopTrue inteq-reflection*)
 from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopSFinImpSFin*:

⊢ *f* ⌢ *sfin* (*init w*) ⟶ *sfin* (*init w*)
proof –
have *1*: ⊢ *f* ⌢ *sfin* (*init w*) ⟶ ◇ (*sfin* (*init w*)) **by** (*rule SChopImpDiamond*)
have *2*: ⊢ ◇ (*sfin* (*init w*)) ⟶ *sfin* (*init w*) **by** (*rule DiamondSFinImpSFin*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *SFinImpSYieldsSFin*:
⊢ *sfin* (*init w*) ⟶ *f syields* (*sfin* (*init w*))
proof –
have *1*: ⊢ *f*⌢ (*sfin* (*init* (¬ *w*))) ⟶ (*sfin* (*init* (¬ *w*)))
by (*simp add*: *SChopSFinImpSFin*)
have *2*: ⊢ *finite* ⟶ (¬ (*sfin* (*init w*))) = (*sfin* (*init* (¬ *w*)))
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
hence *3*: ⊢ *finite* ⟶ *f* ⌢ (¬ (*sfin* (*init w*))) = *f*⌢ (*sfin* (*init* (¬ *w*)))
using *FiniteRightSChopEqvSChop* **by** *blast*
have *4*: ⊢ *f*⌢ (¬ (*sfin* (*init w*))) ∧ *finite* ⟶ (¬ (*sfin* (*init w*)))
using *1 2 3* **by** *fastforce*
hence *5*: ⊢ *sfin* (*init w*) ⟶ ¬ (*f*⌢ (¬ (*sfin* (*init w*))))
by (*metis* (*no-types*, *lifting*) *DiamondFin SChopImpDiamond int-simps*(*32*) *int-simps*(*4*) *inteq-reflection sfin-d-def*)
from *5* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *SChopAndSFin*:
⊢ ((*f*⌢ *g*) ∧ (*sfin* (*init w*))) = *f*⌢ (*g* ∧ (*sfin* (*init w*)))
proof –
have *1*: ⊢ *sfin* (*init w*) ⟶ *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*: $\vdash f \frown (g \wedge \ sfin \ \ (init\ (\neg\ w))\ ) = f \frown (g \wedge \neg\ (\ sfin \ \ (init\ w)) \wedge finite)$
using *RightSChopEqvSChop* **by** *blast*
from *1 2 4* **show** *?thesis*
by (*metis DiamondEmptyEqvFinite Prop10 SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqv-Diamond*
inteq-reflection)
qed

lemma *SFinSChopChain*:
$\vdash \ \ (((init\ w\ ) \ \wedge finite \longrightarrow \ sfin \ \ (init\ w1)))\frown$
$(((init\ w1) \ \wedge finite \longrightarrow \ sfin \ \ (init\ w2)))$
$\wedge\ finite$
$\longrightarrow (((init\ w\ ) \ \wedge finite \longrightarrow \ sfin \ \ (init\ w2)))$
proof –
have *01*: $\vdash (init\ w \wedge finite \wedge$
$((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))) =$
$(init\ w \wedge\ empty) \frown (((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$
by (*meson Prop04 SChopAndCommute StateAndEmptySChop*)
have *02*: $\vdash (finite \wedge init\ w) = (init\ w \wedge\ empty) \frown finite$
by (*metis StateAndEmptySChop inteq-reflection lift-and-com*)
have *03*: $\vdash\ init\ w \wedge finite \wedge$
$((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow finite \wedge init\ w$
by *force*
have *04*: $\vdash(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) =$
$(init\ w \wedge (finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)))$
by (*simp add*: *StateAndChop*)
have *05*: $\vdash\ init\ w \wedge finite \wedge ((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow$
$(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))$
by (*metis 04 AndChopCommute ChopAndB StateAndEmptyChop int-eq*)
have *06*: $\vdash\ init\ w \wedge finite \wedge ((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow$
$(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$
by (*meson 03 05 Prop12*)
have *1*: $\vdash \ \ (init\ w) \wedge finite \wedge$
$((init\ w) \wedge finite \longrightarrow \ sfin \ \ (init\ w1))\frown$
$((init\ w1) \wedge finite \longrightarrow \ sfin \ \ (init\ w2))$
$\longrightarrow$
$((init\ w) \wedge finite \wedge \ ((init\ w) \wedge finite \longrightarrow sfin \ \ (init\ w1)))\frown$
$(((init\ w1) \wedge finite\longrightarrow \ sfin \ \ (init\ w2)) \wedge finite)$
unfolding *schop-d-def* **using** *06 ChopAndFiniteDist* **by** *fastforce*
have *2*: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite\longrightarrow \ sfin \ \ (init\ w1)) \longrightarrow$
$sfin \ \ (init\ w1)$
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 *51*: ⊢ ◇(*init w1* ∧ *finite* ∧ (*init w1* ∧ *finite* ⟶ *sfin* (*init w2*))) ⟶ *sfin* (*init w2*)
using *42 5 lift-imp-trans* **by** *blast*
have *52*: ⊢ *init w* ∧ *finite* ∧ (*init w* ∧ *finite* ⟶ *sfin* (*init w1*)) ⌢ (*init w1* ∧ *finite* ⟶ *sfin* (*init w2*))
⟶ *sfin* (*init w1*) ⌢ ((*init w1* ∧ *finite* ⟶ *sfin* (*init w2*)) ∧ *finite*)
by (*meson 1 3 lift-imp-trans*)
have *53*: ⊢ *init w* ∧ *finite* ∧ (*init w* ∧ *finite* ⟶ *sfin* (*init w1*)) ⌢ (*init w1* ∧ *finite* ⟶ *sfin* (*init w2*))
⟶ ◇(*init w1* ∧ (*init w1* ∧ *finite* ⟶ *sfin* (*init w2*)) ∧ *finite*)
by (*metis 52 SFinSChopEqvDiamond inteq-reflection*)
have *6*: ⊢ (*init w*) ∧ *finite* ∧ ((*init w*) ∧ *finite* ⟶ *sfin* (*init w1*))⌢
((*init w1*) ∧ *finite* ⟶ *sfin* (*init w2*))
⟶ *sfin* (*init w2*)
by (*metis 42 5 53 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*)
by (*metis ChopEmpty SChopAssoc StateAndSChopImport inteq-reflection schop-d-def*)
have *2*: ⊢ (*init w*) ∧ *f* ∧ *finite* ⟶ *sfin* (*init w1*)
using *assms* **by** *auto*
have *21*: ⊢ (*init w* ∧ *f* ∧ *finite*);(*f1* ∧ *finite*) = ((*init w* ∧ *f*);*f1* ∧ *finite*)
by (*metis ChopAndFiniteDist StateAndChop StateAndSChop inteq-reflection schop-d-def*)
have *22*: ⊢ (*init w* ∧ *f* ∧ *finite*) = ((*init w* ∧ *f* ∧ *finite*) ∧ *sfin* (*init w1*))
using *2 Prop10* **by** *blast*
hence *3*: ⊢ ((*init w*) ∧ *f*)⌢ (*f1* ∧ *finite*) ⟶ (*sfin* (*init w1*))⌢ (*f1* ∧ *finite*)
using *21 22*
by (*metis* (*no-types, opaque-lifting*) *2 ChopEmpty EmptySChop SChopAssoc StateAndSChop StateAndSChopImpSChopRule int-eq schop-d-def*)
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*: $\vdash \Diamond((init\ w1) \wedge f1 \wedge finite) \longrightarrow \Diamond\ (sfin\ \ (init\ w2))$
  by (*rule DiamondImpDiamond*)
have *7*: $\vdash \Diamond(\ sfin\ \ (init\ w2)) \longrightarrow\ \ sfin\ \ (init\ w2)$
  using *DiamondSFinImpSFin* **by** *blast*
from *1 3 4 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopRep*:
assumes $\vdash\ \ (init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge\ \ sfin\ \ (init\ w1)$
$\vdash\ \ (init\ w1) \wedge g \wedge finite \longrightarrow g1$
shows $\vdash\ \ (init\ w) \wedge ((f\frown g) \wedge finite) \longrightarrow (f1\frown g1)$
proof –
have *1*: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow (f1 \wedge\ \ sfin\ \ (init\ w1))$
  using *assms* **by** *auto*
hence *2*: $\vdash (init\ w) \wedge (f\frown (g \wedge finite))\ \ \longrightarrow (f1 \wedge\ \ sfin\ \ (init\ w1))\frown (g \wedge finite)$
  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 *31*: $\vdash\ \ (init\ w) \wedge ((f\frown g) \wedge finite) \longrightarrow f1\frown ((init\ w1) \wedge (g \wedge finite))$
  using *2 3* **by** (*metis ChopEmpty SChopAssoc inteq-reflection schop-d-def*)
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*
show *?thesis* **using** *31 5* **by** *fastforce*
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*)
show *?thesis*
  by (*metis 3 4 5 6 Prop12 lift-imp-trans*)
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 (*simp add*: *FiniteChopFmoreEqvFmore 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** *fastforce*
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*: ⊢ ◇ (*more*⌢*f*) = #*True*⌢(*more*⌢*f*)
by (*simp add*: *DiamondSChopdef*)
have *13*: ⊢ #*True*⌢(*more*⌢*f*) = (#*True*⌢*more*)⌢*f*
by (*rule SChopAssoc*)
have *14*: ⊢ ◇ (*more*⌢*f*) = *more*⌢*f*
using *12 13* **by** (*metis TrueSChopMoreEqvMore inteq-reflection*)
have *2*: ⊢ *more* ⌢ *f* = ○(◇ *f*)
using *MoreSChopEqvNextDiamond* **by** *blast*
have *3*: ⊢ ◇ (*f* ∧ *finite*) ⟶ ○(◇ *f*)
using *11 14 2* **by** *fastforce*
have *31*: ⊢ ◇ (*f* ∧ *finite*) = ((◇ *f*) ∧ *finite*)
by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SFinAndSChop SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection lift-and-com*)
have *32*: ⊢ (◇ *f*) ∧ *finite* ⟶ ○(◇ *f*)
using *3 31* **by** *fastforce*
hence *4*: ⊢ *finite* ⟶ ¬ (◇ *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*: ⊢ ¬ (◇ *f*) ⟶ ¬ *f*
by (*simp add*: *NowImpDiamond*)
from *4 5* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *MoreSChopContraFinite*:
assumes ⊢ (*f* ∧ ¬ *g*) ∧ *finite* ⟶ (*more* ⌢ (*f* ∧ ¬ *g*))
shows ⊢ *f* ∧ *finite* ⟶ *g*
proof −
have *1*: ⊢ (*f* ∧ ¬ *g*) ∧ *finite* ⟶ (*more* ⌢ (*f* ∧ ¬ *g*)) **using** *assms* **by** *auto*
hence *2*: ⊢ *finite* ⟶ ¬ (*f* ∧ ¬ *g*) **by** (*simp add*: *MoreSChopLoopFinite*)
from *2* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *SChopLoop*:
assumes ⊢ *f* ⟶ *g*⌢*f*
⊢ *g* ⟶ *fmore*
shows ⊢ *finite* ⟶ ¬ *f*
proof −
have *1*: ⊢ *f* ⟶ *g*⌢ *f* **using** *assms* **by** *auto*
have *2*: ⊢ *g* ⟶ *more* **using** *assms* **by** (*simp add*: *Prop12 fmore-d-def*)
hence *3*: ⊢ *g*⌢ *f* ⟶ *more* ⌢ *f* **by** (*rule LeftSChopImpSChop*)
have *4*: ⊢ *f* ⟶ *more* ⌢ *f* **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **using** *MoreSChopLoop* **by** *auto*
qed

lemma *SChopLoopB*:
assumes ⊢ *f* ⟶ *g*⌢*f*
⊢ *g* ⟶ *more*
shows ⊢ *finite* ⟶ ¬ *f*
proof −
have *1*: ⊢ *f* ⟶ *g*⌢ *f* **using** *assms* **by** *auto*

have *2*: ⊢ $g \longrightarrow$ *more* **using** *assms* **by** *auto*
hence *3*: ⊢ $g\frown f \longrightarrow$ *more* $\frown f$ **by** (*rule LeftSChopImpSChop*)
have *4*: ⊢ $f \longrightarrow$ *more* $\frown f$ **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **using** *MoreSChopLoop* **by** *blast*
qed

lemma *SChopContra*:
assumes ⊢ $f \wedge \neg\ g \longrightarrow h\frown f \wedge \neg\ (h\frown g)$
⊢ $h \longrightarrow$ *fmore*
shows ⊢ $f \wedge$ *finite* $\longrightarrow g$
proof −
have *1*: ⊢ $f \wedge \neg\ g \longrightarrow h\frown f \wedge \neg\ (h\frown g)$ **using** *assms* **by** *auto*
have *2*: ⊢ $h \longrightarrow$ *more* **using** *assms* **by** (*simp add*: *Prop12 fmore-d-def*)
have *3*: ⊢ $h\frown f \wedge \neg\ (h\frown g) \longrightarrow h\frown (f \wedge \neg\ g)$ **by** (*rule SChopAndNotSChopImp*)
have *4*: ⊢ $h\frown (f \wedge \neg\ g) \longrightarrow$ *more* $\frown (f \wedge \neg\ g)$ **using** *2* **by** (*rule LeftSChopImpSChop*)
have *5*: ⊢ $f \wedge \neg\ g \longrightarrow$ *more* $\frown (f \wedge \neg\ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreSChopContra* **by** *auto*
qed

lemma *SChopContraB*:
assumes ⊢ $f \wedge \neg\ g \longrightarrow h\frown f \wedge \neg\ (h\frown g)$
⊢ $h \longrightarrow$ *more*
shows ⊢ $f \wedge$ *finite* $\longrightarrow g$
proof −
have *1*: ⊢ $f \wedge \neg\ g \longrightarrow h\frown f \wedge \neg\ (h\frown g)$ **using** *assms* **by** *auto*
have *2*: ⊢ $h \longrightarrow$ *more* **using** *assms* **by** *auto*
have *3*: ⊢ $h\frown f \wedge \neg\ (h\frown g) \longrightarrow h\frown (f \wedge \neg\ g)$ **by** (*rule SChopAndNotSChopImp*)
have *4*: ⊢ $h\frown (f \wedge \neg\ g) \longrightarrow$ *more* $\frown (f \wedge \neg\ g)$ **using** *2* **by** (*rule LeftSChopImpSChop*)
have *5*: ⊢ $f \wedge \neg\ g \longrightarrow$ *more* $\frown (f \wedge \neg\ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreSChopContra* **by** *blast*
qed

7.7 Properties of Halt

lemma *HaltSChopEqv*:
⊢ ((*halt* (*init w*)) $\frown f$) = (*if*$_i$ (*init w*) *then* (*f*) *else* (○((*halt* (*init w*))$\frown f$)))
proof −
have *1*: ⊢ *halt*(*init w*) =
(*if*$_i$ (*init w*) *then* *empty* *else* (○(*halt* (*init w*))))
by (*rule HaltStateEqvIfStateThenEmptyElseNext*)
hence *2*: ⊢ ((*halt*(*init w*))$\frown f$) =
(*if*$_i$ (*init w*) *then* (*empty*$\frown f$) *else* (○(*halt* (*init w*))$\frown f$))
by (*rule IfSChopEqvRule*)
have *3*: ⊢ *empty* $\frown f = f$
by (*rule EmptySChop*)
have *4*: ⊢ (○ (*halt* (*init w*)))$\frown f$ = ○(*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*: $\vdash$ $\neg$ (*init w*) $\wedge$ (*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ $\bigcirc$(*halt* (*init w*)$\frown$ *f*)
by (*rule NotAndHaltSChopImpNext*)
have *2*: $\vdash$ $\bigcirc$(*halt* (*init w*)$\frown$ *f*) $\longrightarrow$ *skip syields* (*halt* (*init w*)$\frown$ *f*)
by (*rule NextImpSkipSYields*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndEmptyEqvSChopAndEmpty*:
$\vdash$ ((*#True*$\frown$(*f* $\wedge$ *empty*)) $\wedge$ *g*) = (*g*$\frown$(*f*$\wedge$ *empty*))
proof –
have *1*: $\vdash$ (*#True*$\frown$(*f* $\wedge$ *empty*)) $\wedge$ *g* $\longrightarrow$ *g* $\frown$(*f* $\wedge$ *empty*)
by (*simp add*: *FiniteChopAndEmptyEqvChopAndEmpty int-iffD1 schop-d-def*)
have *2*: $\vdash$ *g*$\frown$(*f*$\wedge$ *empty*) $\longrightarrow$ (*#True*$\frown$(*f* $\wedge$ *empty*)) $\wedge$ *g*
by (*metis AndSFinEqvSChopAndEmpty Prop12 SFinEqvTrueSChopAndEmpty int-iffD1 inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotSChopSkipEqvFmoreAndNotSChopSkip*:
$\vdash$ ($\neg$ *f*)$\frown$*skip* = (*fmore* $\wedge$ $\neg$(*f*$\frown$*skip*))
proof –
have *1*: $\vdash$ ($\neg$ *f*)$\frown$*skip* = (($\neg$*f* $\wedge$ *finite*);*skip*)
by (*simp add*: *schop-d-def*)
have *2*: $\vdash$ ($\neg$*f* $\wedge$ *finite*);*skip* = ($\neg$(*f* $\vee$ *inf*));*skip*
by (*metis* (*no-types*, *lifting*) *LeftChopEqvChop finite-d-def int-simps*(*14*) *int-simps*(*33*)
inteq-reflection)
have *3*: $\vdash$ ($\neg$(*f* $\vee$ *inf*));*skip* = (*more* $\wedge$ $\neg$((*f* $\vee$ *inf*);*skip*))

using *NotChopSkipEqvMoreAndNotChopSkip* **by** *blast*
have *4*: $\vdash (f \vee inf);skip = (f;skip \vee inf)$
by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv inteq-reflection*)
have *5*: $\vdash (more \wedge \neg((f \vee inf);skip)) = (more \wedge \neg(f;skip \vee inf))$
using *4* **by** *auto*
have *6*: $\vdash (more \wedge \neg(f;skip \vee inf)) = (more \wedge \neg(f;skip) \wedge finite)$
unfolding *finite-d-def* **by** *fastforce*
have *7*: $\vdash (more \wedge \neg(f;skip) \wedge finite) = (more \wedge \neg(f \frown skip \vee (f \wedge inf)) \wedge finite)$
by (*metis ChopEmpty ChopSChopdef inteq-reflection*)
have *8*: $\vdash (more \wedge \neg(f \frown skip \vee (f \wedge inf)) \wedge finite) =$
$(more \wedge \neg(f \frown skip) \wedge \neg(f \wedge inf) \wedge finite)$
by *auto*
have *9*: $\vdash (\neg(f \wedge inf) \wedge finite) = finite$
unfolding *finite-d-def* **by** *force*
have *10*: $\vdash (more \wedge \neg(f \frown skip) \wedge \neg(f \wedge inf) \wedge finite) =$
$(more \wedge \neg(f \frown skip) \wedge finite)$
using *9* **by** *fastforce*
have *11*: $\vdash (more \wedge \neg(f \frown skip) \wedge finite) = (fmore \wedge \neg(f \frown skip))$
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*:
$\vdash \quad halt\ (\ init\ w) \frown\ f \wedge finite \longrightarrow \neg\ (\ halt\ (\ init\ w) \frown\ (\neg\ \ f))$
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*)⌢ *f* ∧ *halt* (*init w*)⌢ (¬*f*) ⟶
○ (*halt* (*init w*)⌢ *f* ∧ *halt* (*init w*)⌢ (¬*f*))
using *7 8* **by** *fastforce*
hence *10*: $\vdash$ *finite* ⟶ ¬(*halt* (*init w*)⌢ *f* ∧ *halt* (*init w*)⌢ (¬*f*))
using *NextLoop* **by** *blast*
from *10* **show** *?thesis* **by** *auto*
qed

lemma *HaltSChopImpHaltSYields*:
$\vdash$ *halt* (*init w*)⌢ *f* ∧ *finite* ⟶ (*halt* (*init w*)) *syields f*
proof −
have *1*: $\vdash$ *halt* (*init w*)⌢ *f* ∧ *finite* ⟶ ¬ (*halt* (*init w*)⌢ (¬ *f*))
by (*rule HaltSChopImpNotHaltSChopNot*)
from *1* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

lemma *HaltSChopAnd*:
$\vdash$ (*halt* (*init w*))⌢ *f* ∧ (*halt* (*init w*))⌢ *g* ∧ *finite*⟶ (*halt* (*init w*))⌢ (*f* ∧ *g*)
proof −
have *1*: $\vdash$ (*halt* (*init w*))⌢ *g* ∧ *finite* ⟶ (*halt* (*init w*)) *syields g*
by (*rule HaltSChopImpHaltSYields*)
hence *2*: $\vdash$ (*halt* (*init w*))⌢ *f* ∧ (*halt* (*init w*))⌢ *g* ∧ *finite* ⟶
(*halt* (*init w*))⌢ *f* ∧ (*halt* (*init w*)) *syields g*
by *auto*
have *3*: $\vdash$ (*halt* (*init w*))⌢ *f* ∧ (*halt* (*init w*)) *syields g*⟶
(*halt* (*init w*))⌢ (*f* ∧ *g*)
by (*rule SChopAndSYieldsImp*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltAndSChopAndHaltSChopImpHaltAndSChopAnd*:
$\vdash$ (*halt* (*init w*) ∧ *f*)⌢ *f1* ∧ (*halt* (*init w*)⌢ *g*) ∧ *finite*
⟶ (*halt* (*init w*) ∧ *f*)⌢ (*f1* ∧ *g*)
proof −
have *1*: $\vdash$ *f1* ⟶ ¬ *g* ∨ (*f1* ∧ *g*)
by *auto*
hence *2*: $\vdash$ (*halt* (*init w*) ∧ *f*)⌢ *f1* ⟶
(*halt* (*init w*) ∧ *f*)⌢ (¬ *g*) ∨ ((*halt* (*init w*) ∧ *f*)⌢ (*f1* ∧ *g*))
by (*rule SChopOrImpRule*)
have *3*: $\vdash$ (*halt* (*init w*) ∧ *f*)⌢ (¬ *g*) ⟶ *halt* (*init w*)⌢ (¬ *g*)
by (*rule AndSChopA*)
have *31*: $\vdash$ (*halt* (*init w*) ∧ *f*)⌢ *f1* ⟶
halt (*init w*)⌢ (¬ *g*) ∨ ((*halt* (*init w*) ∧ *f*)⌢ (*f1* ∧ *g*))
using *23* **by** *fastforce*
have *4*: $\vdash$ *halt* (*init w*)⌢ *g* ∧ *finite* ⟶ ¬ (*halt* (*init w*)⌢ (¬ *g*))
by (*rule HaltSChopImpNotHaltSChopNot*)
hence *41*: $\vdash$ (*halt* (*init w*)⌢ (¬ *g*)) ∧ *finite* ⟶ ¬(*halt* (*init w*)⌢ *g*)
by *auto*
have *42*: $\vdash$ (*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 *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*) $\wedge$ (($\Box$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*))) $\longrightarrow$
(*halt* (*init w*)$\frown$ *f*) $\wedge$ $\neg$ (*init w*)
by *fastforce*
have *6*: $\vdash$ *halt* (*init w*)$\frown$ *f* = if_i (*init w*) *then f else* ($\bigcirc$(*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*) $\wedge$ ($\Box$($\neg$ (*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*: $\vdash$ $\Box$ ($\neg$ (*init w*)) $\longrightarrow$ *empty* $\vee$ $\bigcirc$($\Box$($\neg$(*init w*)))
by (*metis BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq*)
hence *9*: $\vdash$ (($\Box$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*))) $\longrightarrow$
$\neg$ (*halt* (*init w*)$\frown$ *f*) $\vee$ $\bigcirc$(($\Box$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
by (*rule EmptyOrNextSChopImpRule*)
hence *10*: $\vdash$ ((*halt* (*init w*))$\frown$ *f*) $\wedge$ ($\Box$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$
$\bigcirc$(($\Box$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
by *fastforce*
have *11*: $\vdash$ (*halt* (*init w*))$\frown$ *f* $\wedge$ ($\Box$ ($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)) $\longrightarrow$
$\bigcirc$((*halt* (*init w*))$\frown$ *f*) $\wedge$ $\bigcirc$(($\Box$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
using *7 10* **by** *fastforce*
have *12*: $\vdash$ $\bigcirc$((*halt* (*init w*))$\frown$ *f*) $\wedge$ $\bigcirc$(($\Box$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*)))
$\longrightarrow$ $\bigcirc$(((*halt* (*init w*))$\frown$ *f*) $\wedge$ (($\Box$($\neg$ (*init w*)))$\frown$ ($\neg$ (*halt* (*init w*)$\frown$ *f*))))

using *NextAndEqvNextAndNext* **by** *fastforce*
have *13*: $\vdash$ (*halt* (*init w*))⌢ *f* $\wedge$ ($\Box$ ($\neg$ (*init w*)))⌢ ($\neg$ (*halt* (*init w*)⌢ *f*)) $\longrightarrow$
○(((*halt* (*init w*))⌢ *f*) $\wedge$ (($\Box$($\neg$ (*init w*)))⌢ ($\neg$ (*halt* (*init w*)⌢ *f*))))
using *11 12* **by** *fastforce*
hence *14*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ((*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* $\wedge$ *finite* $\longrightarrow$ $\neg$ (($\Box$ ($\neg$ (*init w*)))⌢ ($\neg$ (*halt* (*init w*)⌢ *f*)))
by *auto*
from *15* **show** *?thesis* **by** (*simp add*: *syields-d-def*)
qed

7.8 Properties of Groups of strong chops

lemma *NestedSChopImpSChop*:
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 RightSChopImpSChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

end

8 Finite and Infinite ITL theorems using Weak Chop

theory *Chopstar*
imports
SChopTheorems
begin

This theory defines the chopstar operator for infinite ITL and provides a library of lemmas. We also define the strong version schopstar, the weak version wpowerstar and a semantic version achopstar. The wpowerstar corresponds to the Kleene star operator from Kleene Algebra [1]. We provide lemmas that express various relationships between them. We also ported the numerous Kleene algebra lemmas from [1] to ITL.

8.1 Definitions

primrec *wpower-d* :: ('*a*::*world*) *formula* ⇒ *nat* ⇒ '*a formula*
where *wpow-0* : (*wpower-d F 0*) = *LIFT*(*empty*)
| *wpow-Suc*: (*wpower-d F* (*Suc n*)) = *LIFT*((*F*);(*wpower-d F n*))

syntax

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

syntax (*ASCII*)
-wpower-d :: [*lift*,*nat*] ⇒ *lift* ((*wpower* - -) [*88*,*88*] *87*)

translations
-wpower-d ⇌ *CONST wpower-d*

definition *fpower-d* :: (*'a*::*world*) *formula* ⇒ *nat* ⇒ *'a formula*
where
fpower-d F n ≡ *LIFT*(*wpower* (*F* ∧ *finite*) *n*)

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

syntax (*ASCII*)
-fpower-d :: [*lift*,*nat*] ⇒ *lift* ((*fpower* - -) [*88*,*88*] *87*)

translations
-fpower-d ⇌ *CONST fpower-d*

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

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

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

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

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

translations
-len-d ⇌ *CONST len-d*
-wpowerstar-d ⇌ *CONST wpowerstar-d*
-fpowerstar-d ⇌ *CONST fpowerstar-d*

definition *powerstar-d* :: ($'a$::*world*) *formula* $\Rightarrow$ $'a$ *formula*
where *powerstar-d F* $\equiv$ *LIFT*(*fpowerstar F*;(*empty* $\vee$ (*F* $\wedge$ *inf*)))

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

syntax (*ASCII*)
-powerstar-d :: *lift* $\Rightarrow$ *lift* ((*powerstar* -) [*85*] *85*)

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

definition *chopstar-d* :: ($'a$::*world*) *formula* $\Rightarrow$ $'a$ *formula*
where *chopstar-d F* $\equiv$ *LIFT*(*powerstar* (*F* $\wedge$ *more*))

definition *schopstar-d* :: ($'a$::*world*) *formula* $\Rightarrow$ $'a$ *formula*
where *schopstar-d F* $\equiv$ *LIFT*(*fpowerstar* (*F* $\wedge$ *more*))

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

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

translations
-chopstar-d $\rightleftharpoons$ *CONST chopstar-d*
-schopstar-d $\rightleftharpoons$ *CONST schopstar-d*

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

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

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

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

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

where *swhile-d F G* ≡ *LIFT*(*schopstar*(*F* ∧ *G*) ∧ (*sfin* ((¬*F*))))

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

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

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

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

definition *srepeat-d* :: ('*a*::*world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where *srepeat-d F G* ≡ *LIFT*(*F*⌢*swhile* (¬ *G*) *do F*)

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

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

translations
-*srepeat-d* ⇌ *CONST srepeat-d*

definition *aschopstar-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where *aschopstar-d F* ≡
λ*s*. (∃ (*l*:: *nat nellist*).
(*nnth l 0*) = *0* ∧ *nfinite l* ∧ *nidx l* ∧
(*enat* (*nlast l*)) = (*nlength s*) ∧ *nfinite s* ∧
(∀ (*i*::*nat*) . ((*enat i*)< (*nlength l*)) ⟶
((*nsubn s* (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *F*))
)

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

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

translations
-*aschopstar-d* ⇌ *CONST aschopstar-d*

lemma *FChopSem-var* [*mono*]:
(*w* ⊨ *f*;*g*) =
(((∃ *n*. *enat n* ≤ *nlength w* ∧ *f* ((*ntaken n w*)) ∧ *g* (*ndropn n w*)) ∨

$(\neg$ *nfinite* $w \wedge f\ w))$)
by (*simp add*: *itl-defs*)

inductive *istar-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *'a formula*
for F **where**
$(s \models$ *empty*$) \Longrightarrow (s \models$ (*istar-d* F))
| $(s \models F$;(*istar-d* F)) $\Longrightarrow$ $(s \models$ (*istar-d* F))

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

syntax (*ASCII*)
-istar-d :: *lift* $\Rightarrow$ *lift* ((*istar* -) [*85*] *85*)

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

8.2 Semantic lemmas

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

lemma *SChopExist*:
$\vdash$ ($\exists$ *k*. $f \frown g\ k$) = $f \frown$($\exists$ *k*. *g k*)
by (*auto simp add*: *itl-defs Valid-def*)

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

lemma *ExistSChop*:
$\vdash$ ($\exists$ *k*. (*g k*)$\frown$*f*) = ($\exists$ *k*. *g k*)$\frown$*f*
by (*auto simp add*: *itl-defs Valid-def*)

lemma *wpowersem1*:
$(\sigma \models$ ($\exists$ *k*. *wpower f k*) = (*empty* $\vee$ ($\exists$ *k*. *wpower f* (*Suc k*))))
proof (*auto*)
show $\bigwedge x.\ \sigma \models$ (*wpower f x*) $\Longrightarrow \forall k.\ \neg$ $(\sigma \models f$;*wpower f k*) $\Longrightarrow \sigma \models$ *empty*
by (*metis not0-implies-Suc wpow-0 wpow-Suc*)
show $\sigma \models$ *empty* $\Longrightarrow \exists x.\ \sigma \models$ (*wpower f x*)
by (*metis wpow-0*)
show $\bigwedge k.\ \sigma \models$ (*f*;*wpower f k*) $\Longrightarrow \exists x.\ \sigma \models$ (*wpower f x*)
by (*metis wpow-Suc*)
qed

lemma *fpowersem1*:
$(\sigma \models$ ($\exists$ *k*. *fpower f k*) = (*empty* $\vee$ ($\exists$ *k*. *fpower f* (*Suc k*))))

unfolding *fpower-d-def* **using** *wpowersem1*[*of LIFT* (*f* ∧ *finite*) σ]
by *blast*

lemma *wpowersem*:
⊢ (∃ *k*. *wpower f k*) = (*empty* ∨ *f*;(∃ *k*. (*wpower f k*)))
proof −
have *1*: ⊢ (∃ *k*. *wpower f k*) = (*empty* ∨ (∃ *k*. *wpower f* (*Suc k*)))
using *wpowersem1* **by** *blast*
have *2*: ⊢ (∃ *k*. *wpower f* (*Suc k*)) = (∃ *k*. *f*;*wpower f k*)
by *simp*
have *3*: ⊢ (∃ *k*. *f*;(*wpower f k*)) = *f*;(∃ *k*. (*wpower f k*))
using *ChopExist* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *fpowersem*:
⊢ (∃ *k*. *fpower f k*) = (*empty* ∨ (*f* ∧ *finite*);(∃ *k*. (*fpower f k*)))
unfolding *fpower-d-def* **using** *wpowersem*[*of LIFT* (*f* ∧ *finite*)]
by *blast*

lemma *finite-nidx-bounded-nlast*:
assumes *nfinite l*
nidx l
(*enat* (*nlast l*)) = (*nlength s*)
(*enat i*)≤ (*nlength l*)
shows (*nnth l i*) ≤ *nlength s*
using *assms*
by (*metis enat-ord-simps*(*1*) *nfinite-nlength-enat nidx-less-eq nnth-nlast*
the-enat.simps verit-comp-simplify1(*2*))

lemma *aschopstar-wpower-chain-a*:
assumes (∃ (*l*:: *nat nellist*).
(*nlength l*) = (*Suc n*) ∧ *nfinite l* ∧ *nidx l* ∧ (*nnth l 0*) = *0* ∧
(*enat* (*nlast l*)) = (*nlength* σ) ∧ *nfinite* σ ∧
(∀ (*i*::*nat*). *i*< (*nlength l*) ⟶
((*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *f*)
)
)
shows (∃ *k*. *0* ≤*k* ∧ *k* ≤ *nlength* σ ∧ *0*< *k* ∧ (*nsubn* σ *0 k* ⊨ *f*) ∧
(∃ *ls*. *nfinite ls* ∧ (*nlength ls*) = *n* ∧ *nidx ls* ∧ (*nnth ls 0*) = *0* ∧
(*enat* (*nlast ls*)) = (*nlength* (*ndropn k* σ)) ∧ *nfinite*(*ndropn k* σ) ∧
(∀ (*i*::*nat*). *i*< (*nlength ls*) ⟶
((*nsubn* (*ndropn k* σ) (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f*)
))
)
proof −

obtain *l* **where** *1*: (*nlength l*) = (*Suc n*) ∧ *nfinite l* ∧ *nidx l* ∧ (*nnth l 0*) = *0* ∧
(*enat* (*nlast l*)) = (*nlength σ*) ∧ *nfinite σ* ∧
(∀ (*i*::*nat*). *i*< (*nlength l*) ⟶
((*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *f*)
)
using *assms* **by** *auto*
have *2*: *nlength l* >*0*
using *1* **by** (*simp add*: *enat-0-iff*(*1*))
have *3*: *l* = *NCons* (*nnth l 0*) (*ndropn 1 l*)
using *2* **by** (*simp add*: *ndropn-Suc-conv-ndropn zero-enat-def*)
have *4*: *nlength* (*ndropn 1 l*) = *n*
by (*simp add*: *1*)
have *5*: *0*≤(*nnth l 0*)
by *simp*
have *6*: (*nnth l 0*) ≤ *nlength σ*
using *1* **using** *i0-lb zero-enat-def* **by** *presburger*
have *7*: (*nnth l 0*) = *0*
using *1* **by** *blast*
have *71*: (*nnth* (*ndropn 1 l*) *0*) = (*nnth l 1*)
by *auto*
have *8*: *nnth l 0* < *nnth* (*ndropn 1 l*) *0*
by (*metis 1 71 One-nat-def eSuc-enat enat-ord-simps*(*2*) *ileI1 nidx-gr-first zero-less-Suc*)
have *9*: *nidx* (*ndropn 1 l*)
by (*metis 1 2 One-nat-def Suc-eq-plus1 Suc-ile-eq enat-min-eq ndropn-nlength ndropn-nnth nidx-expand plus-1-eq-Suc plus-enat-simps*(*1*) *zero-enat-def*)
have *10*: (*enat* (*nlast* (*ndropn 1 l*))) = (*nlength σ*)
using *1 3* **by** (*metis nlast-NCons*)
have *101*: ⋀*j*. *j* ≤ *nlength* (*ndropn 1 l*) ⟶
nnth (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*)) *j* =
(*nnth l* (*Suc j*)) − (*nnth l 1*)
by *simp*
have *102*: ⋀*j*. *j* ≤ *nlength* (*ndropn 1 l*) ⟶
(*nnth l 1*) ≤ (*nnth l* (*Suc j*))
by (*simp add*: *1 nidx-less-eq*)
have *103*: ⋀*j*. *j*< *nlength* (*ndropn 1 l*) ⟶
(*nnth l* (*Suc j*)) − (*nnth l 1*) <
(*nnth l* (*Suc* (*Suc j*))) − (*nnth l 1*)
using *1 nidx-expand*[*of l*]
by (*metis 102 4 diff-less-mono eSuc-enat ileI1 iless-Suc-eq order-less-imp-le*)
have *11*: *nidx* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*))
using *103 101 nidx-expand*[*of* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*))]
using *Suc-ile-eq* **by** *force*
have *12*: *nlength* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*)) = *n*
using *4* **by** *auto*
have *13*: (*enat* (*nlast* (*nmap* (λ*x*. *x*− (*nnth l 1*)) (*ndropn 1 l*))))
=
(*nlength* (*ndropn* (*nnth l 1*) *σ*))
by (*metis 1 10 idiff-enat-enat ndropn-nlength nfinite-ndropn nlast-nmap*)
have *14*: (*nsubn σ 0* (*nnth l 1*) ⊨ *f*)
by (*metis 1 One-nat-def enat-ord-simps*(*2*) *zero-less-Suc*)

```
 have 15: (∀ (i::nat). i < nlength (ndropn 1 l) ⟶
            (nsubn σ (nnth (ndropn 1 l) i) (nnth (ndropn 1 l) (Suc i)) ⊨ f))
   using 1 3 eSuc-enat ileI1 iless-Suc-eq ndropn-nnth nlength-NCons plus-1-eq-Suc by metis

 have 16: (∀ (i::nat). i< nlength (nmap (λx. x− (nnth l 1)) (ndropn 1 l)) ⟶
          (nsubn σ ((nnth (nmap (λx. x− (nnth l 1)) (ndropn 1 l)) i) + (nnth l 1))
                  ((nnth (nmap (λx. x− (nnth l 1)) (ndropn 1 l)) (Suc i)) + (nnth l 1))⊨ f))
   using 102 12 15 4 by force
 have 17: (∀ (i::nat). i< nlength (nmap (λx. x− (nnth l 1)) (ndropn 1 l)) ⟶
             ((nsubn (ndropn (nnth l 1) σ)
                    (nnth (nmap (λx. x− (nnth l 1)) (ndropn 1 l)) i)
                    (nnth (nmap (λx. x− (nnth l 1)) (ndropn 1 l)) (Suc i))) ⊨ f))
   by (metis 11 16 eSuc-enat ileI1 nidx-expand nsubn-ndropn)
 have 18: nfinite (nmap (λx. x− (nnth l 1)) (ndropn 1 l))
    using 12 nlength-eq-enat-nfiniteD by blast
 have 19: nfinite (ndropn (nnth l 1) σ)
    using 1 nfinite-ndropn-a by blast
 have 20: (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
                  (enat (nlast ls)) = (nlength (ndropn (nnth l 1) σ)) ∧ nfinite(ndropn (nnth l 1) σ) ∧
              (∀(i::nat). i< (nlength ls) ⟶
                       ((nsubn (ndropn (nnth l 1) σ) (nnth ls i) (nnth ls (Suc i)) ) ⊨ f)
                ))
  by (metis 11 12 13 17 18 19 71 diff-self-eq-0 enat-le-plus-same(1) gen-nlength-def
      nlength-code nnth-nmap)
 show ?thesis
 by (metis 1 2 20 71 8 One-nat-def Suc-ile-eq finite-nidx-bounded-nlast zero-enat-def
     zero-order(1))
qed

lemma aschopstar-wpower-chain-b:
assumes (∃ k. 0 ≤k ∧ k ≤ nlength σ ∧ 0< k ∧
            (nsubn σ 0 k ⊨ f) ∧
            (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
                  (enat (nlast ls)) = (nlength (ndropn k σ)) ∧ nfinite(ndropn k σ) ∧
              (∀(i::nat). i< (nlength ls) ⟶
                       ((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i)) ) ⊨ f)
                ))
        )
shows (∃ (l:: nat nellist).
              (nlength l) = (Suc n) ∧ nfinite l ∧ nidx l ∧ (nnth l 0) = 0 ∧
               (enat (nlast l)) = (nlength σ) ∧ nfinite σ ∧
                      (∀(i::nat). i< (nlength l) ⟶
                         ((nsubn σ (nnth l i) (nnth l (Suc i))) ⊨ f)
                      )
             )
proof −
 obtain k where 1: 0 ≤k ∧ k ≤ nlength σ ∧ k > 0 ∧
            (nsubn σ 0 k ⊨ f) ∧
            (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
                  (enat (nlast ls)) = (nlength (ndropn k σ)) ∧ nfinite(ndropn k σ) ∧
```

(∀ (i::nat). i< (nlength ls) ⟶
((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i))) ⊨ f)
))
using *assms* **by** *auto*
have *2*: (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
(enat (nlast ls)) = (nlength (ndropn k σ)) ∧ nfinite(ndropn k σ) ∧
(∀ (i::nat). i< (nlength ls) ⟶
((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i))) ⊨ f)
))
using *1* **by** *auto*
obtain *ls* **where** *3*: nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
(enat (nlast ls)) = (nlength (ndropn k σ)) ∧ nfinite(ndropn k σ) ∧
(∀ (i::nat). i< (nlength ls) ⟶
((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i))) ⊨ f)
)
using *2* **by** *auto*
have *4*: nidx (nmap (λx. x+ k) ls)
using *3*
by (*simp add*: *nidx-expand*)
have *41*: ⋀j. j ≤ nlength (nmap (λx. x+ k) ls) ⟶
0 < (nnth (nmap (λx. x+ k) ls) j)
by (*simp add*: *1*)
have *5*: nidx (NCons 0 (nmap (λx. x+ k) ls))

using *3 4 nidx-expand*[*of ls*] *nidx-expand*[*of* (nmap (λx. x+ k) ls)]
nidx-expand[*of* (NCons 0 (nmap (λx. x+ k) ls))]
by (*metis* (*no-types*, *lifting*) *1 Suc-ile-eq diff-zero iless-Suc-eq le-add-diff-inverse lessI less-Suc-eq-0-disj nlength-NCons nlength-nmap nnth-0 nnth-Suc-NCons nnth-nmap*)
have *6*: (nlength ((NCons 0 (nmap (λx. x+ k) ls)))) = (Suc n)
by (*simp add*: *3 eSuc-enat*)
have *7*: (enat (nlast ((NCons 0 (nmap (λx. x+ k) ls))))) = (nlength σ)
by (*metis 1 3 add.commute enat.distinct*(*2*) *enat-add-sub-same less-eqE ndropn-nlength nlast-NCons nlast-nmap plus-enat-simps*(*1*))
have *8*: (nsubn σ 0 k ⊨ f)
using *1* **by** *auto*

have *9*: (∀ (i::nat). i< (nlength ls) ⟶
((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i))) ⊨ f)
)
using *3* **by** *auto*
have *10*: (∀ (i::nat). i< (nlength ls) ⟶
((nsubn σ ((nnth ls (i))+k) ((nnth ls ((i)+1))+k)) ⊨ f)
)
by (*metis 3 Suc-ile-eq add.commute add.right-neutral nidx-less nsubn-ndropn plus-1-eq-Suc*)

have *11*: (∀ (i::nat). i < nlength ((((nmap (λx. x+ k) ls)))) ⟶
(nsubn σ (nnth ((((nmap (λx. x+ k) ls)))) i) (nnth ((((nmap (λx. x+ k) ls)))) (Suc i)) ⊨ f))
by (*metis 10 add.commute eSuc-enat ileI1 nlength-nmap nnth-nmap order-less-imp-le plus-1-eq-Suc*)
have *12*: (∀ i. (0<i ∧ i< 1+ (nlength (nmap (λx. x+ k) ls))) ⟶

((*nsubn* σ ((*nnth* (*nmap* (λ*x*. *x*+ *k*) *ls*) (*i*−*1*)))
((*nnth* (*nmap* (λ*x*. *x*+ *k*) *ls*) ((*i*))))) ⊨ *f*)
)
using *11*
by (*metis 3 Suc-diff-1 Suc-ile-eq eSuc-enat iless-Suc-eq nlength-nmap one-enat-def plus-1-eq-Suc plus-enat-simps*(*1*))
have *13*: (∀ (*i*::*nat*). *i* < *nlength* ((*NCons 0* ((*nmap* (λ*x*. *x*+ *k*) *ls*)))) ⟶
(*nsubn* σ (*nnth* (((*NCons 0* (*nmap* (λ*x*. *x*+ *k*) *ls*)))) *i*)
(*nnth* (((*NCons 0* (*nmap* (λ*x*. *x*+ *k*) *ls*)))) (*Suc i*)) ⊨ *f*))
using *12*
using *1 11 3 6 less-Suc-eq-0-disj* **by** *auto*

have *14*: (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *k*) *ls*)) *0*) = *0*
by *simp*
have *15*: (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *k*) *ls*)) (*1*)) = *k*
using *4* **by** (*simp add*: *3*)
have *16*: (*nsubn* σ (*nnth* (*NCons 0* (*nmap* (λ*x*. *x*+ *k*) *ls*)) *0*) *k* ⊨ *f*)
by (*simp add*: *8*)
have *17*: *nfinite* (*NCons 0* (*nmap* (λ*x*. *x*+ *k*) *ls*))
using *6 nlength-eq-enat-nfiniteD* **by** *blast*
show *?thesis*
by (*metis 13 3 5 6 7 nfinite-NCons nfinite-ndropn nfinite-nmap nnth-0*)
qed

lemma *chop-wpower-eqv-sem*:
((σ ⊨(∃ *n*. (*wpower* ((*f* ∧ *more*) ∧ *finite*) *n*)))) =
((σ ⊨ *empty*) ∨ ((σ ⊨ ((*f* ∧ *more*) ∧ *finite*);(∃ *n*. (*wpower* ((*f* ∧ *more*) ∧ *finite*) *n*)))))
using *wpowersem* **by** *fastforce*

lemma *aschopstar-eqv-wpower-chop-help*:
(σ ⊨ *wpower* ((*f* ∧ *more*) ∧ *finite*) *n*) =
(∃ (*l*:: *nat nellist*).
(*nlength l*) = (*n*) ∧ *nfinite l* ∧ *nidx l* ∧ (*nnth l 0*) = *0* ∧
(*enat* (*nlast l*)) = (*nlength* σ) ∧ *nfinite* σ ∧
(∀(*i*::*nat*). *i*< (*nlength l*) ⟶
((*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *f*)
)
)

proof
(*induct n arbitrary*: σ)
case *0*
then show *?case*
by (*auto simp add*: *empty-defs nidx-expand nnth-nlast zero-enat-def*)
(*metis eSuc-enat enat-0-iff*(*1*) *iless-Suc-eq leD nfinite-conv-nlength-enat nlast-NNil nlength-NNil nnth-NNil zero-le*)
next
case (*Suc n*)
then show *?case*
proof −

have ($\sigma \models$ *wpower* ((*f* $\wedge$ *more*) $\wedge$ *finite*) (*Suc n*)) =
($\sigma \models$ (((*f* $\wedge$ *more*) $\wedge$ *finite*);(*wpower* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *n*)))
by *simp*
also have ... =
($\exists$ *k*. *0* $\leq$*k* $\wedge$ *k* $\leq$ *nlength* (σ) $\wedge$ *k* > *0* $\wedge$
(*ntaken k* (σ) $\models$ *f*) $\wedge$
(*ndropn k* (σ) $\models$ *wpower* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *n*)
)

using *enat-0-iff*(*1*) *le-zero-eq* **by** (*auto simp add*: *more-defs chop-defs finite-defs*)
also have ... =
($\exists$ *k*. *0* $\leq$*k* $\wedge$ *k* $\leq$*nlength* (σ) $\wedge$ *k* > *0* $\wedge$
(*nsubn* σ *0 k* $\models$ *f*) $\wedge$
(*ndropn* *k* (σ) $\models$ *wpower* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *n*)
)
by (*metis One-nat-def Suc-diff-1 Suc-diff-Suc diff-add ndropn-0 ntaken-ndropn*)
also have ... =
($\exists$ *k*. *0* $\leq$*k* $\wedge$ *k* $\leq$ *nlength* (σ) $\wedge$ *k* > *0* $\wedge$
(*nsubn* σ *0 k* $\models$ *f*) $\wedge$
($\exists$ *l*. *nfinite l* $\wedge$ (*nlength l*) = *n* $\wedge$ *nidx l* $\wedge$ (*nnth l 0*) = *0* $\wedge$
(*enat* (*nlast l*)) = (*nlength* (*ndropn k* σ)) $\wedge$ *nfinite*(*ndropn k* σ) $\wedge$
($\forall$ (*i*::*nat*). *i*< (*nlength l*) $\longrightarrow$
((*nsubn* (*ndropn k* σ) (*nnth l i*) (*nnth l* (*Suc i*))) $\models$ *f*)
))
)
using *Suc.hyps* **by** *blast*
also have ... =
($\exists$ (*l*:: *nat nellist*).
(*nlength l*) = (*Suc n*) $\wedge$ *nfinite l* $\wedge$ *nidx l* $\wedge$ (*nnth l 0*) = *0* $\wedge$
(*enat* (*nlast l*)) = (*nlength* σ) $\wedge$ *nfinite* σ $\wedge$
($\forall$ (*i*::*nat*). *i*< (*nlength l*) $\longrightarrow$
((*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) $\models$ *f*)
)
)

using *aschopstar-wpower-chain-a*[*of n* σ *f*]
aschopstar-wpower-chain-b [*of* σ *f n*] **by** *auto*
finally show ($\sigma \models$ *wpower* ((*f* $\wedge$ *more*) $\wedge$ *finite*) (*Suc n*)) =
($\exists$ (*l*:: *nat nellist*).
(*nlength l*) = (*Suc n*) $\wedge$ *nfinite l* $\wedge$ *nidx l* $\wedge$ (*nnth l 0*) = *0* $\wedge$
(*enat* (*nlast l*)) = (*nlength* σ) $\wedge$ *nfinite* σ $\wedge$

($\forall$ (*i*::*nat*). *i*< (*nlength l*) $\longrightarrow$
((*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) $\models$ *f*)
)
) .
qed
qed

lemma *aschopstar-eqv-power-chop*:

$(\sigma \models \textit{aschopstar } f) = (\ (\sigma \models (\exists \ k. \textit{ wpower } ((f \wedge \textit{more}) \wedge \textit{finite}) \ k)))$
using *nfinite-conv-nlength-enat* **by** (*simp add*: *aschopstar-d-def aschopstar-eqv-wpower-chop-help*)
blast

lemma *ASChopstarEqvSem*:
$(\sigma \models (\textit{aschopstar } f = \ (\textit{empty} \vee \ ((f \wedge \textit{more}) \wedge \textit{finite}); (\textit{aschopstar } f))) \ )$
proof −
have *1*: $(\sigma \models \textit{aschopstar } f) = (\ (\sigma \models (\exists \ k. \textit{ wpower } ((f \wedge \textit{more}) \wedge \textit{finite}) \ k)))$
using *aschopstar-eqv-power-chop* **by** *simp*
have *2*: $(\ (\sigma \models (\exists \ k. \textit{ wpower } ((f \wedge \textit{more}) \wedge \textit{finite}) \ k))) =$
$((\sigma \models \textit{empty}) \vee (\sigma \models ((f \wedge \textit{more}) \wedge \textit{finite});(\exists n. \textit{ wpower } ((f \wedge \textit{more}) \wedge \textit{finite}) \ n)))$
using *chop-wpower-eqv-sem* **by** *simp*
have *3*: $(\sigma \models ((f \wedge \textit{more}) \wedge \textit{finite});(\exists n. \textit{ wpower } ((f \wedge \textit{more}) \wedge \textit{finite}) \ n)) =$
$(\exists n. \ n \leq \textit{nlength } \sigma \wedge ((\textit{ntaken } n \ \sigma) \models (f \ \wedge \textit{more}) \wedge \textit{finite}) \wedge$
$((\textit{ndropn } n \ \sigma) \models (\exists x. (\textit{wpower } ((f \wedge \textit{more}) \wedge \textit{finite}) \ x) \ )))$
by (*simp add*: *chop-defs finite-defs*) *blast*
have *4*: $(\exists n. \ n \leq \textit{nlength } \sigma \wedge ((\textit{ntaken } n \ \sigma) \models (f \ \wedge \textit{more}) \wedge \textit{finite}) \wedge$
$((\textit{ndropn } n \ \sigma) \models (\exists x. (\textit{wpower } ((f \wedge \textit{more}) \wedge \textit{finite}) \ x) \ ))) =$
$(\exists n. \ n \leq \textit{nlength } \sigma \wedge ((\textit{ntaken } n \ \sigma) \models (f \ \wedge \textit{more}) \wedge \textit{finite}) \wedge ((\textit{ndropn } n \ \sigma) \models \ \textit{aschopstar } f \ ))$
by (*simp add*: *aschopstar-eqv-power-chop*)
have *5*: $(\exists n. \ n \leq \textit{nlength } \sigma \wedge ((\textit{ntaken } n \ \sigma) \models (f \ \wedge \textit{more}) \wedge \textit{finite}) \wedge ((\textit{ndropn } n \ \sigma) \models \ \textit{aschopstar } f \ )) =$
$(\sigma \models \ ((f \wedge \textit{more}) \wedge \textit{finite}); (\textit{aschopstar } f) \ )$
by (*simp add*: *chop-defs finite-defs*) *blast*
have *6*: $((\sigma \models \textit{empty}) \vee (\sigma \models ((f \wedge \textit{more}) \wedge \textit{finite});(\exists n. \textit{ wpower } ((f \wedge \textit{more}) \wedge \textit{finite}) \ n))) =$
$(\sigma \models (\textit{empty} \vee \ ((f \wedge \textit{more}) \wedge \textit{finite}); (\textit{aschopstar } f)))$
using *3 4 5* **by** *auto*
show *?thesis* **using** *1 2 6* **by** *auto*
qed

lemma *ASChopstarEqvSChopstar*:
$\vdash (\textit{aschopstar } f) = (\textit{schopstar } f)$
by (*simp add*: *Valid-def schopstar-d-def aschopstar-eqv-power-chop fpowerstar-d-def fpower-d-def*)

lemma *len-defs* :
$(w \models \textit{len } n) = (\textit{nlength } w = n)$
proof
(*simp add*: *len-d-def*)
show $(w \models (\textit{wpower skip } n)) = (\textit{nlength } w = n)$
proof (*induct n arbitrary*:*w*)
case *0*
then show *?case* **by** (*simp add*: *empty-defs zero-enat-def*)
next
case (*Suc n*)
then show *?case*
by (*auto simp add*: *min-def len-d-def empty-defs chop-defs skip-defs finite-defs nlength-eq-enat-nfiniteD*)
(*metis One-nat-def enat.distinct*(*2*) *enat-add-sub-same le-iff-add plus-1-eq-Suc plus-enat-simps*(*1*))
qed
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 (*auto simp add*: *Valid-def itl-defs*)

lemma *PowerstarEqvSemhelp3*:
$\vdash$ ((*f* $\wedge$ *inf*);*g* $\vee$ (*f* $\wedge$ *finite*);*g*) = (*f* ;*g*)
by (*auto simp add*: *Valid-def itl-defs*)

lemma *WPowerstarEqvSem*:
(σ $\models$ (*wpowerstar f*) = (*empty* $\vee$ *f*;(*wpowerstar f*)))
by (*metis intD wpowersem wpowerstar-d-def*)

lemma *FPowerstarEqvSem*:
(σ $\models$ (*fpowerstar f*) = (*empty* $\vee$ (*f* $\wedge$ *finite*);(*fpowerstar f*)))
by (*metis fpowersem fpowerstar-d-def intD*)

lemma *PowerstarEqvSem*:
(σ $\models$ (*powerstar f*) = (*empty* $\vee$ *f*;(*powerstar f*)))
proof –
have *1*: (σ $\models$ (*powerstar f*)) =
(σ $\models$ ($\exists$ *k*. *fpower f k*);(*empty* $\vee$ *f* $\wedge$ *inf*))
by (*simp add*: *powerstar-d-def fpowerstar-d-def*)
have *2*: (σ $\models$ ($\exists$ *k*. *fpower f k*);(*empty* $\vee$ *f* $\wedge$ *inf*)) =
(σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *finite*);($\exists$ *k*. (*fpower f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*))
using *fpowersem* **by** (*metis inteq-reflection*)
have *3*: (σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *finite*);($\exists$ *k*. (*fpower f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*)) =
(σ $\models$ *empty*;(*empty* $\vee$ (*f* $\wedge$ *inf*)) $\vee$
((*f* $\wedge$ *finite*);($\exists$ *k*. (*fpower f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*))
by (*metis OrChopEqv inteq-reflection*)
have *4*: (σ $\models$ *empty*;(*empty* $\vee$ (*f* $\wedge$ *inf*)) $\vee$
((*f* $\wedge$ *finite*);($\exists$ *k*. (*fpower f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*)) =
(σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *inf*) $\vee$ ((*f* $\wedge$ *finite*);($\exists$ *k*. (*fpower 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*. (*fpower f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*))) =
(σ $\models$ (*empty* $\vee$ (*f* $\wedge$ *inf*);(($\exists$ *k*. (*fpower f k*));(*empty* $\vee$ *f* $\wedge$ *inf*)) $\vee$
((*f* $\wedge$ *finite*);($\exists$ *k*. (*fpower f k*)));(*empty* $\vee$ *f* $\wedge$ *inf*)))
using *PowerstarEqvSemhelp2*

by (*metis* (*mono-tags*, *lifting*) *inteq-reflection*)
have *51*: ($\sigma \models ((f \wedge \mathit{finite});(\exists\ k.\ (\mathit{fpower}\ f\ k)));(\mathit{empty} \vee f \wedge \mathit{inf}) =$
$(f \wedge \mathit{finite});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})))$
by (*auto simp add*: *ChopAssocSemHelp1*)
have *6*: ($\sigma \models (\mathit{empty} \vee (f \wedge \mathit{inf});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})) \vee$
$((f \wedge \mathit{finite});(\exists\ k.\ (\mathit{fpower}\ f\ k)));(\mathit{empty} \vee f \wedge \mathit{inf}))\ ) =$
$(\sigma \models (\mathit{empty} \vee (f \wedge \mathit{inf});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})) \vee$
$(f \wedge \mathit{finite});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf}))))$

using *51* **by** *auto*
have *7* : ($\sigma \models (\mathit{empty} \vee (f \wedge \mathit{inf});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})) \vee$
$(f \wedge \mathit{finite});((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})))) =$
$(\sigma \models (\mathit{empty} \vee f;((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf}))))$
using *PowerstarEqvSemhelp3* **by** *fastforce*
have *8*: ($\sigma \models (\mathit{empty} \vee f;((\exists\ k.\ (\mathit{fpower}\ f\ k));(\mathit{empty} \vee f \wedge \mathit{inf})))) =$
$(\sigma \models (\mathit{empty} \vee f;(\mathit{powerstar}\ f)))$
by (*simp add*: *powerstar-d-def fpowerstar-d-def*)
from *1 2 3 4 5 6 7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *wpowerchopsem*:
$\vdash (\exists\ k.\ \mathit{wpower}\ (f \wedge \mathit{more})\ k) =$
$(\ \mathit{empty} \vee (f \wedge \mathit{more});(\exists\ k.\ (\mathit{wpower}\ (f \wedge \mathit{more})\ k))$
$)$
by (*simp add*: *wpowersem*)

lemma *powerchopsem*:
$\vdash (\exists\ k.\ \mathit{fpower}\ (f \wedge \mathit{more})\ k) =$
$(\ \mathit{empty} \vee ((f \wedge \mathit{more}) \wedge \mathit{finite});(\exists\ k.\ (\mathit{fpower}\ (f \wedge \mathit{more})\ k))$
$)$
using *fpowersem* **by** *auto*

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

lemma *SChopstarEqvSem*:
$(\sigma \models (\mathit{schopstar}\ f) = (\mathit{empty} \vee (f \wedge \mathit{more}) \frown (\mathit{schopstar}\ f)\ ))$
by (*metis FPowerstarEqvSem schop-d-def schopstar-d-def*)

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

lemma *IStarIntros*:
$\vdash \mathit{empty} \vee f;(\mathit{istar}\ f) \longrightarrow (\mathit{istar}\ f)$
unfolding *Valid-def* **using** *istar-d.intros* **by** *fastforce*

lemma *IStarCases*:
$(w \models (istar\ F)) \implies$
$((w \models empty \lor (F;istar\ F)) \implies P) \implies P$
using *istar-d.cases*[*of F w P*] **by** *auto*

lemma *IStarEqvIStarSem*:
$(w \models (istar\ f)) = (w \models empty \lor f;(istar\ f))$
using *IStarCases*[*of f*]
by (*metis istar-d.intros*(*1*) *istar-d.intros*(*2*) *unl-lift2*)

lemma *IStarEqvIStar*:
$\vdash (istar\ f) = (empty \lor f;(istar\ f))$
using *IStarEqvIStarSem*[*of f*] **unfolding** *Valid-def* **by** *auto*

lemma *IStarInductSem*:
assumes $(\bigwedge s.\ (s \models empty\ ) \implies P\ s)$
$(\bigwedge s.\ (s \models (F;((istar\ F)\ \land P))) \implies P\ s)$
shows $(w \models (istar\ F) \longrightarrow P)$
using *assms istar-d.induct*[*of F w P*]
chop-defs[*of F LIFT* ((*istar F*) $\land$ *P*)]
unfolding *chop-d-def* **by** (*metis intensional-rews*(*3*))

lemma *IStarInduct*:
assumes $\vdash empty \lor f;((istar\ f) \land g) \longrightarrow g$
shows $\vdash (istar\ f) \longrightarrow g$
using *assms IStarInductSem*[*of g*] **unfolding** *Valid-def* **by** (*metis intensional-rews*(*3*))

lemma *IStarWeakInductSem*:
assumes $(\bigwedge s.\ (s \models empty\ ) \implies P\ s)$
$(\bigwedge s.\ (s \models (F;P)) \implies P\ s)$
shows $(w \models (istar\ F) \longrightarrow P)$
using *assms istar-d.induct*[*of F w P*] **using** *chop-defs*[*of F P*]
chop-defs[*of F LIFT* ((*istar F*) $\land$ *P*)] **unfolding** *chop-d-def*
by (*metis intensional-rews*(*3*))

lemma *IStarWeakInduct*:
assumes $\vdash empty \lor f;\ g \longrightarrow g$
shows $\vdash (istar\ f) \longrightarrow g$
using *assms IStarWeakInductSem*[*of g*] **unfolding** *Valid-def*
by (*metis intensional-rews*(*3*))

8.3 Helper lemmas

lemma *AndEmptyChopAndEmptyEqvAndEmpty*:
$\vdash (f \land empty);(f \land empty) = (f \land empty)$
proof –
have *1*: $\vdash (f \land empty);(f \land empty) \longrightarrow (f \land empty)$

by (*metis ChopAndB ChopEmpty int-eq*)
have *2*: ⊢ (*f* ∧ *empty*) ⟶ (*f* ∧ *empty*);(*f* ∧ *empty*)
by (*auto simp add*: *Valid-def itl-defs zero-enat-def*)
(*metis iless-Suc-eq less-numeral-extra*(*1*) *ntake-0 ntake-all one-eSuc zero-enat-def*)
show *?thesis*
by (*simp add*: *1 2 int-iffI*)
qed

8.4 Properties of Chopstar and Chopplus

lemma *FPowerstardef*:
⊢ *fpowerstar f* = (∃ *n*. *fpower f n*)
by (*simp add*: *fpowerstar-d-def*)

lemma *Powerstardef*:
⊢ *powerstar f* = (*fpowerstar f*);(*empty* ∨ (*f* ∧ *inf*))
by (*simp add*: *fpowerstar-d-def powerstar-d-def*)

lemma *Chopstardef*:
⊢ *chopstar f* = *powerstar* (*f* ∧ *more*)
by (*simp add*: *chopstar-d-def*)

lemma *SChopstardef*:
⊢ *schopstar f* = *fpowerstar* (*f* ∧ *more*)
by (*simp add*: *schopstar-d-def*)

lemma *WPowerEqRule*:
assumes ⊢ *f* = *g*
shows ⊢ *wpower f n* = *wpower g n*
using *assms*
by (*metis int-eq int-simps*(*20*))

lemma *WPowerCommute*:
⊢ (*f*) ;(*wpower f n*) = (*wpower f n*);(*f*)
proof
(*induct n*)
case *0*
then show *?case*
by (*metis ChopEmpty EmptyChop inteq-reflection wpow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis ChopAssoc inteq-reflection wpow-Suc*)
qed

lemma *FPowerCommute*:
⊢ (*f* ∧ *finite*) ;*fpower f n* = *fpower f n*;(*f* ∧ *finite*)
unfolding *fpower-d-def* **using** *WPowerCommute*[*of LIFT* (*f* ∧ *finite*)]

by *blast*

lemma *WPowerChopInductL*:
assumes $\vdash g \vee f;h \longrightarrow h$
shows $\vdash (wpower\ f\ n);g \longrightarrow h$
using *assms*
proof
(*induct n*)
case *0*
then show *?case* **using** *EmptyChop*
by (*metis MP Prop12 int-eq int-iffD1 int-simps*(*33*) *wpow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis ChopAssoc Prop05 Prop11 RightChopImpChop lift-imp-trans wpow-Suc*)
qed

lemma *FPowerChopInductL*:
assumes $\vdash g \vee f;h \longrightarrow h$
shows $\vdash (fpower\ f\ n);g \longrightarrow h$
unfolding *fpower-d-def* **using** *assms WPowerChopInductL*[*of g LIFT* (*f* ∧ *finite*) *h n*]
using *AndChopA* **by** *fastforce*

lemma *FPowerChopInductFiniteL*:
assumes $\vdash g \vee (f \wedge finite);h \longrightarrow h$
shows $\vdash (fpower\ f\ n);g \longrightarrow h$
unfolding *fpower-d-def* **using** *assms WPowerChopInductL*[*of g LIFT* (*f* ∧ *finite*) *h n*]
by *blast*

lemma *WPowerChopInductMoreL*:
assumes $\vdash g \vee (f \wedge more);h \longrightarrow h$
shows $\vdash (wpower\ f\ n);g \longrightarrow h$
using *assms*
proof
(*induct n*)
case *0*
then show *?case*
by (*metis WPowerChopInductL wpow-0*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: $\vdash wpower\ f\ (Suc\ n);g\ = (f;wpower\ f\ n);g$
by *simp*
have *2*: $\vdash (f;wpower\ f\ n);g = f;((wpower\ f\ n);g)$
by (*meson ChopAssoc Prop11*)
have *3*: $\vdash f;((wpower\ f\ n);g) \longrightarrow f;h$
using *RightChopImpChop Suc.hyps Suc.prems* **by** *blast*
have *31*: $\vdash f = ((f \wedge more) \vee (f \wedge empty))$
unfolding *empty-d-def* **by** *fastforce*

have *4*: $\vdash$ *f*;*h* = (((*f* $\wedge$ *more*)); *h* $\vee$ ((*f* $\wedge$ *empty*));*h*)
using *31 OrChopEqvRule* **by** *blast*
have *5*: $\vdash$ ((*f* $\wedge$ *more*)); *h* $\longrightarrow$ *h*
by (*metis Prop03 Prop10 assms inteq-reflection lift-imp-trans*)
have *6*: $\vdash$ ((*f* $\wedge$ *empty*)); *h* $\longrightarrow$ *h*
by (*metis AndChopB EmptyChop inteq-reflection*)
from *5 6 4 3 2 1* **show** *?thesis*
by (*metis Prop02 inteq-reflection lift-imp-trans*)
qed
qed

lemma *FPowerChopInductFiniteMoreL*:
assumes $\vdash$ *g* $\vee$ ((*f* $\wedge$ *finite*) $\wedge$ *more*);*h* $\longrightarrow$ *h*
shows $\vdash$ (*fpower f n*);*g* $\longrightarrow$ *h*
unfolding *fpower-d-def* **using** *assms*
WPowerChopInductMoreL[*of g LIFT* (*f* $\wedge$ *finite*) *h n*]
by *blast*

lemma *FPowerChopInductInfL*:
assumes $\vdash$ *g* $\vee$ *f*;*h* $\longrightarrow$ *h*
shows $\vdash$ ((*fpower f n*);(*f* $\wedge$ *inf*));*g* $\longrightarrow$ *h*
using *assms*
proof
(*induct n*)
case *0*
then show *?case*
by (*metis* (*no-types*, *lifting*) *AndInfChopEqvAndInf ChopAssoc FPowerChopInductFiniteL PowerstarE-qvSemhelp3*
Prop03 Prop10 Prop12 inteq-reflection)
next
case (*Suc n*)
then show *?case*
proof –
have $\vdash$(*f* $\wedge$ *finite*);(*fpower f n*;((*f* $\wedge$ *inf*);*g*)) = (*fpower f* (*Suc n*);(*f* $\wedge$ *inf*));*g*
by (*metis ChopAssoc fpower-d-def int-eq wpow-Suc*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *AndChopA ChopAndB ChopAssoc Prop03 Prop10 Suc assms int-eq lift-imp-trans*)

qed
qed

lemma *FChopInductInfMoreL*:
assumes $\vdash$ *g* $\vee$ *f*;*h* $\longrightarrow$ *h*
shows $\vdash$ ((*fpower f n*);((*f* $\wedge$ *more*) $\wedge$ *inf*));*g* $\longrightarrow$ *h*
using *FPowerChopInductInfL*
by (*metis AndMoreAndInfEqvAndInf assms inteq-reflection*)

lemma *WPowerChopInductR*:

assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;(*wpower f n*) $\longrightarrow$ *h*
using *assms*
proof
(*induct n*)
case *0*
then show *?case* **using** *ChopEmpty*
by (*metis MP Prop12 int-iffD2 int-simps*(*33*) *inteq-reflection wpow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis AndChopB ChopAssoc Prop03 Prop10 WPowerCommute inteq-reflection lift-imp-trans wpow-Suc*)
qed

lemma *FPowerChopInductR*:
assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;(*fpower f n*) $\longrightarrow$ *h*
unfolding *fpower-d-def* **using** *assms WPowerChopInductR*[*of g h LIFT* (*f* $\wedge$ *finite*) *n*]
using *ChopAndA* **by** *fastforce*

lemma *FpowerChopInductInfR*:
assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;((*fpower f n*);(*f* $\wedge$ *inf*)) $\longrightarrow$ *h*
using *assms*
by (*metis ChopAndA ChopAssoc FPowerChopInductR LeftChopImpChop Prop05 int-iffD1 lift-imp-trans*)

lemma *WPowerStarCommute*:
$\vdash$ *f*;($\exists$ *n*. *wpower f n*) = ($\exists$ *n*. *wpower f n*);*f*
proof –
have *1*: $\vdash$ *f* ;($\exists$ *n*. *wpower f n*) = ($\exists$ *n*. *f* ;*wpower f n*)
by (*metis ChopExist Prop11*)
have *2*: $\vdash$ ($\exists$ *n*. *f* ;*wpower f n*) = ($\exists$ *n*. (*wpower f n*);*f*)
using *WPowerCommute* **by** (*metis ExEqvRule*)
have *3*: $\vdash$ ($\exists$ *n*. (*wpower f n*);*f*) = ($\exists$ *n*. (*wpower f n*));*f*
by (*simp add*: *ExistChop*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FPowerStarCommute*:
$\vdash$ (*f* $\wedge$ *finite*);($\exists$ *n*. *fpower f n*) = ($\exists$ *n*. *fpower f n*);(*f* $\wedge$ *finite*)
unfolding *fpower-d-def* **using** *WPowerStarCommute*[*of LIFT* (*f* $\wedge$ *finite*)]
by *blast*

lemma *WPowerSucAndEmptyEqvAndEmpty*:
$\vdash$ (*wpower* (*f* $\wedge$ *empty*) (*Suc n*)) = (*f* $\wedge$ *empty*)
proof
(*induct n*)
case *0*
then show *?case*

by (*metis ChopEmpty wpow-0 wpow-Suc*)
next
case (*Suc n*)
then show *?case*
by (*metis AndEmptyChopAndEmptyEqvAndEmpty int-eq wpow-Suc*)
qed

lemma *FPowerSucAndEmptyEqvAndEmpty*:
⊢ (*fpower* (*f* ∧ *empty*) (*Suc n*)) = (*f* ∧ *empty*)
unfolding *fpower-d-def* **using** *WPowerSucAndEmptyEqvAndEmpty*[*of LIFT* (*f* ∧ *finite*) *n*]
WPowerEqRule[*of LIFT* ((*f* ∧ *empty*) ∧ *finite*) *LIFT* ((*f* ∧ *finite*) ∧ *empty*) (*Suc n*)]
by (*meson EmptyImpFinite Prop01 Prop04 Prop05 Prop10 WPowerEqRule WPowerSucAndEmptyEqvAndEmpty*)

lemma *WPowerOr*:
⊢ (*wpower* (*f* ∨ *g*) (*Suc n*)) = ((*f*;*wpower* (*f* ∨ *g*) *n*) ∨ (*g*;*wpower* (*f* ∨ *g*) *n*))
by (*simp add*: *OrChopEqv*)

lemma *FPowerOr*:
⊢ (*fpower* (*f* ∨ *g*) (*Suc n*)) = (((*f* ∧ *finite*);*fpower* (*f* ∨ *g*) *n*) ∨
((*g* ∧ *finite*);*fpower* (*f* ∨ *g*) *n*))
by (*simp add*: *FiniteOr OrChopEqvRule fpower-d-def*)

lemma *WPowerEmptyOrMore*:
⊢ (*wpower* ((*f* ∧ *empty*) ∨ (*f* ∧ *more*)) (*Suc n*)) =
((*f* ∧ *empty*);(*wpower* ((*f* ∧ *empty*) ∨ (*f* ∧ *more*)) *n*) ∨
(*f* ∧ *more*);(*wpower* ((*f* ∧ *empty*) ∨ (*f* ∧ *more*)) *n*))
using *WPowerOr* **by** *blast*

lemma *FPowerEmptyOrMore*:
⊢ (*fpower* ((*f* ∧ *empty*) ∨ (*f* ∧ *more*)) (*Suc n*)) =
((*f* ∧ *empty*);(*fpower* ((*f* ∧ *empty*) ∨ (*f* ∧ *more*)) *n*) ∨
(*f* ∧ *fmore*);(*fpower* ((*f* ∧ *empty*) ∨ (*f* ∧ *more*)) *n*))
using *FPowerOr*[*of LIFT* (*f* ∧ *empty*) *LIFT* (*f* ∧ *more*) *n*]
by (*metis* (*no-types*, *lifting*) *AndMoreAndFiniteEqvAndFmore EmptyImpFinite Prop01 Prop05 Prop10 int-eq*)

lemma *WPowerstarInductL*:
assumes ⊢ *g* ∨ *f*;*h* ⟶ *h*
shows ⊢ (*wpowerstar f*);*g* ⟶ *h*
proof −
have *1*: ⊢ (*wpowerstar f*);*g* = ((∃ *n*. *wpower* (*f*) *n*));*g*
by (*simp add*: *wpowerstar-d-def LeftChopEqvChop*)
have *2*: ⊢ (∃ *n*. *wpower* (*f*) *n*);*g* =
(∃ *n*. (*wpower* (*f*) *n*);*g*)
by (*metis ExistChop inteq-reflection*)
have *3*: ⋀ *n*. ⊢ (*wpower* (*f*) *n*);*g* ⟶ *h*
using *WPowerChopInductL*[*of g LIFT*(*f*) *h*] *assms* **by** *auto*
have *4*: ⊢ (∃ *n*. ((*wpower* (*f*) *n*));*g*) ⟶ *h*

by (*metis* (*mono-tags*, *lifting*) *3 Prop10 intI int-eq unl-Rex unl-lift2*)
from *1 2 4* **show** *?thesis*
by (*metis inteq-reflection*)
qed

lemma *FPowerstar-WPowerstar*:
$\vdash$ *fpowerstar f* = *wpowerstar* (*f* $\wedge$ *finite*)
unfolding *fpowerstar-d-def wpowerstar-d-def fpower-d-def* **by** *simp*

lemma *FPowerstarInductL*:
assumes $\vdash$ *g* $\vee$ (*f* $\wedge$ *finite*);*h* $\longrightarrow$ *h*
shows $\vdash$ (*fpowerstar f*);*g* $\longrightarrow$ *h*
using *assms WPowerstarInductL*[*of g LIFT* (*f* $\wedge$ *finite*) *h*]
FPowerstar-WPowerstar[*of f*] **by** (*metis int-eq*)

lemma *WPowerstarInductR*:
assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;(*wpowerstar f*) $\longrightarrow$ *h*
proof –
have *1*: $\vdash$ *g*;(*wpowerstar f*) = *g*;($\exists$ *n*. *wpower f n*)
by (*simp add*: *wpowerstar-d-def*)
have *2*: $\vdash$ (*g*;($\exists$ *n*. *wpower f n*)) = ($\exists$ *n*. *g*;(*wpower f n*))
by (*metis Prop04 ChopExist int-simps*(*31*))
have *3*: $\bigwedge$ *n*. $\vdash$ *g*;(*wpower f n*) $\longrightarrow$ *h*
using *WPowerChopInductR assms* **by** *blast*
have *4*: $\vdash$ ($\exists$ *n*. *g*;(*wpower f n*)) $\longrightarrow$ *h*
by (*metis* (*mono-tags*, *lifting*) *3 Prop10 intI int-eq unl-Rex unl-lift2*)
from *1 2 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *FPowerstarInductR*:
assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;(*fpowerstar f*) $\longrightarrow$ *h*
proof –
have *1*:$\vdash$ *g* $\vee$ *h*;(*f* $\wedge$ *finite*) $\longrightarrow$ *h*
using *assms* **using** *ChopAndA* **by** *fastforce*
show *?thesis*
using *1 WPowerstarInductR*[*of g h LIFT* (*f* $\wedge$ *finite*)]
FPowerstar-WPowerstar[*of f*]
by (*metis inteq-reflection*)
qed

lemma *WPowerstarEqv* :
$\vdash$ (*wpowerstar f*) = (*empty* $\vee$ *f*; (*wpowerstar f*))
using *WPowerstarEqvSem* **by** *blast*

lemma *FPowerstarEqv* :
$\vdash$ (*fpowerstar f*) = (*empty* $\vee$ (*f* $\wedge$ *finite*); (*fpowerstar f*))
by (*simp add*: *fpowersem fpowerstar-d-def*)

lemma *SChopstarEqv* :
 ⊢ (*schopstar f*) = (*empty* ∨ ((*f* ∧ *more*) ∧ *finite*); (*schopstar f*))
by (*simp add*: *FPowerstarEqv schopstar-d-def*)

lemma *WPowerstar-more-absorb*:
⊢ (*wpowerstar* (*f* ∧ *more*)) = (*wpowerstar f*)
proof −
 have *1*: ⊢ (*wpowerstar* (*f* ∧ *more*)) ⟶ (*wpowerstar f*)
 using *WPowerstarInductL*[*of LIFT empty LIFT f* ∧ *more LIFT* (*wpowerstar f*)]
 by (*metis AndChopA ChopEmpty Prop02 Prop05 WPowerChopInductL WPowerstarEqv int-iffD2 integ-reflection wpow-0*)
 have *2*: ⊢ *empty* ⟶ *wpowerstar* (*f* ∧ *more*)
 using *WPowerstarEqv*[*of LIFT f* ∧ *more*] **by** *fastforce*
 have *20*: ⊢ (*f* ∧ *more*);*wpowerstar* (*f* ∧ *more*) ⟶ *wpowerstar* (*f* ∧ *more*)
 by (*meson Prop03 WPowerstarEqv*)
 have *21*: ⊢ (*f* ∧ *empty*);*wpowerstar* (*f* ∧ *more*) ⟶ *wpowerstar* (*f* ∧ *more*)
 by (*metis AndChopB EmptyChop int-eq*)
 have *22*: ⊢ ((*f* ∧ *more*) ∨ (*f* ∧ *empty*));*wpowerstar* (*f* ∧ *more*) =
 ((*f* ∧ *more*);*wpowerstar* (*f* ∧ *more*) ∨ (*f* ∧ *empty*);*wpowerstar* (*f* ∧ *more*))
 by (*simp add*: *OrChopEqv*)
 have *23*: ⊢ ((*f* ∧ *more*) ∨ (*f* ∧ *empty*)) = *f*
 unfolding *empty-d-def* **by** *fastforce*
 have *3*: ⊢ *f*;*wpowerstar* (*f* ∧ *more*) ⟶ *wpowerstar* (*f* ∧ *more*)
 by (*metis 20 21 22 23 Prop02 inteq-reflection*)
 have *4*: ⊢ *empty* ∨ *f*;*wpowerstar* (*f* ∧ *more*) ⟶ *wpowerstar* (*f* ∧ *more*)
 using *2 3* **by** *fastforce*
 have *5*: ⊢ (*wpowerstar f*) ⟶ (*wpowerstar* (*f* ∧ *more*))
 using *4 WPowerstarInductL*[*of LIFT empty f LIFT* (*wpowerstar* (*f* ∧ *more*))]
 by (*metis ChopEmpty inteq-reflection*)
 show *?thesis* **using** *1 5* **by** *fastforce*
qed

lemma *FPowerstar-more-absorb*:
⊢ (*fpowerstar* (*f* ∧ *more*)) = (*fpowerstar f*)
proof −
 have *1*: ⊢ ((*f* ∧ *more*) ∧ *finite*) = ((*f* ∧ *finite*) ∧ *more*)
 by *fastforce*
 show *?thesis*
 using *FPowerstar-WPowerstar*[*of f*] *FPowerstar-WPowerstar*[*of LIFT* (*f* ∧ *more*)]
 WPowerstar-more-absorb[*of LIFT* (*f* ∧ *finite*)] *1*
 by (*metis int-eq*)
qed

lemma *SChopstar-WPowerstar*:
⊢ (*schopstar f*) = (*wpowerstar* (*f* ∧ *finite*))
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb inteq-reflection schopstar-d-def*)

lemma *SChopstar-and-more*:
⊢ (*schopstar* (*f* ∧ *more*)) = (*schopstar f*)

by (*simp add*: *FPowerstar-more-absorb schopstar-d-def*)

lemma *IStarWPowerstar*:
⊢ (*istar f*) = (*wpowerstar f*)
proof −
have *1*: ⊢ (*istar f*) ⟶ (*wpowerstar f*)
by (*metis IStarWeakInduct WPowerstarEqv int-iffD2*)
have *2*: ⊢ (*wpowerstar f*) ⟶ (*istar f*)
using *WPowerstarInductL*[*of LIFT empty f LIFT istar f*]
by (*metis ChopEmpty IStarIntros inteq-reflection*)
show *?thesis*
by (*simp add*: *1 2 Prop11*)
qed

8.5 Kleene Algebra

lemma *WPowerstar-imp-empty*:
⊢ *empty* ⟶ (*wpowerstar f*)
using *WPowerstarEqv*[*of f*] **by** *fastforce*

lemma *SChopstar-imp-empty*:
⊢ *empty* ⟶ (*schopstar f*)
using *SChopstarEqv*[*of f*] **by** *fastforce*

lemma *WPowerstar-swap*:
⊢ (*wpowerstar* (*f* ∨ *g*)) = (*wpowerstar* (*g* ∨*f*))
proof −
have *1*: ⊢ (*f* ∨ *g*) = (*g* ∨ *f*)
by *fastforce*
show *?thesis*
by (*metis 1 WPowerstarEqv inteq-reflection*)
qed

lemma *SChopstar-swap*:
⊢ (*schopstar* (*f* ∨ *g*)) = (*schopstar* (*g* ∨ *f*))
proof −
have *1*: ⊢ (*f* ∨ *g*) = (*g* ∨ *f*)
by *fastforce*
show *?thesis*
by (*metis 1 SChopstardef int-eq*)
qed

lemma *WPowerstar-1L*:
⊢ *f*;(*wpowerstar f*) ⟶ (*wpowerstar f*)
by (*meson Prop03 WPowerstarEqv*)

lemma *SChopstar-1L*:
⊢ (*f*)⌢(*schopstar f*) ⟶ (*schopstar f*)
by (*metis SChopstar-WPowerstar WPowerstar-1L inteq-reflection schop-d-def*)

lemma *SChopstarMore-1L*:
$\vdash$ $(f \wedge more)\frown(schopstar\ f) \longrightarrow (schopstar\ f)$
by (*meson AndSChopA SChopstar-1L lift-imp-trans*)

lemma *WPowerstar-trans-eq*:
$\vdash$ $(wpowerstar\ f);(wpowerstar\ f) = (wpowerstar\ f)$
proof −
have *1*: $\vdash$ $(wpowerstar\ f) \vee f;(wpowerstar\ f) \longrightarrow (wpowerstar\ f)$
by (*simp add*: *WPowerstar-1L*)
have *2*: $\vdash$ $(wpowerstar\ f);(wpowerstar\ f) \longrightarrow (wpowerstar\ f)$
using *1 WPowerstarInductL* **by** *blast*
have *3*: $\vdash$ $(wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ f)$
by (*metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection*)
show *?thesis*
by (*simp add*: *2 3 int-iffI*)
qed

lemma *SChopstar-trans-eq*:
$\vdash$ $(schopstar\ f);(schopstar\ f) = (schopstar\ f)$
by (*metis SChopstar-WPowerstar WPowerstar-trans-eq inteq-reflection*)

lemma *WPowerstar-trans*:
$\vdash$ $(wpowerstar\ f);(wpowerstar\ f) \longrightarrow (wpowerstar\ f)$
using *WPowerstar-trans-eq* **by** *fastforce*

lemma *SChopstar-trans*:
$\vdash$ $(schopstar\ f);(schopstar\ f) \longrightarrow (schopstar\ f)$
using *SChopstar-trans-eq* **by** *fastforce*

lemma *WPowerstar-induct-lvar*:
assumes $\vdash f;g \longrightarrow g$
shows $\vdash (wpowerstar\ f);g \longrightarrow g$
using *assms*
by (*simp add*: *WPowerstarInductL*)

lemma *SChopstar-induct-lvar*:
assumes $\vdash (f)\frown g \longrightarrow g$
shows $\vdash (schopstar\ f)\frown g \longrightarrow g$
using *assms*
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-lvar inteq-reflection lift-imp-trans schop-d-def*)

lemma *SChopstarMore-induct-lvar*:
assumes $\vdash (f \wedge more)\frown g \longrightarrow g$
shows $\vdash (schopstar\ f)\frown g \longrightarrow g$
using *assms*
by (*metis FPowerstar-WPowerstar SChopstar-induct-lvar SChopstar-WPowerstar inteq-reflection schopstar-d-def*)

lemma *WPowerstar-inductL-var-equiv*:

$(\vdash (\textit{wpowerstar } f);g \longrightarrow g) = (\vdash f;g \longrightarrow g)$
proof –
 have *1*: $(\vdash f;g \longrightarrow g) \Longrightarrow (\vdash (\textit{wpowerstar } f);g \longrightarrow g)$
 by (*simp add*: *WPowerstar-induct-lvar*)
 have *2*: $(\vdash (\textit{wpowerstar } f);g \longrightarrow g) \Longrightarrow (\vdash f;g \longrightarrow g)$
 by (*metis* (*no-types*, *lifting*) *AndChopB ChopAssoc EmptyChop Prop10 WPowerstar-1L WPowerstar-imp-empty int-eq lift-and-com*)
 show *?thesis*
 using *1 2* **by** *blast*
qed

lemma *SChopstar-inductL-var-equiv*:
$(\vdash (\textit{schopstar } f)\frown g \longrightarrow g) = (\vdash (f)\frown g \longrightarrow g)$
proof –
 have *1*: $(\vdash (f)\frown g \longrightarrow g) \Longrightarrow (\vdash (\textit{schopstar } f)\frown g \longrightarrow g)$
 by (*simp add*: *SChopstar-induct-lvar*)
 have *10*: $\vdash (\textit{schopstar } f) \longrightarrow (\textit{empty} \lor (f \land \textit{finite}) \lor f\frown(\textit{schopstar } f))$
 by (*metis AndSChopA FPowerstarEqv Prop05 Prop08 Prop11 schop-d-def schopstar-d-def*)
 have *101*: $\vdash (f \land \textit{finite}) \longrightarrow (\textit{schopstar } f)$
 by (*metis ChopEmpty RightChopImpChop SChopstar-1L SChopstar-imp-empty int-eq lift-imp-trans schop-d-def*)
 have *11*: $\vdash (\textit{empty} \lor (f \land \textit{finite}) \lor f\frown(\textit{schopstar } f)) \longrightarrow (\textit{schopstar } f)$
 by (*meson 101 Prop02 SChopstar-1L SChopstar-imp-empty*)
 have *12*: $\vdash (\textit{schopstar } f) = (\textit{empty} \lor (f \land \textit{finite}) \lor f\frown(\textit{schopstar } f))$
 using *10 11 int-iffI* **by** *blast*
 have *2*: $(\vdash (\textit{schopstar } f)\frown g \longrightarrow g) \Longrightarrow (\vdash (f)\frown g \longrightarrow g)$
 using *12*
 by (*metis* (*no-types*, *opaque-lifting*) *EmptySChop LeftSChopImpSChop SChopAssoc SChopstar-1L SChopstar-imp-empty int-iffI inteq-reflection*)
 show *?thesis*
 using *1 2* **by** *blast*
qed

lemma *SChopstarMore-induct-lvar-equiv*:
$(\vdash (\textit{schopstar } f)\frown g \longrightarrow g) = (\vdash (f \land \textit{more})\frown g \longrightarrow g)$
using *SChopstar-inductL-var-equiv*[*of LIFT f* $\land$ *more g*]
SChopstar-and-more[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-eq*:
assumes $\vdash f;g = g$
 shows $\vdash (\textit{wpowerstar } f);g \longrightarrow g$
using *assms*
using *WPowerstar-induct-lvar int-iffD1* **by** *blast*

lemma *SChopstar-induct-lvar-eq*:
 assumes $\vdash (f)\frown g = g$
 shows $\vdash (\textit{schopstar } f)\frown g \longrightarrow g$
using *assms*
using *SChopstar-induct-lvar int-iffD1* **by** *blast*

lemma *SChopstarMore-induct-lvar-eq*:
assumes $\vdash$ $(f \wedge more) \frown g = g$
shows $\vdash$ $(schopstar\ f) \frown g \longrightarrow g$
using *assms SChopstar-induct-lvar-eq*[*of LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-eq2*:
assumes $\vdash$ $f;g = g$
shows $\vdash$ $(wpowerstar\ f);g = g$
using *assms*
by (*meson ChopImpChop EmptyChop Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-eq lift-imp-trans*)

lemma *SChopstar-induct-lvar-eq2*:
assumes $\vdash$ $(f) \frown g = g$
shows $\vdash$ $(schopstar\ f) \frown g = g$
using *assms*
by (*metis AndSChopB EmptySChop Prop10 SChopstar-imp-empty SChopstar-induct-lvar int-eq int-iffD1 int-iffI*)

lemma *SChopstarMore-induct-lvar-eq2*:
assumes $\vdash$ $(f \wedge more) \frown g = g$
shows $\vdash$ $(schopstar\ f) \frown g = g$
using *assms SChopstar-induct-lvar-eq2*[*of LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-empty*:
assumes $\vdash$ $empty \vee f\ ;\ g \longrightarrow g$
shows $\vdash$ $(wpowerstar\ f) \longrightarrow g$
using *assms*
by (*metis ChopEmpty WPowerstarInductL inteq-reflection*)

lemma *SChopstar-induct-lvar-empty*:
assumes $\vdash$ $empty \vee (f) \frown g \longrightarrow g$
shows $\vdash$ $(schopstar\ f) \longrightarrow g$
using *assms*
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb WPowerstar-induct-lvar-empty inteq-reflection*

schop-d-def schopstar-d-def)

lemma *SChopstarMore-induct-lvar-empty*:
assumes $\vdash$ $empty \vee (f \wedge more) \frown g \longrightarrow g$
shows $\vdash$ $(schopstar\ f) \longrightarrow g$
using *assms SChopstar-induct-lvar-empty*[*of LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-star*:
assumes $\vdash$ $f\ ;\ (wpowerstar\ g) \longrightarrow (wpowerstar\ g)$
shows $\vdash$ $(wpowerstar\ f) \longrightarrow (wpowerstar\ g)$
using *assms*
by (*meson Prop02 WPowerstar-imp-empty WPowerstar-induct-lvar-empty*)

lemma *SChopstar-induct-lvar-star*:
assumes $\vdash$ $(f\ ) \frown (schopstar\ g) \longrightarrow (schopstar\ g)$
shows $\vdash (schopstar\ f) \longrightarrow (schopstar\ g)$
using *assms*
by (*meson Prop02 SChopstar-imp-empty SChopstar-induct-lvar-empty*)

lemma *SChopstarMore-induct-lvar-star*:
assumes $\vdash$ $(f \wedge more) \frown (schopstar\ g) \longrightarrow (schopstar\ g)$
shows $\vdash (schopstar\ f) \longrightarrow (schopstar\ g)$
using *assms* **using** *SChopstar-induct-lvar-star*[*of LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-leq*:
assumes $\vdash (h \vee f;g) = g$
shows $\vdash (wpowerstar\ f);h \longrightarrow g$
using *assms*
using *WPowerstarInductL int-iffD1* **by** *blast*

lemma *SChopstar-induct-leq*:
assumes $\vdash (h \vee (f)\frown g) = g$
shows $\vdash (schopstar\ f)\frown h \longrightarrow g$
using *assms*
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-leq inteq-reflection lift-imp-trans schop-d-def*)

lemma *SChopstarMore-induct-leq*:
assumes $\vdash (h \vee (f \wedge more)\frown g) = g$
shows $\vdash (schopstar\ f)\frown h \longrightarrow g$
using *assms SChopstar-induct-leq*[*of h LIFT f* $\wedge$ *more g*] *SChopstar-and-more*[*of f*] **by** (*metis int-eq*)

lemma *WPowerstar-subid*:
assumes $\vdash f \longrightarrow empty$
shows $\vdash (wpowerstar\ f) = empty$
using *assms*
by (*meson ChopEmpty Prop02 Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-empty lift-imp-trans*)

lemma *SChopstar-subid*:
assumes $\vdash f \longrightarrow empty$
shows $\vdash (schopstar\ f) = empty$
using *assms*
by (*metis EmptyImpFinite FPowerstar-WPowerstar FPowerstar-more-absorb Prop10 WPowerstar-subid int-eq*
lift-imp-trans schopstar-d-def)

lemma *WPowerstar-subdist*:
$\vdash (wpowerstar\ f) \longrightarrow (wpowerstar\ (f \vee g))$

proof −
 have *1*: ⊢ *f*;(*wpowerstar* (*f* ∨ *g*)) ⟶ (*f* ∨ *g*);(*wpowerstar* (*f* ∨ *g*))
  using *OrChopEqv* **by** *fastforce*
 have *2*: ⊢ (*f* ∨ *g*);(*wpowerstar* (*f* ∨ *g*)) ⟶ (*wpowerstar* (*f* ∨ *g*))
  by (*simp add*: *WPowerstar-1L*)
 show *?thesis*
 using *1 2 WPowerstar-induct-lvar-star lift-imp-trans* **by** *blast*
qed

lemma *SChopstar-subdist*:
⊢ (*schopstar f*) ⟶ (*schopstar* (*f* ∨ *g*))
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb FiniteOr WPowerstar-subdist int-eq schopstar-d-def*)

lemma *WPowerstar-subdist-var*:
⊢ (*wpowerstar f*) ∨ (*wpowerstar g*) ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*metis Prop02 WPowerstar-subdist WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var*:
⊢ (*schopstar f*) ∨ (*schopstar g*)⟶ (*schopstar* (*f* ∨ *g*))
by (*metis Prop02 SChopstar-subdist SChopstar-swap inteq-reflection*)

lemma *WPowerstar-iso*:
 assumes ⊢ *f* ⟶ *g*
 shows ⊢ (*wpowerstar f*) ⟶ (*wpowerstar g*)
by (*metis AndChopB Prop05 Prop10 WPowerstar-induct-lvar-star WPowerstarEqv assms inteq-reflection*)

lemma *SChopstar-iso*:
 assumes ⊢ *f* ⟶ *g*
 shows ⊢ (*schopstar f*) ⟶ (*schopstar g*)
using *assms*
by (*metis AndSChopB Prop10 SChopstar-1L SChopstar-induct-lvar-star inteq-reflection lift-imp-trans*)

lemma *WPowerstar-invol*:
⊢ (*wpowerstar* (*wpowerstar f*)) = (*wpowerstar f*)
proof −
 have *1*: ⊢ (*wpowerstar f*);(*wpowerstar f*) = (*wpowerstar f*)
  by (*simp add*: *WPowerstar-trans-eq*)
 have *2*: ⊢ (*wpowerstar* (*wpowerstar f*)) ⟶ (*wpowerstar f*)
  using *1 WPowerstar-induct-lvar-star int-iffD1* **by** *blast*
 have *3*: ⊢ (*wpowerstar* (*wpowerstar f*));(*wpowerstar* (*wpowerstar f*)) ⟶ (*wpowerstar* (*wpowerstar f*))
  by (*simp add*: *WPowerstar-trans*)
 have *4*: ⊢ *f*;(*wpowerstar* (*wpowerstar f*)) ⟶ (*wpowerstar* (*wpowerstar f*))
  using *WPowerstar-1L WPowerstar-inductL-var-equiv* **by** *blast*
 have *5*: ⊢ (*wpowerstar f*) ⟶ (*wpowerstar* (*wpowerstar f*))
  by (*simp add*: *4 WPowerstar-induct-lvar-star*)
 show *?thesis*
 by (*simp add*: *2 5 Prop11*)
qed

lemma *SChopstar-invol*:
$\vdash$ (*schopstar* (*schopstar f*)) = (*schopstar f*)
by (*meson Prop11 SChopstar-1L SChopstar-inductL-var-equiv SChopstar-induct-lvar-star*)

lemma *WPowerstar-star2*:
$\vdash$ (*wpowerstar* (*empty* $\vee$ *f*)) = (*wpowerstar f*)
by (*meson EmptyOrChopEqv Prop02 Prop03 Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-empty*

WPowerstarEqv lift-imp-trans)

lemma *SChopstar-star2*:
$\vdash$ (*schopstar* (*empty* $\vee$ *f*)) = (*schopstar f*)
by (*metis EmptyImpFinite FiniteOr Prop10 SChopstar-WPowerstar WPowerstar-star2 int-eq*)

lemma *Chop-WPowerstar-Closure*:
assumes $\vdash$ *f* $\longrightarrow$ (*wpowerstar h*)
$\vdash$ *g* $\longrightarrow$ (*wpowerstar h*)
shows $\vdash$ *f*;*g* $\longrightarrow$ (*wpowerstar h*)
proof −
have *1*: $\vdash$ *g* $\vee$ (*wpowerstar h*);(*wpowerstar h*) $\longrightarrow$ (*wpowerstar h*)
by (*metis Prop02 WPowerstar-1L WPowerstar-induct-lvar assms*(*2*))
have *2*: $\vdash$ (*wpowerstar h*); *g* $\longrightarrow$ (*wpowerstar h*)
by (*meson Prop02 WPowerstarInductL WPowerstar-1L assms*(*2*))
have *3*: $\vdash$ *f*;*g* $\longrightarrow$ (*wpowerstar h*);*g*
by (*simp add*: *LeftChopImpChop assms*(*1*))
show *?thesis*
using *2 3 lift-imp-trans* **by** *blast*
qed

lemma *SChop-SChopstar-Closure*:
assumes $\vdash$ *f* $\longrightarrow$ (*schopstar h*)
$\vdash$ *g* $\longrightarrow$ (*schopstar h*)
shows $\vdash$ $f \frown g$ $\longrightarrow$ (*schopstar h*)
using *assms*
by (*metis AndSChopA Prop10 SChopAndB SChopstarMore-1L SChopstarMore-induct-lvar inteq-reflection lift-and-com lift-imp-trans*)

lemma *WPowerstar-wpowerstar-closure*:
assumes $\vdash$ (*wpowerstar f*) $\longrightarrow$ (*wpowerstar h*)
shows $\vdash$ (*wpowerstar* (*wpowerstar f*)) $\longrightarrow$ (*wpowerstar h*)
using *assms*
by (*simp add*: *Chop-WPowerstar-Closure WPowerstar-induct-lvar-star*)

lemma *SChopstar-SChopstar-closure*:
assumes $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar h*)
shows $\vdash$ (*schopstar* (*schopstar f*)) $\longrightarrow$ (*schopstar h*)
using *assms*
by (*metis SChopstar-invol inteq-reflection*)

lemma *WPowerstar-closed-unfold*:
assumes $\vdash$ (*wpowerstar f*) = *f*
shows $\vdash$ *f* = (*empty* $\vee$ *f*;*f*)
using *assms*
by (*metis WPowerstarEqv int-eq*)

lemma *SChopstar-closed-unfold*:
assumes $\vdash$ (*schopstar f*) = *f*
shows $\vdash$ *f* = (*empty* $\vee$ (*f*)$\frown$*f*)
using *assms*
by (*metis SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *SChopstarMore-closed-unfold*:
assumes $\vdash$ (*schopstar f*) = *f*
shows $\vdash$ *f* = (*empty* $\vee$ (*f* $\wedge$ *more*)$\frown$*f*)
using *assms*
by (*metis SChopstarEqv int-eq schop-d-def*)

lemma *WPowerstar-ext*:
$\vdash$ *f* $\longrightarrow$ (*wpowerstar f*)
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *f*;(*wpowerstar f*)
by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty int-eq*)
show *?thesis* **by** (*meson 1 WPowerstar-1L lift-imp-trans*)
qed

lemma *SChopstar-ext*:
$\vdash$ *f* $\wedge$ *finite*$\longrightarrow$ (*schopstar f*)
by (*metis SChopstar-WPowerstar WPowerstar-ext inteq-reflection*)

lemma *SChopstarMore-ext*:
$\vdash$ *f* $\wedge$ *more* $\wedge$ *finite*$\longrightarrow$ (*schopstar f*)
by (*metis AndMoreAndFiniteEqvAndFmore FPowerstar-WPowerstar SChopstar-ext SChopstar-WPowerstar fmore-d-def int-eq schopstar-d-def*)

lemma *WPowerstar-1R*:
$\vdash$ (*wpowerstar f*) ; *f* $\longrightarrow$ (*wpowerstar f*)
by (*simp add*: *Chop-WPowerstar-Closure WPowerstar-ext*)

lemma *SChopstar-1R*:
$\vdash$ (*schopstar f*) $\frown$ (*f* $\wedge$ *finite*) $\longrightarrow$ (*schopstar f*)
by (*simp add*: *SChop-SChopstar-Closure SChopstar-ext*)

lemma *SChopstarMore-1R*:
$\vdash$ (*schopstar f*) $\frown$ (*f* $\wedge$ *fmore*) $\longrightarrow$ (*schopstar f*)
by (*simp add*: *SChop-SChopstar-Closure SChopstarMore-ext fmore-d-def*)

lemma *WPowerstar-unfoldR*:
$\vdash$ *empty* $\vee$ (*wpowerstar f*) ; *f* $\longrightarrow$ (*wpowerstar f*)
by (*meson Prop02 WPowerstar-1R WPowerstar-imp-empty*)

lemma *SChopstar-unfoldR*:
$\vdash$ *empty* $\vee$ (*schopstar f*) $\frown$ (*f* $\wedge$ *finite*) $\longrightarrow$ (*schopstar f*)
by (*meson Prop02 SChopstar-1R SChopstar-imp-empty*)

lemma *SChopstarMore-unfoldR*:
$\vdash$ *empty* $\vee$ (*schopstar f*) $\frown$ (*f* $\wedge$ *fmore*) $\longrightarrow$ (*schopstar f*)
by (*meson Prop02 SChopstarMore-1R SChopstar-imp-empty*)

lemma *WPowerstar-sim1*:
assumes $\vdash$ *f* ; *h* $\longrightarrow$ *h* ; *g*
shows $\vdash$ (*wpowerstar f*) ;*h* $\longrightarrow$ *h* ;(*wpowerstar g*)
proof –
have *1*: $\vdash$ (*f* ;*h*);(*wpowerstar g*) $\longrightarrow$ (*h* ; *g*);(*wpowerstar g*)
by (*simp add*: *LeftChopImpChop assms*)
have *2*: $\vdash$ (*h* ; *g*);(*wpowerstar g*) $\longrightarrow$ *h* ;(*wpowerstar g*)
by (*metis ChopAssoc RightChopImpChop WPowerstar-1L inteq-reflection*)
have *3*: $\vdash$ (*f* ; *h*);(*wpowerstar g*) $\longrightarrow$ *h* ;(*wpowerstar g*)
using *1 2 lift-imp-trans* **by** *blast*
have *4*: $\vdash$ (*wpowerstar g*) =(*empty* $\vee$ *g*;(*wpowerstar g*))
by (*simp add*: *WPowerstarEqv*)
have *41*: $\vdash$ *h*;(*empty* $\vee$ *g*;(*wpowerstar g*)) = (*h*;*empty* $\vee$ *h*;(*g*;(*wpowerstar g*)))
by (*simp add*: *ChopOrEqv*)
have *42*: $\vdash$ *h*;(*g*;(*wpowerstar g*)) = (*h*;*g*);(*wpowerstar g*)
by (*simp add*: *ChopAssoc*)
have *5*: $\vdash$ *h* ;(*wpowerstar g*) = (*h* $\vee$ (*h*;*g*);(*wpowerstar g*))
by (*metis 4 41 42 ChopEmpty inteq-reflection*)
have *6*: $\vdash$ *h* $\longrightarrow$ *h* ;(*wpowerstar g*)
using *5* **by** *fastforce*
have *7*: $\vdash$ *h* $\vee$ (*f* ; *h*);(*wpowerstar g*) $\longrightarrow$ *h* ;(*wpowerstar g*)
using *3 6 Prop02* **by** *blast*
show *?thesis*
using *WPowerstarInductL*[*of LIFT h f LIFT h* ;(*wpowerstar g*)]
by (*metis 3 6 ChopAssoc Prop02 inteq-reflection*)
qed

lemma *SChopstar-sim1*:
assumes $\vdash$ *f* $\frown$ *h* $\longrightarrow$ *h* $\frown$ *g*
shows $\vdash$ (*schopstar f*) $\frown$(*h* $\wedge$ *finite*) $\longrightarrow$ *h* $\frown$(*schopstar g*)
proof –
have *1*: $\vdash$ (*f* $\frown$*h*)$\frown$(*schopstar g*) $\longrightarrow$ (*h* $\frown$ *g*)$\frown$(*schopstar g*)
by (*simp add*: *LeftSChopImpSChop assms*)
have *2*: $\vdash$ (*h* $\frown$ *g*)$\frown$(*schopstar g*) $\longrightarrow$ *h* $\frown$(*schopstar g*)
by (*metis RightSChopImpSChop SChopAssoc SChopstar-1L inteq-reflection*)
have *3*: $\vdash$ (*f* $\frown$ *h*)$\frown$(*schopstar g*) $\longrightarrow$ *h* $\frown$(*schopstar g*)
using *1 2 lift-imp-trans* **by** *blast*

have *4*: ⊢ (*schopstar g*) =(*empty* ∨ (*g* ∧ *more*)⁀(*schopstar g*))
by (*simp add*: *SChopstarEqv schop-d-def*)
have *5*: ⊢ *h* ⁀(*schopstar g*) = ((*h* ∧ *finite*) ∨ (*h*⁀(*g* ∧ *more*))⁀(*schopstar g*))
by (*metis ChopEmpty SChopAssoc SChopOrEqv SChopstarEqv int-eq schop-d-def*)
have *6*: ⊢ *h* ∧ *finite* ⟶ *h* ⁀(*schopstar g*)
using *5* **by** *fastforce*
have *7*: ⊢ (*h* ∧ *finite*) ∨ (*f* ⁀ *h*)⁀(*schopstar g*) ⟶ *h* ⁀(*schopstar g*)
using *3 6 Prop02* **by** *blast*
show *?thesis* **using** *7*
by (*metis 3 6 Prop10 SChopAndB SChopAssoc SChopstar-induct-lvar inteq-reflection lift-imp-trans*)
qed

lemma *SChopstarMore-sim1*:
assumes ⊢ (*f* ∧ *more*) ⁀ *h* ⟶ *h* ⁀ (*g* ∧ *more*)
shows ⊢ (*schopstar f*) ⁀(*h* ∧ *finite*) ⟶ *h* ⁀(*schopstar g*)
using *assms SChopstar-sim1*[*of LIFT f* ∧ *more h LIFT g* ∧ *more*]
by (*metis SChopstar-and-more int-eq*)

lemma *WPowerstar-Quasicomm-varA*:
assumes ⊢ (*g*; *f* ⟶ *f* ; (*wpowerstar* (*f* ∨ *g*)))
shows ⊢ ((*wpowerstar g*);*f* ⟶ *f* ; (*wpowerstar* (*f* ∨ *g*)))
proof −
have *0*: ⊢ (*wpowerstar* (*wpowerstar* (*f* ∨ *g*))) = (*wpowerstar* (*f* ∨ *g*))
by (*meson WPowerstar-1L WPowerstar-inductL-var-equiv WPowerstar-induct-lvar-star int-iffI*)
have *2*: ⊢ *f* ; *wpowerstar wpowerstar* (*f* ∨ *g*) =
f ; (*wpowerstar* (*f* ∨ *g*))
by (*simp add*: *0 RightChopEqvChop*)
have *4*: ⊢ (*wpowerstar g* ; *f* ⟶
f ; *wpowerstar* (*f* ∨ *g*)) =
(*wpowerstar g* ; *f* ⟶
f ; *wpowerstar wpowerstar* (*f* ∨ *g*))
using *2* **by** *fastforce*
show *?thesis*
using *4 WPowerstar-sim1*[*of LIFT g LIFT f LIFT* (*wpowerstar* (*f* ∨ *g*))]
by (*metis 0 assms int-eq*)
qed

lemma *SChopstar-Quasicomm-varA*:
assumes ⊢ ((*g*)⁀ ((*f*) ∧ *finite*) ⟶ (*f*) ⁀ (*schopstar* (*f* ∨ *g*)))
shows ⊢ ((*schopstar g*)⁀((*f*) ∧ *finite*) ⟶ (*f*) ⁀ (*schopstar* (*f* ∨ *g*)))
proof −
have *0*: ⊢ (*schopstar* (*schopstar* (*f* ∨ *g*))) = (*schopstar* (*f* ∨ *g*))
by (*simp add*: *SChopstar-invol*)
have *2*: ⊢((*f*) ∧ *finite*) ⁀ *schopstar schopstar* (*f* ∨ *g*) =
(*f*) ⁀ (*schopstar* (*f* ∨ *g*))
by (*metis* (*no-types*, *lifting*) *0 AndSChopA Prop11 Prop12 inteq-reflection itl-def*(*9*) *lift-and-com*)
have *3*: ⊢ (*schopstar* (*g*) ⁀ (((*f*) ∧ *finite*) ∧ *finite*)) =
((*schopstar g*) ⁀ ((*f*) ∧ *finite*))
by (*metis Prop12 RightSChopImpSChop int-iffD2 int-iffI lift-and-com*)

have *4*: ⊢ (*schopstar* (*g*) ⌢ (((*f*) ∧ *finite*)) ⟶
(*f*) ⌢ *schopstar* (*f* ∨ *g*)) =
(*schopstar* (*g*) ⌢ (((*f*) ∧ *finite*) ∧ *finite*) ⟶
((*f*) ∧ *finite*) ⌢ *schopstar schopstar* (*f* ∨ *g*))

using *2 3* **by** *fastforce*
show *?thesis*
using *4 SChopstar-sim1*[*of LIFT g LIFT* (*f*) ∧ *finite LIFT* (*schopstar* (*f* ∨ *g*))]
by (*metis 0 2 assms int-eq*)
qed

lemma *SChopstarMore-or-and*:
⊢ *schopstar* (*f* ∧ *more* ∨ *g* ∧ *more*) = (*schopstar* ((*f* ∨ *g*) ∧ *more*))
proof −
have *1*: ⊢ (*f* ∧ *more* ∨ *g* ∧ *more*) = ((*f* ∨ *g*) ∧ *more*)
by *fastforce*
show *?thesis*
by (*metis 1 SChopstardef int-eq*)
qed

lemma *SChopstar-QuasicommMore-varA*:
assumes ⊢ ((*g* ∧ *more*)⌢ ((*f* ∧ *more*) ∧ *finite*) ⟶ (*f* ∧ *more*) ⌢ (*schopstar* (*f* ∨ *g*)))
shows ⊢ ((*schopstar g*)⌢((*f* ∧ *more*) ∧ *finite*) ⟶ (*f* ∧ *more*) ⌢ (*schopstar* (*f* ∨ *g*)))
using *assms SChopstar-Quasicomm-varA*[*of LIFT g* ∧ *more LIFT f* ∧ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*] *SChopstar-and-more*[*of LIFT* (*f* ∨ *g*)]
AndMoreAndFiniteEqvAndFmore[*of f*]
by (*metis SChopstarMore-or-and inteq-reflection*)

lemma *WPowerstar-Quasicomm-varB*:
assumes ⊢ ((*wpowerstar g*);*f* ⟶ *f* ; (*wpowerstar* (*f* ∨ *g*)))
shows ⊢ (*g*; *f* ⟶ *f* ; (*wpowerstar* (*f* ∨ *g*)))
proof −
have *1*: ⊢ *g*; *f* ⟶ (*wpowerstar g*);*f*
by (*simp add*: *LeftChopImpChop WPowerstar-ext*)
show *?thesis*
using *1 assms lift-imp-trans* **by** *blast*
qed

lemma *SChopstar-Quasicomm-varB*:
assumes ⊢ ((*schopstar g*)⌢((*f*) ∧ *finite*) ⟶ (*f*) ⌢ (*schopstar* (*f* ∨ *g*)))
shows ⊢ ((*g*)⌢ ((*f*) ∧ *finite*) ⟶ (*f*) ⌢ (*schopstar* (*f* ∨ *g*)))
proof −
have *1*: ⊢ (*g*)⌢ ((*f*) ∧ *finite*) ⟶ (*schopstar g*)⌢((*f*) ∧ *finite*)
by (*metis LeftChopImpChop Prop12 SChopstar-ext int-iffD2 lift-and-com schop-d-def*)
show *?thesis*
using *1 assms lift-imp-trans* **by** *blast*
qed

lemma *SChopstar-QuasicommMore-varB*:

assumes $\vdash ((schopstar\ g)\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))$
shows $\vdash ((g \wedge more)\frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))$
using *assms SChopstar-Quasicomm-varB[of LIFT g ∧ more LIFT f ∧ more] SChopstar-and-more[of f] SChopstar-and-more[of g] SChopstar-and-more[of LIFT (f ∨ g)] AndMoreAndFiniteEqvAndFmore[of f]*
by (*metis SChopstarMore-or-and inteq-reflection*)

lemma *WPowerstar-Quasicomm-var*:
$(\vdash (g;\ f \longrightarrow f\ ;\ (wpowerstar\ (f \vee g)))\ ) =$
$(\vdash\ ((wpowerstar\ g);f \longrightarrow f\ ;\ (wpowerstar\ (f \vee g))))$
using *WPowerstar-Quasicomm-varA WPowerstar-Quasicomm-varB* **by** *blast*

lemma *SChopstar-Quasicomm-var*:
$(\vdash ((g)\frown ((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g)))\ ) =$
$(\vdash\ ((schopstar\ g)\frown((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g))))$
using *SChopstar-Quasicomm-varA SChopstar-Quasicomm-varB* **by** *blast*

lemma *SChopstar-QuasicommMore-var*:
$(\vdash ((g \wedge more)\frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))\ ) =$
$(\vdash\ ((schopstar\ g)\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g))))$
using *SChopstar-QuasicommMore-varA SChopstar-QuasicommMore-varB* **by** *blast*

lemma *WPowerstar-slide1*:
$\vdash (wpowerstar\ (f\ ;\ g));f \longrightarrow f;(wpowerstar\ (g;f))$
by (*simp add*: *ChopAssoc WPowerstar-sim1 int-iffD2*)

lemma *SChopstar-slide1*:
$\vdash (schopstar\ (f \frown g))\frown(f \wedge finite) \longrightarrow f\frown(schopstar\ (g\frown f))$
using *SChopstar-sim1*
by (*metis SChopAssoc int-iffD2*)

lemma *WPowerstar-slide1-var1*:
$\vdash (wpowerstar\ f);f \longrightarrow f;(wpowerstar\ f)$
by (*meson Prop04 WPowerstar-sim1 int-iffD1 int-simps*(*27*))

lemma *SChopstar-slide1-var1*:
$\vdash (schopstar\ f)\frown(f \wedge finite) \longrightarrow f\frown(schopstar\ f)$
by (*simp add*: *SChopstar-sim1*)

lemma *SChopstarMore-slide1-var1*:
$\vdash (schopstar\ f)\frown((f \wedge more) \wedge finite) \longrightarrow (f \wedge more)\frown(schopstar\ f)$
using *SChopstar-slide1-var1[of LIFT f ∧ more] SChopstar-and-more[of f]*
by (*metis inteq-reflection*)

lemma *wpowerstar-unfoldl-eq*:
$\vdash (empty \vee f;(wpowerstar\ f)) = (wpowerstar\ f)$
by (*meson Prop04 WPowerstar-1L WPowerstar-induct-lvar-star WPowerstarEqv int-iffI*)

lemma *SChopstar-unfoldl-eq*:
$\vdash$ (*empty* $\vee$ *f*$\frown$(*schopstar f*)) = (*schopstar f*)
by (*metis SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *SChopstarMore-unfoldl-eq*:
$\vdash$ (*empty* $\vee$ (*f* $\wedge$ *more*)$\frown$(*schopstar f*)) = (*schopstar f*)
using *SChopstar-unfoldl-eq*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-rtc1-eq*:
$\vdash$ (*empty* $\vee$ *f* $\vee$ (*wpowerstar f*);(*wpowerstar f*)) = (*wpowerstar f*)
by (*meson Prop02 Prop05 Prop11 WPowerstar-ext WPowerstar-imp-empty WPowerstar-trans-eq*)

lemma *SChopstar-rtc1-eq*:
$\vdash$ (*empty* $\vee$ (*f* $\wedge$ *finite*) $\vee$ (*schopstar f*)$\frown$(*schopstar f*)) = (*schopstar f*)
by (*meson EmptySChop LeftSChopImpSChop Prop02 Prop04 Prop05 Prop08 Prop11 SChop-SChopstar-Closure*

SChopstar-ext SChopstar-imp-empty)

lemma *SChopstarMore-rtc1-eq*:
$\vdash$ (*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\vee$ (*schopstar f*)$\frown$(*schopstar f*)) = (*schopstar f*)
using *SChopstar-rtc1-eq*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-rtc1*:
$\vdash$ (*empty* $\vee$ *f* $\vee$ (*wpowerstar f*);(*wpowerstar f*)) $\longrightarrow$ (*wpowerstar f*)
by (*meson WPowerstar-rtc1-eq int-iffD1*)

lemma *SChopstar-rtc1*:
$\vdash$ (*empty* $\vee$ (*f* $\wedge$ *finite*) $\vee$ (*schopstar f*)$\frown$(*schopstar f*)) $\longrightarrow$ (*schopstar f*)
by (*meson SChopstar-rtc1-eq int-iffD1*)

lemma *SChopstarMore-rtc1*:
$\vdash$ (*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\vee$ (*schopstar f*)$\frown$(*schopstar f*)) $\longrightarrow$ (*schopstar f*)
using *SChopstar-rtc1*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-rtc2*:
($\vdash$ *empty* $\vee$ *f*;*f* $\longrightarrow$ *f*) = ($\vdash$ *f* = (*wpowerstar f*))
proof –
have *1*: ($\vdash$ *empty* $\vee$ *f*;*f* $\longrightarrow$ *f*) $\Longrightarrow$ ($\vdash$ *f* = (*wpowerstar f*))
using *WPowerstar-induct-lvar-empty*[*of f LIFT f*]
by (*simp add*: *WPowerstar-ext int-iffI*)
have *2*: ($\vdash$ *f* = (*wpowerstar f*)) $\Longrightarrow$ ($\vdash$ *empty* $\vee$ *f*;*f* $\longrightarrow$ *f*)
by (*metis WPowerstar-unfoldR inteq-reflection*)
show *?thesis*
by (*metis 1 2 inteq-reflection*)

qed

lemma *SChopstar-rtc2*:
$(\vdash \textit{empty} \vee (f)\frown(f \wedge \textit{finite}) \longrightarrow (f \wedge \textit{finite})) = (\vdash (f \wedge \textit{finite}) = (\textit{schopstar}\ f))$
proof –
have *1*: $(\vdash \textit{empty} \vee (f)\frown(f \wedge \textit{finite}) \longrightarrow (f \wedge \textit{finite})) \Longrightarrow (\vdash (f \wedge \textit{finite}) = (\textit{schopstar}\ f))$
using *SChopstar-induct-lvar-empty*[*of f LIFT f* $\wedge$ *finite*]
by (*simp add*: *Prop11 SChopstar-ext*)
have *2*: $(\vdash (f \wedge \textit{finite}) = (\textit{schopstar}\ f)) \Longrightarrow (\vdash \textit{empty} \vee (f)\frown(f \wedge \textit{finite}) \longrightarrow (f \wedge \textit{finite}))$
by (*metis Prop02 SChopstar-1L SChopstar-imp-empty inteq-reflection*)
show *?thesis*
by (*metis 1 2 inteq-reflection*)
qed

lemma *SChopstarMore-rtc2*:
$(\vdash \textit{empty} \vee (f \wedge \textit{more})\frown((f \wedge \textit{more}) \wedge \textit{finite}) \longrightarrow ((f \wedge \textit{more}) \wedge \textit{finite})) =$
$(\vdash ((f \wedge \textit{more}) \wedge \textit{finite}) = (\textit{schopstar}\ f))$
using *SChopstar-rtc2*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *SChopstarMore-rtc2-alt*:
$(\vdash \textit{empty} \vee (f \wedge \textit{more})\frown(f \wedge \textit{finite}) \longrightarrow (f \wedge \textit{finite})) = (\vdash (f \wedge \textit{finite}) = (\textit{schopstar}\ f))$
proof –
have *1*: $(\vdash \textit{empty} \vee (f \wedge \textit{more})\frown(f \wedge \textit{finite}) \longrightarrow (f \wedge \textit{finite})) \Longrightarrow (\vdash (f \wedge \textit{finite}) = (\textit{schopstar}\ f))$
using *SChopstarMore-induct-lvar-empty*[*of f LIFT f* $\wedge$ *finite*]
by (*simp add*: *SChopstar-ext int-iffI*)
have *2*: $(\vdash (f \wedge \textit{finite}) = (\textit{schopstar}\ f)) \Longrightarrow (\vdash \textit{empty} \vee (f \wedge \textit{more})\frown(f \wedge \textit{finite}) \longrightarrow (f \wedge \textit{finite}))$
by (*metis SChopstarMore-unfoldl-eq int-eq int-iffD1*)
show *?thesis*
by (*metis 1 2 inteq-reflection*)
qed

lemma *WPowerstar-rtc3*:
$(\vdash (\textit{empty} \vee f;f) = f) = (\vdash f = (\textit{wpowerstar}\ f))$
by (*metis WPowerstar-rtc2 int-iffD1 inteq-reflection wpowerstar-unfoldl-eq*)

lemma *SChopstar-rtc3*:
$(\vdash (\textit{empty} \vee (f)\frown(f \wedge \textit{finite})) = (f \wedge \textit{finite})) = (\vdash (f \wedge \textit{finite}) = (\textit{schopstar}\ f))$
by (*metis SChopstar-rtc2 SChopstar-unfoldl-eq int-iffD1 inteq-reflection*)

lemma *SChopstarMore-rtc3*:
$(\vdash (\textit{empty} \vee (f \wedge \textit{more})\frown((f \wedge \textit{more}) \wedge \textit{finite})) = ((f \wedge \textit{more}) \wedge \textit{finite})) =$
$(\vdash ((f \wedge \textit{more}) \wedge \textit{finite}) = (\textit{schopstar}\ f))$
using *SChopstar-rtc3*[*of LIFT f* $\wedge$ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *SChopstarMore-rtc3-alt*:
$(\vdash (\textit{empty} \vee (f \wedge \textit{more})\frown(f \wedge \textit{finite})) = (f \wedge \textit{finite})) = (\vdash (f \wedge \textit{finite}) = (\textit{schopstar}\ f))$
by (*metis SChopstarMore-induct-lvar-empty SChopstarMore-unfoldl-eq SChopstar-ext int-iffD1 int-iffI*

inteq-reflection)

lemma *WPowerstar-rtc-least*:
assumes ⊢ *empty* ∨ *f* ∨ *g*;*g* ⟶ *g*
shows ⊢ (*wpowerstar f*) ⟶ *g*
proof −
have *1*: ⊢ *f* ⟶ *g*
using *assms* **by** *fastforce*
have *2*: ⊢ *g*;*g* ⟶ *g*
using *assms* **by** *fastforce*
have *3*: ⊢ *f*;*g* ⟶ *g* ;*g*
by (*metis 1 LeftChopImpChop*)
have *4*: ⊢ *f* ;*g* ⟶ *g*
using *2 3 lift-imp-trans* **by** *blast*
have *5*: ⊢ *empty* ⟶ *g*
using *assms* **by** *fastforce*
show *?thesis*
by (*meson 4 5 Prop02 WPowerstar-induct-lvar-empty*)
qed

lemma *SChopstar-rtc-least*:
assumes ⊢ *empty* ∨ (*f* ∧ *finite*) ∨ (*g*)⌢(*g*) ⟶ (*g*)
shows ⊢ (*schopstar f*) ⟶ *g*
proof −
have *1*: ⊢ (*f* ∧ *finite*) ⟶ (*g* ∧ *finite*)
using *assms* **by** *fastforce*
have *2*: ⊢ (*g*)⌢(*g*) ⟶ *g*
using *assms* **by** *fastforce*
have *3*: ⊢ (*f*)⌢(*g*) ⟶ (*g*) ⌢(*g*)
by (*metis 1 LeftChopImpChop schop-d-def*)
have *4*: ⊢ (*f*)⌢(*g*) ⟶ *g*
using *2 3 lift-imp-trans* **by** *blast*
have *5*: ⊢ *empty* ⟶ *g*
using *assms* **by** *fastforce*
show *?thesis*
by (*meson 4 5 Prop02 SChopstar-induct-lvar-empty*)
qed

lemma *SChopstarMore-rtc-least*:
assumes ⊢ *empty* ∨ ((*f* ∧ *more*) ∧ *finite*) ∨ (*g*)⌢(*g*) ⟶ (*g*)
shows ⊢ (*schopstar f*) ⟶ *g*
using *assms SChopstar-rtc-least*[*of LIFT f* ∧ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-rtc-least-eq*:
assumes ⊢ (*empty* ∨ *f* ∨ *g*;*g*) = *g*
shows ⊢ (*wpowerstar f*) ⟶ *g*
using *assms*

using *WPowerstar-rtc-least int-iffD1* **by** *blast*

lemma *SChopstar-rtc-least-eq*:
assumes ⊢ (*empty* ∨ (*f* ∧ *finite*) ∨ (*g*)⌢(*g*)) = (*g*)
shows ⊢ (*schopstar f*) ⟶ *g*
using *SChopstar-rtc-least assms int-iffD1* **by** *blast*

lemma *SChopstarMore-rtc-least-eq*:
assumes ⊢ (*empty* ∨ ((*f* ∧ *more*) ∧ *finite*) ∨ (*g*)⌢(*g*)) = (*g*)
shows ⊢ (*schopstar f*) ⟶ *g*
using *assms SChopstar-rtc-least-eq*[*of LIFT f* ∧ *more*] *SChopstar-and-more*[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-subdist-var-1*:
⊢ *f* ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*meson WPowerstar-ext WPowerstar-subdist lift-imp-trans*)

lemma *SChopstar-subdist-var-1*:
⊢ *f* ∧ *finite* ⟶ (*schopstar* (*f* ∨ *g*))
by (*meson SChopstar-ext SChopstar-subdist lift-imp-trans*)

lemma *WPowerstar-subdist-var-2*:
⊢ *f*;*g* ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*metis Chop-WPowerstar-Closure WPowerstar-subdist-var-1 WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var-2*:
⊢ (*f*)⌢(*g* ∧ *finite*) ⟶ (*schopstar* (*f* ∨ *g*))
by (*metis Chop-WPowerstar-Closure SChopstar-swap SChopstar-WPowerstar SChopstar-subdist-var-1 inteq-reflection schop-d-def*)

lemma *SChopstarMore-subdist-var-2*:
⊢ (*f* ∧ *more*)⌢((*g* ∧ *more*) ∧ *finite*) ⟶ (*schopstar* (*f* ∨ *g*))
using *SChopstar-subdist-var-2*[*of LIFT f* ∧ *more LIFT g* ∧ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT* (*f* ∨ *g*)]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-subdist-var-3*:
⊢ (*wpowerstar f*); (*wpowerstar g*) ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*metis Chop-WPowerstar-Closure WPowerstar-subdist WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var-3*:
⊢ (*schopstar f*)⌢ (*schopstar g*) ⟶ (*schopstar* (*f* ∨ *g*))
by (*metis SChop-SChopstar-Closure SChopstar-subdist SChopstar-swap inteq-reflection*)

lemma *WPowerstar-denest*:
⊢ (*wpowerstar* (*f* ∨ *g*)) = (*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
proof –
have *1*: ⊢ *f* ⟶ (*wpowerstar f*)

by (*simp add*: *WPowerstar-ext*)
have *2*: ⊢ (*wpowerstar f*) ⟶ (*wpowerstar f*);(*wpowerstar g*)
by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty int-eq*)
have *3*: ⊢ *f* ⟶ (*wpowerstar f*);(*wpowerstar g*)
using *1 2* **by** *fastforce*
have *4*: ⊢ *g* ⟶ (*wpowerstar g*)
by (*simp add*: *WPowerstar-ext*)
have *5*: ⊢ (*wpowerstar g*) ⟶ (*wpowerstar f*);(*wpowerstar g*)
by (*meson EmptyChop LeftChopImpChop Prop11 WPowerstar-imp-empty lift-imp-trans*)
have *6*: ⊢ *g* ⟶ (*wpowerstar f*);(*wpowerstar g*)
using *4 5* **by** *fastforce*
have *7*: ⊢ *f* ∨ *g* ⟶ (*wpowerstar f*);(*wpowerstar g*)
using *3 6* **by** *fastforce*
have *9*: ⊢ (*wpowerstar* (*f* ∨ *g*)) ⟶ (*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
using *7 WPowerstar-iso* **by** *blast*
have *10*: ⊢ (*wpowerstar f*);(*wpowerstar g*) ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*simp add*: *WPowerstar-subdist-var-3*)
have *11*: ⊢ (*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*))) ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*simp add*: *10 Chop-WPowerstar-Closure WPowerstar-induct-lvar-star*)
show *?thesis* **using** *11 9 int-iffI* **by** *blast*
qed

lemma *SChopstar-denest*:
⊢ (*schopstar* (*f* ∨ *g*)) = (*schopstar* ((*schopstar f*)⌢(*schopstar g*)))
proof −
have *1*: ⊢ (*f* ∧ *finite*) ⟶ (*schopstar f*)
by (*simp add*: *SChopstar-ext*)
have *2*: ⊢ (*schopstar f*) ⟶ (*schopstar f*)⌢(*schopstar g*)
by (*metis* (*no-types*, *lifting*) *AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop EmptyImpFinite FiniteAndEmptyEqvEmpty Prop02 Prop12 SChopAssoc SChopImpSChop SChopstar-1L SChopstar-imp-empty*
SChopstar-induct-lvar-empty inteq-reflection schop-d-def)
have *3*: ⊢ (*f* ∧ *finite*) ⟶ (*schopstar f*)⌢(*schopstar g*)
using *1 2* **by** *fastforce*
have *4*: ⊢ (*g* ∧ *finite*) ⟶ (*schopstar g*)
by (*simp add*: *SChopstar-ext*)
have *5*: ⊢ (*schopstar g*) ⟶ (*schopstar f*)⌢(*schopstar g*)
by (*meson EmptySChop LeftSChopImpSChop Prop11 SChopstar-imp-empty lift-imp-trans*)
have *6*: ⊢ (*g* ∧ *finite*) ⟶ (*schopstar f*)⌢(*schopstar g*)
using *4 5* **by** *fastforce*
have *7*: ⊢ (*f* ∧ *finite*) ∨ (*g* ∧ *finite*) ⟶ (*schopstar f*)⌢(*schopstar g*)
using *3 6* **by** *fastforce*
have *8*: ⊢ ((*f* ∧ *finite*) ∨ (*g* ∧ *finite*)) = ((*f* ∨ *g*) ∧ *finite*)
by *fastforce*
have *9*: ⊢ (*schopstar* (*f* ∨ *g*)) ⟶ (*schopstar* ((*schopstar f*)⌢(*schopstar g*)))
by (*metis* (*no-types*, *opaque-lifting*) *7 8 ChopEmpty EmptySChop FPowerstar-WPowerstar*
FPowerstar-more-absorb SChopAssoc SChopstar-iso inteq-reflection schop-d-def schopstar-d-def)
have *10*: ⊢ (*schopstar f*)⌢(*schopstar g*) ⟶ (*schopstar* (*f* ∨ *g*))
by (*simp add*: *SChopstar-subdist-var-3*)
have *11*: ⊢ (*schopstar* ((*schopstar f*)⌢(*schopstar g*))) ⟶ (*schopstar* (*f* ∨ *g*))

by (*simp add*: *10 SChop-SChopstar-Closure SChopstar-induct-lvar-star*)
show *?thesis* **using** *11 9 int-iffI* **by** *blast*
qed

lemma *WPowerstar-or-var*:
$\vdash$ (*wpowerstar* ((*wpowerstar f*) $\vee$ (*wpowerstar g*))) = (*wpowerstar* (*f* $\vee$ *g*))
using *WPowerstar-denest*[*of f g*]
WPowerstar-denest[*of LIFT* (*wpowerstar f*)]
WPowerstar-invol[*of f*] *WPowerstar-invol*[*of g*]
by (*metis int-eq*)

lemma *SChopstar-or-var*:
$\vdash$ (*schopstar* ((*schopstar f*) $\vee$ (*schopstar g*))) = (*schopstar* (*f* $\vee$ *g*))
by (*metis* (*no-types*, *opaque-lifting*) *FPowerstar-WPowerstar FPowerstar-more-absorb FiniteOr SChopstar-invol WPowerstar-denest inteq-reflection schopstar-d-def*)

lemma *WPowerstar-denest-var*:
$\vdash$ (*wpowerstar f*) ; (*wpowerstar* (*g*;(*wpowerstar f*))) = (*wpowerstar* (*f* $\vee$ *g*))
proof –
have *1*: $\vdash$ *empty* $\longrightarrow$ (*wpowerstar f*);(*wpowerstar* (*g*;(*wpowerstar f*)))
by (*metis AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop FiniteAndEmptyEqvEmpty WPowerstar-imp-empty inteq-reflection*)
have *2*: $\vdash$ (*f*;(*wpowerstar f*));(*wpowerstar* (*g*;(*wpowerstar f*))) $\longrightarrow$
(*wpowerstar f*);(*wpowerstar* (*g*;(*wpowerstar f*)))
by (*simp add*: *LeftChopImpChop WPowerstar-1L*)
have *3*: $\vdash$ (*g*;(*wpowerstar f*));(*wpowerstar* (*g*;(*wpowerstar f*))) $\longrightarrow$
(*wpowerstar f*);(*wpowerstar* (*g*;(*wpowerstar f*)))
by (*metis EmptyChop LeftChopImpChop WPowerstar-1L WPowerstar-imp-empty inteq-reflection lift-imp-trans*)

have *4*: $\vdash$ ((*f*;(*wpowerstar f*));(*wpowerstar* (*g*;(*wpowerstar f*))) $\vee$
(*g*;(*wpowerstar f*));(*wpowerstar* (*g*;(*wpowerstar f*)))) =
((*f* $\vee$ *g*);(*wpowerstar f*));(*wpowerstar* (*g*;(*wpowerstar f*)))
by (*metis OrChopEqv int-eq*)
have *5*: $\vdash$ *empty* $\vee$ ((*f* $\vee$ *g*);(*wpowerstar f*));(*wpowerstar* (*g*;(*wpowerstar f*))) $\longrightarrow$
(*wpowerstar f*);(*wpowerstar* (*g*;(*wpowerstar f*)))
using *1 2 3 4* **by** (*metis Prop02 int-eq*)
have *6*: $\vdash$ (*wpowerstar* (*f* $\vee$ *g*)) $\longrightarrow$ (*wpowerstar f*);(*wpowerstar* (*g*;(*wpowerstar f*)))
by (*metis 5 ChopAssoc WPowerstar-induct-lvar-empty int-eq*)
have *7*: $\vdash$ (*wpowerstar* (*g* ;(*wpowerstar f*))) $\longrightarrow$ (*wpowerstar* ((*wpowerstar g*);(*wpowerstar f*)))
by (*simp add*: *LeftChopImpChop WPowerstar-ext WPowerstar-iso*)
have *8*: $\vdash$ (*wpowerstar* (*g* ;(*wpowerstar f*))) $\longrightarrow$ (*wpowerstar* (*f* $\vee$ *g*))
by (*metis 7 WPowerstar-denest WPowerstar-swap inteq-reflection*)
have *9*: $\vdash$ (*wpowerstar f*) $\longrightarrow$ (*wpowerstar* (*f* $\vee$ *g*))
by (*simp add*: *WPowerstar-subdist*)
have *10*:$\vdash$ (*wpowerstar f*) ; (*wpowerstar* (*g*;(*wpowerstar f*))) $\longrightarrow$ (*wpowerstar* (*f* $\vee$ *g*))
using *8 9 Chop-WPowerstar-Closure* **by** *blast*
show *?thesis*
by (*simp add*: *10 6 int-iffI*)
qed

lemma *SChopstar-denest-var*:
$\vdash$ (*schopstar f*) $\frown$ (*schopstar* ((*g*)$\frown$(*schopstar f*))) = (*schopstar* (*f* $\vee$ *g*))
proof −
have *1*: $\vdash$ *empty* $\longrightarrow$ (*schopstar* ((*g*)$\frown$(*schopstar f*)))
by (*simp add*: *SChopstar-imp-empty*)
have *11*: $\vdash$ *empty* $\longrightarrow$ (*schopstar f*)$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*)))
by (*metis AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop EmptyImpFinite FiniteAndEmptyE-qvEmpty*
Prop12 SChopstar-imp-empty inteq-reflection itl-def(*9*))
have *2*: $\vdash$ ((*f*)$\frown$(*schopstar f*))$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*))) $\longrightarrow$
(*schopstar f*)$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*)))
using *LeftSChopImpSChop SChopstar-1L* **by** *blast*
have *3*: $\vdash$ ((*g*)$\frown$(*schopstar f*))$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*))) $\longrightarrow$
(*schopstar f*)$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*)))
by (*metis EmptySChop LeftSChopImpSChop SChopstar-1L SChopstar-imp-empty inteq-reflection lift-imp-trans*)
have *4*: $\vdash$ ((*f* $\vee$ *g*)$\frown$(*schopstar f*))$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*))) =
(((*f*)$\frown$(*schopstar f*))$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*))) $\vee$
((*g*)$\frown$(*schopstar f*))$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*))))

by (*metis OrSChopEqv inteq-reflection*)
have *5*: $\vdash$ *empty* $\vee$ ((*f* $\vee$ *g*)$\frown$(*schopstar f*))$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*))) $\longrightarrow$
(*schopstar f*)$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*)))
using *11 2 3 4*
by (*metis Prop02 inteq-reflection*)
have *6*: $\vdash$ (*schopstar* (*f* $\vee$ *g*)) $\longrightarrow$ (*schopstar f*)$\frown$(*schopstar* ((*g*)$\frown$(*schopstar f*)))
by (*metis 5 SChopAssoc SChopstar-induct-lvar-empty inteq-reflection*)
have *7*: $\vdash$ (*schopstar* (*g* $\frown$(*schopstar f*))) $\longrightarrow$ (*schopstar* ((*schopstar g*)$\frown$(*schopstar f*)))
by (*metis AndChopB ChopEmpty LeftChopImpChop Prop12 SChopstar-ext SChopstar-iso inteq-reflection schop-d-def*)
have *8*: $\vdash$ (*schopstar* (*g* $\frown$(*schopstar f*))) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*metis 7 SChopstar-denest SChopstar-swap inteq-reflection*)
have *9*: $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*simp add*: *SChopstar-subdist*)
have *10*:$\vdash$ (*schopstar f*) $\frown$ (*schopstar* ((*g*)$\frown$(*schopstar f*))) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*simp add*: *8 9 SChop-SChopstar-Closure*)
show *?thesis*
by (*simp add*: *10 6 int-iffI*)
qed

lemma *SChopstarMore-denest-var*:
$\vdash$ (*schopstar f*) $\frown$ (*schopstar* ((*g* $\wedge$ *more*)$\frown$(*schopstar f*))) = (*schopstar* (*f* $\vee$ *g*))
using *SChopstar-denest-var*[*of LIFT f* $\wedge$ *more LIFT g* $\wedge$ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT* (*f* $\vee$ *g*)]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-denest-var-2*:
$\vdash$ (*wpowerstar f*); (*wpowerstar* (*g*;(*wpowerstar f*))) =

(*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
by (*metis WPowerstar-denest WPowerstar-denest-var int-eq*)

lemma *SChopstar-denest-var-2*:
⊢ (*schopstar f*)⌢ (*schopstar* (*g*⌢(*schopstar f*))) = (*schopstar* ((*schopstar f*)⌢(*schopstar g*)))
by (*metis SChopstar-denest SChopstar-denest-var inteq-reflection*)

lemma *SChopstarMore-denest-var-2*:
⊢ (*schopstar f*)⌢ (*schopstar* ((*g* ∧ *more*)⌢(*schopstar f*))) = (*schopstar* ((*schopstar f*)⌢(*schopstar g*)))
using *SChopstar-denest-var-2*[*of f LIFT g* ∧ *more*] *SChopstar-and-more*[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-denest-var-3*:
⊢ (*wpowerstar f*); (*wpowerstar* ((*wpowerstar g*);(*wpowerstar f*))) =
(*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
by (*metis WPowerstar-denest WPowerstar-denest-var WPowerstar-ext WPowerstar-induct-lvar-star WPowerstar-trans int-iffI inteq-reflection*)

lemma *SChopstar-denest-var-3*:
⊢ (*schopstar f*)⌢ (*schopstar* ((*schopstar g*)⌢(*schopstar f*))) = (*schopstar* ((*schopstar f*)⌢(*schopstar g*)))
by (*metis SChopstar-denest-var-2 SChopstar-invol int-eq*)

lemma *WPowerstar-denest-var-4*:
⊢ (*wpowerstar* ((*wpowerstar g*);(*wpowerstar f*))) =
(*wpowerstar* ((*wpowerstar f*);(*wpowerstar g*)))
by (*metis WPowerstar-denest WPowerstar-swap inteq-reflection*)

lemma *SChopstar-denest-var-4*:
⊢ (*schopstar* ((*schopstar g*)⌢(*schopstar f*))) = (*schopstar* ((*schopstar f*)⌢(*schopstar g*)))
by (*metis SChopstar-denest SChopstar-swap inteq-reflection*)

lemma *WPowerstar-denest-var-5*:
⊢ (*wpowerstar f*); (*wpowerstar* (*g*;(*wpowerstar f*))) =
(*wpowerstar g*); (*wpowerstar* (*f*;(*wpowerstar g*)))
by (*metis WPowerstar-denest-var WPowerstar-swap int-eq*)

lemma *SChopstar-denest-var-5*:
⊢ (*schopstar f*)⌢ (*schopstar* (*g*⌢(*schopstar f*))) = (*schopstar g*)⌢ (*schopstar* (*f*⌢(*schopstar g*)))
by (*metis SChopstar-denest-var SChopstar-swap inteq-reflection*)

lemma *SChopstarMore-denest-var-5*:
⊢ (*schopstar f*)⌢ (*schopstar* ((*g* ∧ *more*)⌢(*schopstar f*))) = (*schopstar g*)⌢ (*schopstar* (*f*⌢(*schopstar g*)))

using *SChopstar-denest-var-5*[*of f LIFT g* ∧ *more*]
SChopstar-and-more[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-denest-var-6*:
$\vdash ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g));(\mathit{wpowerstar}\ (f \vee g)) = (\mathit{wpowerstar}\ (f \vee g))$
by (*metis ChopAssoc WPowerstar-denest WPowerstar-denest-var-3 inteq-reflection*)

lemma *SChopstar-denest-var-6*:
$\vdash ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g))\frown(\mathit{schopstar}\ (f \vee g)) = (\mathit{schopstar}\ (f \vee g))$
by (*metis SChopAssoc SChopstar-denest SChopstar-denest-var-3 inteq-reflection*)

lemma *WPowerstar-denest-var-7*:
$\vdash (\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)) = (\mathit{wpowerstar}\ (f \vee g))$
proof –
have *1*: $\vdash (\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)) \longrightarrow$
$(\mathit{wpowerstar}\ (f \vee g))\ ;\ (\mathit{wpowerstar}\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)))$
by (*simp add*: *RightChopImpChop WPowerstar-ext*)
have *2*: $\vdash (\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)) \longrightarrow (\mathit{wpowerstar}\ (f \vee g))$
by (*simp add*: *Chop-WPowerstar-Closure WPowerstar-subdist-var-3*)
have *3*: $\vdash \mathit{empty} \longrightarrow (\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g))$
by (*metis ChopEmpty ChopImpChop WPowerstar-imp-empty inteq-reflection*)
have *4*: $\vdash (f \vee g)\ ;\ ((\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g))) \longrightarrow$
$((\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)))$
by (*metis ChopAssoc LeftChopImpChop WPowerstar-1L inteq-reflection*)
have *5*: $\vdash \mathit{empty} \vee (f \vee g)\ ;\ ((\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g))) \longrightarrow$
$((\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)))$
using *3 4 Prop02* **by** *blast*
have *6*: $\vdash (\mathit{wpowerstar}\ (f \vee g)) \longrightarrow ((\mathit{wpowerstar}\ (f \vee g))\ ;\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)))$
using *5 WPowerstar-induct-lvar-empty* **by** *blast*
show *?thesis* **using** *2 6 int-iffI* **by** *blast*
qed

lemma *SChopstar-denest-var-7*:
$\vdash (\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)) = (\mathit{schopstar}\ (f \vee g))$
proof –
have *1*: $\vdash (\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)) \longrightarrow$
$(\mathit{schopstar}\ (f \vee g)) \frown (\mathit{schopstar}\ ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)))$
by (*meson RightSChopImpSChop SChopstar-denest SChopstar-subdist-var-3 int-iffD1 lift-imp-trans*)
have *2*: $\vdash (\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)) \longrightarrow (\mathit{schopstar}\ (f \vee g))$
by (*simp add*: *SChop-SChopstar-Closure SChopstar-subdist-var-3*)
have *3*: $\vdash \mathit{empty} \longrightarrow (\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g))$
by (*metis EmptySChop SChopImpSChop SChopstar-imp-empty inteq-reflection*)
have *4*: $\vdash (f \vee g) \frown ((\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g))) \longrightarrow$
$((\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)))$
by (*metis LeftSChopImpSChop SChopAssoc SChopstar-1L inteq-reflection*)
have *5*: $\vdash \mathit{empty} \vee (f \vee g) \frown ((\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g))) \longrightarrow$
$((\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)))$
using *3 4 Prop02* **by** *blast*
have *6*: $\vdash (\mathit{schopstar}\ (f \vee g)) \longrightarrow ((\mathit{schopstar}\ (f \vee g)) \frown ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)))$
using *5 SChopstar-induct-lvar-empty* **by** *blast*
show *?thesis* **using** *2 6 int-iffI* **by** *blast*

qed

lemma *WPowerstar-denest-var-8*:
$\vdash ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g));(\mathit{wpowerstar}\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g))) =$
$(\mathit{wpowerstar}\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)))$
by (*metis ChopAssoc WPowerstar-denest-var-3 int-eq*)

lemma *SChopstar-denest-var-8*:
$\vdash ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g))\frown(\mathit{schopstar}\ ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g))) =$
$(\mathit{schopstar}\ ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)))$
by (*metis SChopstar-denest SChopstar-denest-var-6 inteq-reflection*)

lemma *WPowerstar-denest-var-9*:
$\vdash (\mathit{wpowerstar}\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)));((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)) =$
$(\mathit{wpowerstar}\ ((\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)))$
by (*metis WPowerstar-denest WPowerstar-denest-var-7 inteq-reflection*)

lemma *SChopstar-denest-var-9*:
$\vdash (\mathit{schopstar}\ ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)))\frown((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)) =$
$(\mathit{schopstar}\ ((\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)))$
by (*metis SChopstar-denest SChopstar-denest-var-7 inteq-reflection*)

lemma *WPowerstar-confluence*:
$(\vdash g\ ;\ (\mathit{wpowerstar}\ f) \longrightarrow (\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)) =$
$(\vdash (\mathit{wpowerstar}\ g)\ ;\ (\mathit{wpowerstar}\ f) \longrightarrow (\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)\ )$
proof –
have *1*: $(\vdash g\ ;\ (\mathit{wpowerstar}\ f) \longrightarrow (\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)) \Longrightarrow$
$(\vdash (\mathit{wpowerstar}\ g)\ ;\ (\mathit{wpowerstar}\ f) \longrightarrow (\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)\ )$
by (*metis Prop02 WPowerstar-imp-empty WPowerstar-rtc2 WPowerstar-sim1 WPowerstar-trans inteq-reflection*)
have *2*: $(\vdash (\mathit{wpowerstar}\ g)\ ;\ (\mathit{wpowerstar}\ f) \longrightarrow (\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g)\ ) \Longrightarrow$
$(\vdash g\ ;\ (\mathit{wpowerstar}\ f) \longrightarrow (\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ g))$
by (*metis AndChopB Prop10 WPowerstar-ext inteq-reflection lift-imp-trans*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *SChopstar-finite*:
$\vdash \mathit{schopstar}\ f \longrightarrow \mathit{finite}$
by (*metis EmptyImpFinite FiniteChopEqvDiamond FiniteChopFiniteEqvFinite Prop02 SChopImpDiamond*

SChopstar-induct-lvar-empty inteq-reflection)

lemma *SChopstar-confluence*:
$(\vdash g \frown (\mathit{schopstar}\ f) \longrightarrow (\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)) =$
$(\vdash (\mathit{schopstar}\ g) \frown (\mathit{schopstar}\ f) \longrightarrow (\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)\ )$
proof –
have *1*: $(\vdash g \frown (\mathit{schopstar}\ f) \longrightarrow (\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)) \Longrightarrow$
$(\vdash (\mathit{schopstar}\ g) \frown (\mathit{schopstar}\ f) \longrightarrow (\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ g)\ )$

using *SChopstar-sim1*[*of g LIFT* (*schopstar f*) *LIFT* (*schopstar g*)]
by (*metis Prop10 SChopstar-finite SChopstar-invol inteq-reflection*)
have *2*: $(\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)\ ) \Longrightarrow$
$(\vdash g \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g))$
by (*metis AndChopB Prop10 SChopstar-ext SChopstar-finite int-eq lift-imp-trans schop-d-def*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *SChopstarMore-confluence*:
$(\vdash (g \wedge more) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)) =$
$(\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)\ )$
using *SChopstar-confluence*[*of LIFT g* ∧ *more f*]
SChopstar-and-more[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-church-rosser*:
assumes $\vdash (wpowerstar\ g)\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
shows $\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
proof −
have *1*: $\vdash ((wpowerstar\ f)\ ;\ (wpowerstar\ g));\ ((wpowerstar\ f)\ ;\ (wpowerstar\ g)) \longrightarrow$
$((wpowerstar\ f)\ ;\ (wpowerstar\ f));\ ((wpowerstar\ g)\ ;\ (wpowerstar\ g))$
by (*metis* (*no-types*, *lifting*) *ChopAssoc ChopImpChop WPowerstar-trans WPowerstar-trans-eq assms int-eq*)
have *2*: $\vdash ((wpowerstar\ f)\ ;\ (wpowerstar\ g));\ ((wpowerstar\ f)\ ;\ (wpowerstar\ g)) \longrightarrow$
$((wpowerstar\ f)\ ;\ (wpowerstar\ g))$
by (*metis 1 WPowerstar-trans-eq inteq-reflection*)
have *3*: $\vdash empty \longrightarrow ((wpowerstar\ f)\ ;\ (wpowerstar\ g))$
by (*metis 2 WPowerstar-denest-var-9 WPowerstar-imp-empty WPowerstar-induct-lvar int-eq lift-imp-trans*)
have *4*: $\vdash empty \vee ((wpowerstar\ f)\ ;\ (wpowerstar\ g));\ ((wpowerstar\ f)\ ;\ (wpowerstar\ g)) \longrightarrow$
$((wpowerstar\ f)\ ;\ (wpowerstar\ g))$
using *2 3 Prop02* **by** *blast*
have *5*: $\vdash (wpowerstar\ ((wpowerstar\ f)\ ;\ (wpowerstar\ g))) \longrightarrow$
$((wpowerstar\ f)\ ;\ (wpowerstar\ g))$
using *4 WPowerstar-induct-lvar-empty* **by** *blast*
have *6*: $\vdash (wpowerstar\ (f \vee g)) \longrightarrow (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
by (*metis 5 WPowerstar-denest inteq-reflection*)
have *7*: $\vdash (wpowerstar\ f)\ ;\ (wpowerstar\ g) \longrightarrow (wpowerstar\ (f \vee g))$
by (*simp add*: *WPowerstar-subdist-var-3*)
show *?thesis*
by (*simp add*: *6 7 int-iffI*)
qed

lemma *SChopstar-church-rosser*:
assumes $\vdash (schopstar\ g) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
shows $\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ g)$
proof −
have *0*: $\vdash ((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f)) \longrightarrow$
$((schopstar\ f) \frown (schopstar\ f)) \frown ((schopstar\ g))$
by (*metis RightSChopImpSChop SChopAssoc assms inteq-reflection*)
have *01*: $\vdash (((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f))) \frown (schopstar\ g) \longrightarrow$

$(((schopstar\ f) \frown (schopstar\ f)) \frown ((schopstar\ g))) \frown (schopstar\ g)$
by (*simp add*: *0 LeftSChopImpSChop*)
have *1*: $\vdash ((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f) \frown (schopstar\ g)) \longrightarrow$
$((schopstar\ f) \frown (schopstar\ f)) \frown ((schopstar\ g) \frown (schopstar\ g))$
by (*metis 01 SChopAssoc inteq-reflection*)
have *2*: $\vdash ((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f) \frown (schopstar\ g)) \longrightarrow$
$((schopstar\ f) \frown (schopstar\ g))$
by (*metis* (*no-types*, *lifting*) *1 SChopstar-denest SChopstar-denest-var-3 int-eq int-simps*(*27*))
have *3*: $\vdash empty \longrightarrow ((schopstar\ f) \frown (schopstar\ g))$
by (*meson EmptySChop Prop11 SChopImpSChop SChopstar-imp-empty lift-imp-trans*)
have *4*: $\vdash empty \vee ((schopstar\ f) \frown (schopstar\ g)) \frown ((schopstar\ f) \frown (schopstar\ g)) \longrightarrow$
$((schopstar\ f) \frown (schopstar\ g))$
using *2 3 Prop02* **by** *blast*
have *5*: $\vdash (schopstar\ ((schopstar\ f) \frown (schopstar\ g))) \longrightarrow ((schopstar\ f) \frown (schopstar\ g))$
using *4 SChopstar-induct-lvar-empty* **by** *blast*
have *6*: $\vdash (schopstar\ (f \vee g)) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
by (*metis 5 SChopstar-denest inteq-reflection*)
have *7*: $\vdash (schopstar\ f) \frown (schopstar\ g) \longrightarrow (schopstar\ (f \vee g))$
by (*simp add*: *SChopstar-subdist-var-3*)
show *?thesis*
by (*simp add*: *6 7 int-iffI*)
qed

lemma *WPowerstar-church-rosser-var*:
assumes $\vdash g\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
shows $\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
using *assms*
by (*simp add*: *WPowerstar-church-rosser WPowerstar-confluence*)

lemma *SChopstar-church-rosser-var*:
assumes $\vdash g \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
shows $\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ g)$
using *assms*
using *SChopstar-church-rosser SChopstar-confluence* **by** *blast*

lemma *SChopstarMore-church-rosser-var*:
assumes $\vdash (g \wedge more) \frown (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$
shows $\vdash (schopstar\ (f \vee g)) = (schopstar\ f) \frown (schopstar\ g)$
using *assms SChopstar-church-rosser-var*[*of LIFT g* $\wedge$ *more LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of g*] *SChopstar-and-more*[*of f*]
SChopstar-and-more[*of LIFT* $(f \vee g)$]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-church-rosser-to-confluence*:
assumes $\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
shows $\vdash (wpowerstar\ g)\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ f)\ ;\ (wpowerstar\ g)$
using *assms*
by (*metis WPowerstar-subdist-var-3 WPowerstar-swap inteq-reflection*)

lemma *SChopstar-church-rosser-to-confluence*:
assumes $\vdash$ (*schopstar* ($f \vee g$)) = (*schopstar* f) $\frown$ (*schopstar* g)
shows $\vdash$ (*schopstar* g) $\frown$ (*schopstar* f) $\longrightarrow$ (*schopstar* f) $\frown$ (*schopstar* g)
using *assms*
by (*metis SChopstar-subdist-var-3 SChopstar-swap inteq-reflection*)

lemma *WPowerstar-church-rosser-equiv*:
($\vdash$ (*wpowerstar* g) ; (*wpowerstar* f) $\longrightarrow$ (*wpowerstar* f) ; (*wpowerstar* g)) =
($\vdash$ (*wpowerstar* ($f \vee g$)) = (*wpowerstar* f) ; (*wpowerstar* g))
using *WPowerstar-church-rosser WPowerstar-church-rosser-to-confluence* **by** *blast*

lemma *SChopstar-church-rosser-equiv*:
($\vdash$ (*schopstar* g) $\frown$ (*schopstar* f) $\longrightarrow$ (*schopstar* f) $\frown$ (*schopstar* g)) =
($\vdash$ (*schopstar* ($f \vee g$)) = (*schopstar* f) $\frown$ (*schopstar* g))
using *SChopstar-church-rosser SChopstar-church-rosser-to-confluence* **by** *blast*

lemma *WPowerstar-confluence-to-local-confluence*:
assumes $\vdash$ (*wpowerstar* g) ; (*wpowerstar* f) $\longrightarrow$ (*wpowerstar* f) ; (*wpowerstar* g)
shows $\vdash$ g ; f $\longrightarrow$ (*wpowerstar* f) ; (*wpowerstar* g)
using *assms*
WPowerstar-church-rosser[*of g f*]
by (*metis WPowerstar-denest WPowerstar-denest-var-4 WPowerstar-subdist-var-2 inteq-reflection*)

lemma *SChopstar-confluence-to-local-confluence*:
assumes $\vdash$ (*schopstar* g) $\frown$ (*schopstar* f) $\longrightarrow$ (*schopstar* f) $\frown$ (*schopstar* g)
shows $\vdash$ (g) $\frown$ ($f \wedge$ *finite*) $\longrightarrow$ (*schopstar* f) $\frown$ (*schopstar* g)
using *assms*
SChopstar-church-rosser[*of g f*]
by (*metis SChopstar-denest SChopstar-denest-var-4 SChopstar-subdist-var-2 inteq-reflection*)

lemma *SChopstarMore-confluence-to-local-confluence*:
assumes $\vdash$ (*schopstar* g) $\frown$ (*schopstar* f) $\longrightarrow$ (*schopstar* f) $\frown$ (*schopstar* g)
shows $\vdash$ ($g \wedge$ *more*) $\frown$ ($f \wedge$ *finite*) $\longrightarrow$ (*schopstar* f) $\frown$ (*schopstar* g)
using *assms*
SChopstar-confluence-to-local-confluence[*of LIFT g* $\wedge$ *more f*]
SChopstar-and-more[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-sup-id-star1*:
assumes $\vdash$ *empty* $\longrightarrow$ f
shows $\vdash$ f;(*wpowerstar* f) = (*wpowerstar* f)
using *assms*
by (*metis* (*no-types, lifting*) *AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop ChopOrEqv ChopOrImp*

FiniteAndEmptyEqvEmpty Prop02 WPowerstarEqv WPowerstar-1L WPowerstar-imp-empty int-iffI inteq-reflection)

lemma *SChopstar-sup-id-star1*:
assumes $\vdash$ *empty* $\longrightarrow$ *f*
shows $\vdash$ *f*$\frown$(*schopstar f*) = (*schopstar f*)
using *assms*
by (*metis EmptyImpFinite Prop12 SChopstar-WPowerstar WPowerstar-sup-id-star1 inteq-reflection schop-d-def*)

lemma *WPowerstar-sup-id-star2*:
assumes $\vdash$ *empty* $\longrightarrow$ *f*
shows $\vdash$ (*wpowerstar f*);*f* = (*wpowerstar f*)
using *assms*
by (*metis ChopEmpty ChopImpChop WPowerstar-1R int-eq int-iffD2 int-iffI*)

lemma *SChopstar-sup-id-star2*:
assumes $\vdash$ *empty* $\longrightarrow$ *f*
shows $\vdash$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*) = (*schopstar f*)
using *assms*
WPowerstar-sup-id-star2[*of LIFT* (*f* $\wedge$ *finite*)] *SChopstar-WPowerstar*[*of f*]
by (*metis EmptyImpFinite Prop10 Prop12 SChopstar-finite inteq-reflection schop-d-def*)

lemma *WPowerstar-unfoldr-eq*:
$\vdash$ (*empty* $\vee$ (*wpowerstar f*);*f*) = (*wpowerstar f*)
proof –
have *1*: $\vdash$ (*empty* $\vee$ (*wpowerstar f*);*f*) $\longrightarrow$ (*wpowerstar f*)
using *WPowerstar-unfoldR* **by** *auto*
have *2*: $\vdash$ (*empty* $\vee$ *f*;(*empty* $\vee$ (*wpowerstar f*);*f*)) =
(*empty* $\vee$ (*empty* $\vee$ *f*;(*wpowerstar f*));*f*)
by (*metis* (*no-types*, *lifting*) *ChopAssoc ChopEmpty ChopOrEqv EmptyOrChopEqv inteq-reflection*)
have *3*: $\vdash$ (*empty* $\vee$ (*empty* $\vee$ *f*;(*wpowerstar f*));*f*) =
(*empty* $\vee$ (*wpowerstar f*);*f*)
by (*metis 2 inteq-reflection wpowerstar-unfoldl-eq*)
have *4*: $\vdash$ (*wpowerstar f*) $\longrightarrow$ (*empty* $\vee$ (*wpowerstar f*);*f*)
by (*metis 2 3 WPowerstar-induct-lvar-empty int-eq int-iffD1*)
show *?thesis*
using *1 4 int-iffI* **by** *blast*
qed

lemma *SChopstar-unfoldr-eq*:
$\vdash$ (*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*)) = (*schopstar f*)
proof –
have *1*: $\vdash$ (*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*)) $\longrightarrow$ (*schopstar f*)
using *SChopstar-unfoldR* **by** *auto*
have *2*: $\vdash$ (*empty* $\vee$ *f*$\frown$(*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*))) =
(*empty* $\vee$ (*empty* $\vee$ *f*$\frown$(*schopstar f*))$\frown$(*f* $\wedge$ *finite*))
by (*metis* (*no-types*, *lifting*) *ChopEmpty EmptyOrSChopEqv SChopAssoc SChopOrEqv inteq-reflection itl-def*(*9*))
have *3*: $\vdash$ (*empty* $\vee$ (*empty* $\vee$ *f*$\frown$(*schopstar f*))$\frown$(*f* $\wedge$ *finite*)) =
(*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*))
by (*metis 2 SChopstar-unfoldl-eq inteq-reflection*)
have *4*: $\vdash$ (*schopstar f*) $\longrightarrow$ (*empty* $\vee$ (*schopstar f*)$\frown$(*f* $\wedge$ *finite*))
by (*metis 2 3 SChopstar-induct-lvar-empty int-eq int-iffD1*)

show *?thesis*
using *1 4 int-iffI* **by** *blast*
qed

lemma *SChopstarMore-unfoldr-eq*:
$\vdash$ (*empty* $\vee$ (*schopstar f*)$\frown$((*f* $\wedge$ *more*) $\wedge$ *finite*)) = (*schopstar f*)
using *SChopstar-unfoldr-eq*[*of LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-star-prod-unfold*:
$\vdash$ (*empty* $\vee$ *f*;((*wpowerstar* (*g*;*f*));*g*)) = (*wpowerstar* (*f* ; *g*))
proof –
have *1*: $\vdash$ (*wpowerstar* (*f* ; *g*)) = (*empty* $\vee$ (*wpowerstar* (*f*;*g*));(*f*;*g*))
by (*meson WPowerstar-unfoldR WPowerstar-unfoldr-eq int-iffD2 int-iffI*)
have *2*: $\vdash$ (*wpowerstar* (*f*;*g*));(*f*;*g*) $\longrightarrow$ *f*;((*wpowerstar* (*g*;*f*));*g*)
using *WPowerstar-slide1*[*of f g*]
by (*metis AndChopB ChopAssoc Prop10 int-eq*)
have *3*: $\vdash$ (*wpowerstar* (*f* ; *g*)) $\longrightarrow$ (*empty* $\vee$ *f*;((*wpowerstar* (*g*;*f*));*g*))
using *1 2* **by** *fastforce*
have *4*: $\vdash$ *f*;((*wpowerstar* (*g*;*f*));*g*) $\longrightarrow$ (*f*;*g*);(*wpowerstar* (*f* ; *g*))
using *WPowerstar-slide1*[*of g f*]
by (*metis ChopAssoc RightChopImpChop inteq-reflection*)
have *5*: $\vdash$ (*empty* $\vee$ *f*;((*wpowerstar* (*g*;*f*));*g*)) $\longrightarrow$
empty $\vee$ (*f*;*g*);(*wpowerstar* (*f* ; *g*))
using *4* **by** *fastforce*
have *6*: $\vdash$ (*empty* $\vee$ (*f*;*g*);(*wpowerstar* (*f* ; *g*))) $\longrightarrow$ (*wpowerstar* (*f* ; *g*))
by (*meson WPowerstarEqv int-iffD2*)
show *?thesis*
by (*meson 3 5 6 int-iffI lift-imp-trans*)
qed

lemma *FiniteImportSChopRight*:
$\vdash$ (*finite* $\wedge$ (*f* $\frown$ *g*)) = *f*$\frown$(*g* $\wedge$ *finite*)
by (*metis ChopEmpty SChopAssoc inteq-reflection lift-and-com schop-d-def*)

lemma *SChopstar-star-prod-unfold*:
$\vdash$ (*empty* $\vee$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*))) = (*schopstar* (*f* $\frown$ *g*))
proof –
have *1*: $\vdash$ (*schopstar* (*f* $\frown$ *g*)) = (*empty* $\vee$ (*schopstar* (*f*$\frown$*g*))$\frown$(*f*$\frown$(*g* $\wedge$ *finite*)))
by (*metis FiniteImportSChopRight SChopAndCommute SChopstar-unfoldr-eq inteq-reflection*)
have *2*: $\vdash$ (*schopstar* (*f*$\frown$*g*))$\frown$(*f*$\frown$(*g* $\wedge$ *finite*)) $\longrightarrow$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*))
using *SChopstar-slide1*[*of f g*]
by (*metis* (*no-types*, *opaque-lifting*) *ChopAndFiniteDist ChopEmpty LeftSChopImpSChop SChopAssoc inteq-reflection schop-d-def*)
have *3*: $\vdash$ (*schopstar* (*f* $\frown$ *g*)) $\longrightarrow$ (*empty* $\vee$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*)))
using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*)) $\longrightarrow$ (*f*$\frown$*g*)$\frown$(*schopstar* (*f* $\frown$ *g*))
using *SChopstar-slide1*[*of g f*]

by (*metis RightSChopImpSChop SChopAssoc inteq-reflection*)
have *5*: $\vdash$ (*empty* $\vee$ (*f*)$\frown$((*schopstar* (*g*$\frown$*f*))$\frown$(*g* $\wedge$ *finite*))) $\longrightarrow$
empty $\vee$ (*f*$\frown$*g*)$\frown$(*schopstar* (*f* $\frown$ *g*))
using *4* **by** *fastforce*
have *6*: $\vdash$ (*empty* $\vee$ (*f*$\frown$*g*)$\frown$(*schopstar* (*f* $\frown$ *g*))) $\longrightarrow$ (*schopstar* (*f* $\frown$ *g*))
by (*meson Prop02 SChopstar-1L SChopstar-imp-empty*)
show *?thesis*
by (*meson 3 5 6 int-iffI lift-imp-trans*)
qed

lemma *WPowerstar-slide* :
$\vdash$ (*wpowerstar* (*f*;*g*));*f* = *f*;(*wpowerstar* (*g*;*f*))
proof –
have *1*: $\vdash$ *f*;(*wpowerstar* (*g*;*f*)) = *f*;(*empty* $\vee$ *g*;((*wpowerstar* (*f*;*g*));*f*))
by (*metis RightChopEqvChop WPowerstar-star-prod-unfold inteq-reflection*)
have *2*: $\vdash$ *f*;(*empty* $\vee$ *g*;((*wpowerstar* (*f*;*g*));*f*)) =
(*f*;*empty* $\vee$ *f*;(*g*;((*wpowerstar* (*f*;*g*));*f*)))
by (*simp add*: *ChopOrEqv*)
have *3*: $\vdash$ *f*;*empty* = *empty*;*f*
by (*metis ChopEmpty EmptyChop int-eq*)
have *4*: $\vdash$ *f*;(*g*;((*wpowerstar* (*f*;*g*));*f*)) =
((*f*;*g*);(*wpowerstar* (*f*;*g*)));*f*
by (*metis ChopAssoc int-eq*)
have *5*: $\vdash$ (*f*;*empty* $\vee$ *f*;(*g*;((*wpowerstar* (*f*;*g*));*f*))) =
(*empty*;*f* $\vee$ ((*f*;*g*);(*wpowerstar* (*f*;*g*)));*f*)
using *3 4* **by** *fastforce*
have *6*: $\vdash$ (*empty*;*f* $\vee$ ((*f*;*g*);(*wpowerstar* (*f*;*g*)));*f*) =
(*empty* $\vee$ ((*f*;*g*);(*wpowerstar* (*f*;*g*))));*f*
by (*meson OrChopEqv Prop11*)
have *7*: $\vdash$ *f*;(*empty* $\vee$ *g*;((*wpowerstar* (*f*;*g*));*f*)) =
(*empty* $\vee$ (*f*;*g*);(*wpowerstar* (*f*;*g*)));*f*
by (*metis 2 3 4 6 inteq-reflection*)
have *8*: $\vdash$ (*empty* $\vee$ (*f*;*g*);(*wpowerstar* (*f*;*g*))) = (*wpowerstar* (*f*;*g*))
by (*simp add*: *wpowerstar-unfoldl-eq*)
show *?thesis* **by** (*metis 1 7 8 int-eq*)
qed

lemma *SChopstar-slide* :
$\vdash$ (*schopstar* (*f*$\frown$*g*))$\frown$(*f* $\wedge$ *finite*) = *f*$\frown$(*schopstar* (*g*$\frown$*f*))
proof –
have *1*: $\vdash$ *f*$\frown$(*schopstar* (*g*$\frown$*f*)) = *f*$\frown$(*empty* $\vee$ *g*$\frown$((*schopstar* (*f*$\frown$*g*))$\frown$(*f* $\wedge$ *finite*)))
by (*metis RightSChopEqvSChop SChopstar-star-prod-unfold inteq-reflection*)
have *2*: $\vdash$ *f*$\frown$(*empty* $\vee$ *g*$\frown$((*schopstar* (*f*$\frown$*g*))$\frown$(*f* $\wedge$ *finite*))) =
(*f*$\frown$*empty* $\vee$ *f*$\frown$(*g*$\frown$((*schopstar* (*f*$\frown$*g*))$\frown$(*f* $\wedge$ *finite*))))
by (*simp add*: *SChopOrEqv*)
have *3*: $\vdash$ *f*$\frown$*empty* = *empty*$\frown$(*f* $\wedge$ *finite*)
by (*metis DiamondEmptyEqvFinite EmptySChop FiniteAndEmptyEqvEmpty SChopAndCommute SChopAndEmptyEqvSChopAndEmpty TrueSChopEqvDiamond inteq-reflection*)
have *4*: $\vdash$ *f*$\frown$(*g*$\frown$((*schopstar* (*f*$\frown$*g*))$\frown$(*f* $\wedge$ *finite*))) =
((*f*$\frown$*g*)$\frown$(*schopstar* (*f*$\frown$*g*)))$\frown$(*f* $\wedge$ *finite*)

by (*metis SChopAssoc inteq-reflection*)
have *5*: $\vdash$ ($f\frown empty \vee f\frown(g\frown((schopstar\ (f\frown g))\frown(f \wedge finite))))$ =
($empty\frown(f \wedge finite) \vee ((f\frown g)\frown(schopstar\ (f\frown g)))\frown(f \wedge finite)$)
using *3 4* **by** *fastforce*
have *6*: $\vdash$ ($empty\frown(f \wedge finite) \vee ((f\frown g)\frown(schopstar\ (f\frown g)))\frown(f \wedge finite)$) =
($empty \vee ((f\frown g)\frown(schopstar\ (f\frown g))))\frown(f \wedge finite)$
by (*meson OrSChopEqv Prop11*)
have *7*: $\vdash f\frown(empty \vee g\frown((schopstar\ (f\frown g))\frown(f \wedge finite)))$ =
($empty \vee (f\frown g)\frown(schopstar\ (f\frown g)))\frown(f \wedge finite)$
by (*metis 2 3 4 6 inteq-reflection*)
have *8*: $\vdash$ ($empty \vee (f\frown g)\frown(schopstar\ (f\frown g))) = (schopstar\ (f\frown g)$)
by (*simp add*: *SChopstar-unfoldl-eq*)
show *?thesis* **by** (*metis 1 7 8 int-eq*)
qed

lemma *WPowerstar-slide-var* :
$\vdash$ (*wpowerstar f*) ; $f = f$;(*wpowerstar f*)
by (*metis EmptyOrChopEqv Prop11 WPowerstarInductR WPowerstar-slide1-var1 WPowerstar-unfoldr-eq int-eq*)

lemma *SChopstar-slide-var* :
$\vdash$ (*schopstar f*) $\frown$ ($f \wedge finite$) = $f\frown$(*schopstar f*)
using *WPowerstar-slide-var*
by (*metis* (*no-types*, *lifting*) *FPowerstar-WPowerstar Prop10 SChopstar-finite SChopstar-WPowerstar int-eq schop-d-def*)

lemma *SChopstarMore-slide-var* :
$\vdash$ (*schopstar f*) $\frown$ (($f \wedge more$) $\wedge finite$) = ($f \wedge more$)$\frown$(*schopstar f*)
using *SChopstar-slide-var*[*of LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-or-unfold-var*:
$\vdash$ ($empty \vee$ (*wpowerstar f*);((*wpowerstar* ($f \vee g$));(*wpowerstar g*))) = (*wpowerstar* ($f \vee g$))
by (*metis ChopAssoc WPowerstar-denest WPowerstar-denest-var-4 WPowerstar-slide inteq-reflection wpowerstar-unfoldl-eq*)

lemma *SChopstar-or-unfold-var*:
$\vdash$ ($empty \vee (schopstar\ f)\frown((schopstar\ (f \vee g))\frown(schopstar\ g))) = (schopstar\ (f \vee g)$)
proof –
have *1*: $\vdash$ ($schopstar\ f)\frown((schopstar\ (f \vee g))\frown(schopstar\ g)$) =
($schopstar\ f)\frown((schopstar\ (f \vee g))\frown((schopstar\ g) \wedge finite)$)
by (*simp add*: *Prop10 RightSChopEqvSChop SChopstar-finite*)
have *2*: $\vdash$ ($schopstar\ f)\frown((schopstar\ (f \vee g))\frown((schopstar\ g) \wedge finite)$) =
($schopstar\ (f \vee g)$)
by (*metis* (*no-types*, *lifting*) *SChopstar-denest-var SChopstar-denest-var-2 SChopstar-denest-var-3 SChopstar-slide SChopstar-swap int-eq*)
show *?thesis*
using *1 2 SChopstar-imp-empty* **by** *fastforce*
qed

lemma *WPowerstar-or-unfold*:
$\vdash ((\mathit{wpowerstar}\ f) \lor (\mathit{wpowerstar}\ f);(g;(\mathit{wpowerstar}\ (f \lor g)))) = (\mathit{wpowerstar}\ (f \lor g))$
proof –
have *1*: $\vdash (\mathit{wpowerstar}\ (f \lor g)) = (\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ (g;(\mathit{wpowerstar}\ f)))$
by (*meson Prop11 WPowerstar-denest-var*)
have *2*: $\vdash (\mathit{wpowerstar}\ f);(\mathit{wpowerstar}\ (g;(\mathit{wpowerstar}\ f))) =$
$(\mathit{wpowerstar}\ f);(\mathit{empty} \lor (g;(\mathit{wpowerstar}\ f));(\mathit{wpowerstar}\ (g;(\mathit{wpowerstar}\ f))))$
using *RightChopEqvChop WPowerstarEqv* **by** *blast*
have *3*: $\vdash (\mathit{wpowerstar}\ f);(\mathit{empty} \lor (g;(\mathit{wpowerstar}\ f));(\mathit{wpowerstar}\ (g;(\mathit{wpowerstar}\ f)))) =$
$(\mathit{wpowerstar}\ f);(\mathit{empty} \lor g;(\mathit{wpowerstar}\ (f \lor g)))$
by (*metis 1 2 ChopAssoc int-eq*)
have *4*: $\vdash (\mathit{wpowerstar}\ f);(\mathit{empty} \lor g;(\mathit{wpowerstar}\ (f \lor g))) =$
$((\mathit{wpowerstar}\ f);\mathit{empty} \lor (\mathit{wpowerstar}\ f);(g;(\mathit{wpowerstar}\ (f \lor g))))$
using *ChopOrEqv* **by** *blast*
have *5*: $\vdash (\mathit{wpowerstar}\ f);\mathit{empty} = (\mathit{wpowerstar}\ f)$
by (*simp add*: *ChopEmpty*)
have *6*: $\vdash ((\mathit{wpowerstar}\ f);\mathit{empty} \lor (\mathit{wpowerstar}\ f);(g;(\mathit{wpowerstar}\ (f \lor g)))) =$
$((\mathit{wpowerstar}\ f) \lor (\mathit{wpowerstar}\ f);(g;(\mathit{wpowerstar}\ (f \lor g))))$
using *5* **by** *auto*
show *?thesis*
by (*metis 1 2 3 4 6 int-eq*)
qed

lemma *SChopstar-or-unfold*:
$\vdash ((\mathit{schopstar}\ f) \lor (\mathit{schopstar}\ f)\frown(g\frown(\mathit{schopstar}\ (f \lor g)))) = (\mathit{schopstar}\ (f \lor g))$
proof –
have *1*: $\vdash (\mathit{schopstar}\ (f \lor g)) = (\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ (g\frown(\mathit{schopstar}\ f)))$
by (*meson Prop11 SChopstar-denest-var*)
have *2*: $\vdash (\mathit{schopstar}\ f)\frown(\mathit{schopstar}\ (g\frown(\mathit{schopstar}\ f))) =$
$(\mathit{schopstar}\ f)\frown(\mathit{empty} \lor (g\frown(\mathit{schopstar}\ f))\frown(\mathit{schopstar}\ (g\frown(\mathit{schopstar}\ f))))$
by (*meson Prop11 RightSChopEqvSChop SChopstar-unfoldl-eq*)
have *3*: $\vdash (\mathit{schopstar}\ f)\frown(\mathit{empty} \lor (g\frown(\mathit{schopstar}\ f))\frown(\mathit{schopstar}\ (g\frown(\mathit{schopstar}\ f)))) =$
$(\mathit{schopstar}\ f)\frown(\mathit{empty} \lor g\frown(\mathit{schopstar}\ (f \lor g)))$
by (*metis 1 2 SChopAssoc inteq-reflection*)
have *4*: $\vdash (\mathit{schopstar}\ f)\frown(\mathit{empty} \lor g\frown(\mathit{schopstar}\ (f \lor g))) =$
$((\mathit{schopstar}\ f)\frown\mathit{empty} \lor (\mathit{schopstar}\ f)\frown(g\frown(\mathit{schopstar}\ (f \lor g))))$
using *SChopOrEqv* **by** *blast*
have *5*: $\vdash (\mathit{schopstar}\ f)\frown\mathit{empty} = (\mathit{schopstar}\ f)$
by (*metis ChopEmpty Prop10 SChopstar-finite inteq-reflection schop-d-def*)
have *6*: $\vdash ((\mathit{schopstar}\ f)\frown\mathit{empty} \lor (\mathit{schopstar}\ f)\frown(g\frown(\mathit{schopstar}\ (f \lor g)))) =$
$((\mathit{schopstar}\ f) \lor (\mathit{schopstar}\ f)\frown(g\frown(\mathit{schopstar}\ (f \lor g))))$
using *5* **by** *auto*
show *?thesis*
by (*metis 1 2 3 4 6 int-eq*)
qed

lemma *SChopstarMore-or-unfold*:
$\vdash ((\mathit{schopstar}\ f) \lor (\mathit{schopstar}\ f)\frown((g \land \mathit{more})\frown(\mathit{schopstar}\ (f \lor g)))) = (\mathit{schopstar}\ (f \lor g))$

using *SChopstar-or-unfold*[*of LIFT f* $\wedge$ *more LIFT g* $\wedge$ *more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT* (*f* $\vee$ *g*)]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-troeger*:
$\vdash$ (*wpowerstar* (*f* $\vee$ *g*));*h* =
(*wpowerstar f*);(*g*;((*wpowerstar* (*f* $\vee$ *g*));*h*) $\vee$ *h*)
proof –
have *1*: $\vdash$ (*wpowerstar* (*f* $\vee$ *g*));*h* =
((*wpowerstar f*) $\vee$ (*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*))));*h*
using *WPowerstar-or-unfold*[*of f g*] **by** (*metis LeftChopEqvChop int-eq*)
have *2*: $\vdash$ ((*wpowerstar f*) $\vee$ (*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*))));*h* =
((*wpowerstar f*);*h* $\vee$ ((*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*))));*h*)
by (*simp add*: *OrChopEqv*)
have *3*: $\vdash$ ((*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*))));*h* =
(*wpowerstar f*);(*g*;((*wpowerstar* (*f* $\vee$ *g*));*h*))
by (*metis ChopAssoc inteq-reflection*)
have *3*: $\vdash$ ((*wpowerstar f*);*h* $\vee$ ((*wpowerstar f*);(*g*;(*wpowerstar* (*f* $\vee$ *g*))));*h*) =
(*wpowerstar f*);(*h* $\vee$ *g*;((*wpowerstar* (*f* $\vee$ *g*));*h*))
by (*metis 3 ChopOrEqv inteq-reflection*)
have *4*: $\vdash$ (*h* $\vee$ *g*;((*wpowerstar* (*f* $\vee$ *g*));*h*)) =
(*g*;((*wpowerstar* (*f* $\vee$ *g*));*h*) $\vee$ *h*)
by *fastforce*
show *?thesis*
by (*metis 1 2 3 4 int-eq*)
qed

lemma *SChopstar-troeger*:
$\vdash$ (*schopstar* (*f* $\vee$ *g*))$\frown$(*h*) =
(*schopstar f*)$\frown$(*g*$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$(*h*)) $\vee$ (*h*))
proof –
have *1*: $\vdash$ (*schopstar* (*f* $\vee$ *g*))$\frown$(*h*) =
((*schopstar f*) $\vee$ (*schopstar f*)$\frown$(*g*$\frown$(*schopstar* (*f* $\vee$ *g*))))$\frown$(*h*)
using *SChopstar-or-unfold*[*of f g*] **by** (*metis LeftSChopEqvSChop int-eq*)
have *2*: $\vdash$ ((*schopstar f*) $\vee$ (*schopstar f*)$\frown$(*g*$\frown$(*schopstar* (*f* $\vee$ *g*))))$\frown$(*h*) =
((*schopstar f*)$\frown$*h* $\vee$ ((*schopstar f*)$\frown$(*g*$\frown$(*schopstar* (*f* $\vee$ *g*))))$\frown$*h*)
by (*simp add*: *OrSChopEqv*)
have *3*: $\vdash$ ((*schopstar f*)$\frown$(*g*$\frown$(*schopstar* (*f* $\vee$ *g*))))$\frown$*h* =
(*schopstar f*)$\frown$(*g*$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$(*h*)))
by (*metis SChopAssoc inteq-reflection*)
have *3*: $\vdash$ ((*schopstar f*)$\frown$*h* $\vee$ ((*schopstar f*)$\frown$(*g*$\frown$(*schopstar* (*f* $\vee$ *g*))))$\frown$*h*) =
(*schopstar f*)$\frown$((*h*) $\vee$ *g*$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$(*h*)))
by (*metis 3 SChopOrEqv inteq-reflection*)
have *4*: $\vdash$ ((*h*) $\vee$ *g*$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$(*h*))) =
(*g*$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$(*h*)) $\vee$ (*h*))
by *fastforce*
show *?thesis*

by (*metis 1 2 3 4 int-eq*)
qed

lemma *SChopstarMore-troeger*:
$\vdash$ (*schopstar* (*f* $\vee$ *g*))$\frown$(*h*) =
(*schopstar f*)$\frown$((*g* $\wedge$ *more*)$\frown$((*schopstar* (*f* $\vee$ *g*))$\frown$(*h*)) $\vee$ (*h*))
using *SChopstar-troeger*[*of LIFT f* $\wedge$ *more LIFT g* $\wedge$ *more h*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT* (*f* $\vee$ *g*)]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-square*:
$\vdash$ (*wpowerstar* (*f*;*f*)) $\longrightarrow$ (*wpowerstar f*)
proof –
have *1*: $\vdash$ (*f*;*f*);(*wpowerstar f*) $\longrightarrow$ (*wpowerstar f*)
by (*simp add*: *Chop-WPowerstar-Closure WPowerstar-ext*)
show *?thesis*
by (*simp add*: *1 WPowerstar-induct-lvar-star*)
qed

lemma *SChopstar-square*:
$\vdash$ (*schopstar* (*f*$\frown$*f*)) $\longrightarrow$ (*schopstar f*)
proof –
have *1*: $\vdash$ (*f*$\frown$*f*)$\frown$(*schopstar f*) $\longrightarrow$ (*schopstar f*)
by (*metis Prop05 RightSChopImpSChop SChopAssoc SChopstar-1L SChopstar-unfoldl-eq inteq-reflection*)
show *?thesis*
by (*simp add*: *1 SChopstar-induct-lvar-star*)
qed

lemma *WPowerstar-meyer-1*:
$\vdash$ (*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*)) = (*wpowerstar f*)
proof –
have *1*: $\vdash$ *f*;((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*))) =
f;(*empty*;(*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*)))
by (*simp add*: *OrChopEqv RightChopEqvChop*)
have *2*: $\vdash$ (*empty*;(*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*))) =
((*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*)))
by (*meson EmptyOrChopEqv OrChopEqv Prop04*)
have *3*: $\vdash$ *f*;(*empty*;(*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*))) =
f;((*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*)))
using *2 RightChopEqvChop* **by** *blast*
have *4*: $\vdash$ *f*;((*wpowerstar* (*f* ; *f*)) $\vee$ *f*;(*wpowerstar* (*f* ; *f*))) $\longrightarrow$
f;(*wpowerstar* (*f* ; *f*)) $\vee$ (*wpowerstar* (*f* ; *f*))
by (*metis ChopAssoc ChopOrImp Prop05 Prop08 int-eq int-iffD2 wpowerstar-unfoldl-eq*)
have *41*: $\vdash$ *f*;((*empty* $\vee$ *f*);(*wpowerstar* (*f* ; *f*))) $\longrightarrow$ *f*;(*wpowerstar* (*f*;*f*) $\vee$ *f*;*wpowerstar* (*f*;*f*))
using *EmptyOrChopEqv*[*of f LIFT* (*wpowerstar* (*f* ; *f*))]
by (*metis 1 Prop11 inteq-reflection*)
have *42*: $\vdash$ *f*;(*wpowerstar* (*f*;*f*) $\vee$ *f*;*wpowerstar* (*f*;*f*)) = (*f*;*wpowerstar* (*f*;*f*) $\vee$ *f*;(*f*;*wpowerstar* (*f*;*f*)))
using *ChopOrEqv*[*of f LIFT wpowerstar* (*f*;*f*) *LIFT f*;*wpowerstar* (*f*;*f*)]

by *auto*
have *43*: $\vdash$ $(f;wpowerstar\ (f;f) \vee f;(f;wpowerstar\ (f;f))) =$
$(\ f;wpowerstar\ (f;f) \vee (f;f);wpowerstar\ (f;f))$
using *ChopAssoc*[*of f f LIFT wpowerstar* (*f*;*f*)] **by** *auto*
have *44*: $\vdash$ $(\ f;wpowerstar\ (f;f) \vee (f;f);wpowerstar\ (f;f)) \longrightarrow ((empty \vee f);(wpowerstar\ (f\ ;\ f)))$
using *wpowerstar-unfoldl-eq*[*of LIFT* (*f*;*f*)]
using *EmptyOrChopEqv* **by** *fastforce*
have *5*: $\vdash$ $f;((empty \vee f);(wpowerstar\ (f\ ;\ f))) \longrightarrow$
$((empty \vee f);(wpowerstar\ (f\ ;\ f)))$
by (*metis 1 3 42 43 44 int-eq*)
have *6*: $\vdash$ $empty \longrightarrow ((empty \vee f);(wpowerstar\ (f\ ;\ f)))$
using *EmptyOrChopEqv wpowerstar-unfoldl-eq* **by** *fastforce*
have *7*: $\vdash$ $empty \vee f;((empty \vee f);(wpowerstar\ (f\ ;\ f))) \longrightarrow$
$((empty \vee f);(wpowerstar\ (f\ ;\ f)))$
using *5 6 Prop02* **by** *blast*
have *8*: $\vdash$ $(wpowerstar\ f) \longrightarrow ((empty \vee f);(wpowerstar\ (f\ ;\ f)))$
using *7 WPowerstar-induct-lvar-empty* **by** *blast*
have *9*: $\vdash$ $((empty \vee f);(wpowerstar\ (f\ ;\ f))) \longrightarrow (empty \vee f)\ ;(wpowerstar\ f)$
using *WPowerstar-square*[*of f*]
using *RightChopImpChop* **by** *blast*
have *10*: $\vdash$ $(empty \vee f)\ ;(wpowerstar\ f) \longrightarrow (wpowerstar\ f)$
by (*meson Prop02 WPowerstarInductR WPowerstar-1R WPowerstar-ext WPowerstar-imp-empty*)
show *?thesis*
by (*meson 10 8 9 int-iffI lift-imp-trans*)
qed

lemma *SChopstar-meyer-1*:
$\vdash$ $(empty \vee f)\frown(schopstar\ (f \frown f)) = (schopstar\ f)$
proof –
have *1*: $\vdash$ $f\frown((empty \vee f)\frown(schopstar\ (f \frown f))) =$
$f\frown(empty\frown(schopstar\ (f \frown f)) \vee f\frown(schopstar\ (f \frown f)))$
by (*simp add*: *OrSChopEqv RightSChopEqvSChop*)
have *2*: $\vdash$ $(empty\frown(schopstar\ (f \frown f)) \vee f\frown(schopstar\ (f \frown f))) =$
$((schopstar\ (f \frown f)) \vee f\frown(schopstar\ (f \frown f)))$
by (*meson EmptyOrSChopEqv OrSChopEqv Prop04*)
have *3*: $\vdash$ $f\frown(empty\frown(schopstar\ (f \frown f)) \vee f\frown(schopstar\ (f \frown f))) =$
$f\frown((schopstar\ (f \frown f)) \vee f\frown(schopstar\ (f \frown f)))$
using *2 RightSChopEqvSChop* **by** *blast*
have *4*: $\vdash$ $f\frown((schopstar\ (f \frown f)) \vee f\frown(schopstar\ (f \frown f))) \longrightarrow$
$(schopstar\ (f \frown f)) \vee f\frown(schopstar\ (f \frown f))$
using *SChopAssoc SChopOrEqv SChopstar-unfoldl-eq* **by** *fastforce*
have *41*: $\vdash$ $((schopstar\ (f \frown f)) \vee f\frown(schopstar\ (f \frown f))) =$
$((empty \vee f)\frown(schopstar\ (f \frown f)))$
by (*metis 2 OrSChopEqv inteq-reflection*)
have *5*: $\vdash$ $f\frown((empty \vee f)\frown(schopstar\ (f \frown f))) \longrightarrow$
$((empty \vee f)\frown(schopstar\ (f \frown f)))$
by (*metis 4 41 int-eq*)
have *6*: $\vdash$ $empty \longrightarrow ((empty \vee f)\frown(schopstar\ (f \frown f)))$
using *EmptyOrSChopEqv SChopstar-imp-empty* **by** *fastforce*
have *7*: $\vdash$ $empty \vee f\frown((empty \vee f)\frown(schopstar\ (f \frown f))) \longrightarrow$

$((empty \lor f)\frown(schopstar\ (f \frown f)))$
using *5 6 Prop02* **by** *blast*
have *8*: $\vdash (schopstar\ f) \longrightarrow ((empty \lor f)\frown(schopstar\ (f \frown f)))$
using *7 SChopstar-induct-lvar-empty* **by** *blast*
have *9*: $\vdash ((empty \lor f)\frown(schopstar\ (f \frown f))) \longrightarrow (empty \lor f) \frown(schopstar\ f)$
using *SChopstar-square[of f]*
using *RightSChopImpSChop* **by** *blast*
have *10*: $\vdash (empty \lor f) \frown(schopstar\ f) \longrightarrow (schopstar\ f)$
by (*metis SChopstar-1L SChopstar-star2 inteq-reflection*)
show *?thesis*
by (*meson 10 8 9 int-iffI lift-imp-trans*)
qed

lemma *SChopstarMore-meyer-1*:
$\vdash (empty \lor (f \land more))\frown(schopstar\ (f \frown f)) = (schopstar\ f)$
proof –
have *1*: $\vdash (f \land more)\frown((empty \lor (f \land more))\frown(schopstar\ (f \frown f))) =$
$(f \land more)\frown(empty\frown(schopstar\ (f \frown f)) \lor (f \land more)\frown(schopstar\ (f \frown f)))$
by (*simp add*: *OrSChopEqv RightSChopEqvSChop*)
have *2*: $\vdash (empty\frown(schopstar\ (f \frown f)) \lor (f \land more)\frown(schopstar\ (f \frown f))) =$
$((schopstar\ (f \frown f)) \lor (f \land more)\frown(schopstar\ (f \frown f)))$
by (*meson EmptyOrSChopEqv OrSChopEqv Prop04*)
have *3*: $\vdash (f \land more)\frown(empty\frown(schopstar\ (f \frown f)) \lor (f \land more)\frown(schopstar\ (f \frown f))) =$
$(f \land more)\frown((schopstar\ (f \frown f)) \lor (f \land more)\frown(schopstar\ (f \frown f)))$
using *2 RightSChopEqvSChop* **by** *blast*
have *30*: $\vdash ((f \land more)\frown(f \land more)) \longrightarrow f \frown f$
by (*metis Prop11 Prop12 SChopImpSChop lift-and-com*)
have *31*: $\vdash ((f \land more)\frown(f \land more)) \frown (schopstar\ (f \frown f)) \longrightarrow (schopstar\ (f \frown f))$
by (*metis 30 AndSChopA Prop05 Prop10 SChopstar-unfoldl-eq int-eq lift-and-com*)
have *4*: $\vdash (f \land more)\frown((schopstar\ (f \frown f)) \lor (f \land more)\frown(schopstar\ (f \frown f))) \longrightarrow$
$(f \land more)\frown(schopstar\ (f \frown f)) \lor (schopstar\ (f \frown f))$
using *31 SChopAssoc SChopOrImp* **by** *fastforce*
have *5*: $\vdash (f \land more)\frown((empty \lor (f \land more))\frown(schopstar\ (f \frown f))) \longrightarrow$
$((empty \lor (f \land more))\frown(schopstar\ (f \frown f)))$
by (*metis* (*no-types*, *opaque-lifting*) *4 EmptyOrSChopEqv Prop11 int-simps*(*33*)
inteq-reflection lift-and-com lift-imp-trans)
have *6*: $\vdash empty \longrightarrow ((empty \lor (f \land more))\frown(schopstar\ (f \frown f)))$
using *EmptyOrSChopEqv SChopstar-unfoldl-eq* **by** *fastforce*
have *7*: $\vdash empty \lor (f \land more)\frown((empty \lor (f \land more))\frown(schopstar\ (f \frown f))) \longrightarrow$
$((empty \lor (f \land more))\frown(schopstar\ (f \frown f)))$
using *5 6 Prop02* **by** *blast*
have *8*: $\vdash (schopstar\ f) \longrightarrow ((empty \lor (f \land more))\frown(schopstar\ (f \frown f)))$
using *7 SChopstarMore-induct-lvar-empty* **by** *blast*
have *9*: $\vdash ((empty \lor (f \land more))\frown(schopstar\ (f \frown f))) \longrightarrow (empty \lor (f \land more)) \frown(schopstar\ f)$
using *SChopstar-square[of f]*
using *RightSChopImpSChop* **by** *blast*
have *10*: $\vdash (empty \lor (f \land more)) \frown(schopstar\ f) \longrightarrow (schopstar\ f)$
by (*metis SChopstar-1L SChopstar-and-more SChopstar-star2 inteq-reflection*)
show *?thesis*
by (*meson 10 8 9 int-iffI lift-imp-trans*)

qed

lemma *WPowerstar-tc*:
assumes $\vdash f;f \longrightarrow f$
shows $\vdash (wpowerstar\ f);f = f$
proof –
 have *1*: $\vdash f \vee f;\ f \longrightarrow f$
   using *assms* **by** *fastforce*
 have *2*: $\vdash (wpowerstar\ f);f \longrightarrow f$
  by (*simp add*: *WPowerstar-inductL-var-equiv assms*)
 have *3*: $\vdash f \longrightarrow (wpowerstar\ f);f$
   by (*metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection*)
 show *?thesis*
 by (*simp add*: *2 3 Prop11*)
qed

lemma *SChopstar-tc*:
assumes $\vdash f\frown f \longrightarrow f$
shows $\vdash (schopstar\ f)\frown(f \wedge finite) = (f \wedge finite)$
proof –
 have *1*: $\vdash f \vee f\frown\ f \longrightarrow f$
   using *assms* **by** *fastforce*
 have *2*: $\vdash (schopstar\ f)\frown(f \wedge finite) \longrightarrow f \wedge finite$
  by (*metis Prop12 SChopAndA SChopstar-1R SChopstar-finite SChopstar-induct-lvar assms lift-imp-trans*)
 have *3*: $\vdash f \wedge finite\longrightarrow (schopstar\ f)\frown(f \wedge finite)$
  by (*metis EmptySChop LeftSChopImpSChop SChopstar-imp-empty inteq-reflection*)
 show *?thesis*
 by (*simp add*: *2 3 Prop11*)
qed

lemma *WPowerstar-tc-eq*:
assumes $\vdash f;f = f$
shows $\vdash (wpowerstar\ f);f = f$
using *assms*
by (*simp add*: *WPowerstar-induct-lvar-eq2*)

lemma *SChopstar-tc-eq*:
assumes $\vdash f\frown f = f$
shows $\vdash (schopstar\ f)\frown(f \wedge finite) = (f \wedge finite)$
using *assms*
by (*simp add*: *SChopstar-tc int-iffD1*)

lemma *WPowerstar-boffa-var*:
 assumes $\vdash f;f \longrightarrow f$
 shows $\vdash (wpowerstar\ f) = (empty \vee f)$
proof –
 have *1*: $\vdash (wpowerstar\ f) = (empty \vee (wpowerstar\ f);f)$
    by (*metis WPowerstar-slide-var WPowerstarEqv int-eq*)
 show *?thesis*
 using *1 Prop06 WPowerstar-tc assms* **by** *blast*

qed

lemma *SChopstar-boffa-var*:
assumes $\vdash f \frown f \longrightarrow f$
shows $\vdash (schopstar\ f) = (empty \lor (f \land finite))$
proof –
have *1*: $\vdash (schopstar\ f) = (empty \lor (schopstar\ f) \frown (f \land finite))$
by (*meson SChopstar-unfoldR SChopstar-unfoldr-eq int-iffD2 int-iffI*)
show *?thesis*
using *1 Prop06 SChopstar-tc assms* **by** *blast*
qed

lemma *WPowerstar-boffa*:
assumes $\vdash f;f = f$
shows $\vdash (wpowerstar\ f) = (empty \lor f)$
using *assms*
by (*simp add*: *WPowerstar-boffa-var int-iffD1*)

lemma *SChopstar-boffa*:
assumes $\vdash f \frown f = f$
shows $\vdash (schopstar\ f) = (empty \lor (f \land finite))$
using *assms*
by (*simp add*: *SChopstar-boffa-var int-iffD1*)

lemma *WPowerstar-sim2*:
assumes $\vdash h\ ;\ f \longrightarrow g\ ;\ h$
shows $\vdash h\ ;\ (wpowerstar\ f) \longrightarrow (wpowerstar\ g)\ ;\ h$
proof –
have *1*: $\vdash (wpowerstar\ g)\ ;\ (h\ ;\ f) \longrightarrow (wpowerstar\ g)\ ;(\ g\ ;\ h\ )$
by (*simp add*: *RightChopImpChop assms*)
have *2*: $\vdash (wpowerstar\ g)\ ;(\ g\ ;\ h\ ) \longrightarrow (wpowerstar\ g)\ ;\ h$
by (*metis ChopAssoc LeftChopImpChop WPowerstar-1L WPowerstar-slide-var inteq-reflection*)
have *3*: $\vdash (wpowerstar\ g)\ ;\ (h\ ;\ f) \longrightarrow (wpowerstar\ g)\ ;\ h$
using *1 2 lift-imp-trans* **by** *blast*
have *4*: $\vdash h \longrightarrow (wpowerstar\ g)\ ;\ h$
by (*metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection*)
have *5*: $\vdash h \lor (wpowerstar\ g)\ ;\ (h\ ;\ f) \longrightarrow (wpowerstar\ g)\ ;\ h$
using *3 4 Prop02* **by** *blast*
show *?thesis*
by (*metis 5 ChopAssoc WPowerstarInductR inteq-reflection*)
qed

lemma *SChopstarInductR*:
assumes $\vdash g \lor h \frown f \longrightarrow h$
shows $\vdash g \frown schopstar\ f \longrightarrow h$
proof –
have *1*: $\vdash\ \ g \land finite\ \ \longrightarrow h$
using *assms* **by** *fastforce*
have *2*: $\vdash h \frown f \longrightarrow h$

using *assms* **by** *fastforce*
have *3*: ⊢ $g \wedge$ *finite* $\vee$ $h \frown f \longrightarrow h$
using *1 2* **by** *fastforce*
have *4*: ⊢ $(g \wedge$ *finite*$)$;*fpowerstar f* $\longrightarrow h$
by (*metis 1 2 ChopSChopdef FPowerstarInductR OrFiniteInf Prop02 Prop03 inteq-reflection*)
show *?thesis*
using *FPowerstar-more-absorb*[*of f*] *4*
by (*metis int-eq schop-d-def schopstar-d-def*)
qed

lemma *SChopstar-sim2*:
assumes ⊢ $h \frown f \longrightarrow g \frown h$
shows ⊢ $h \frown$ (*schopstar f*) $\longrightarrow$ (*schopstar g*) $\frown h$
proof −
have *1*: ⊢ (*schopstar g*) $\frown$ $(h \frown (f)) \longrightarrow$ (*schopstar g*) $\frown(g \frown h)$
by (*simp add*: *RightSChopImpSChop assms*)
have *2*: ⊢ (*schopstar g*) $\frown(g \frown h) \longrightarrow$ (*schopstar g*) $\frown h$
by (*metis* (*no-types*, *lifting*) *ChopAssoc LeftChopImpChop Prop10 SChopstar-1L SChopstar-finite SChopstar-WPowerstar WPowerstar-slide-var inteq-reflection schop-d-def*)
have *3*: ⊢ (*schopstar g*) $\frown$ $(h \frown (f)) \longrightarrow$ (*schopstar g*) $\frown h$
using *1 2 lift-imp-trans* **by** *blast*
have *4*: ⊢ $h \longrightarrow$ (*schopstar g*) $\frown h$
by (*meson EmptySChop LeftSChopImpSChop Prop11 SChopstar-imp-empty lift-imp-trans*)
have *5*: ⊢ $h \vee$ (*schopstar g*) $\frown$ $(h \frown (f)) \longrightarrow$ (*schopstar g*) $\frown h$
using *3 4 Prop02* **by** *blast*
show *?thesis*
by (*metis 5 SChopAssoc SChopstarInductR inteq-reflection*)
qed

lemma *SChopImpFinite*:
⊢ $f \longrightarrow$ *finite* $\Longrightarrow$ ⊢ $g \frown f \longrightarrow$ *finite*
by (*metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop10 SChopSFinExportA SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection*)

lemma *SChopstar-sim2-finite*:
assumes ⊢ $h \frown f \longrightarrow g \frown h$
shows ⊢ $h \frown$ (*schopstar f*) $\longrightarrow$ (*schopstar g*) $\frown$ $(h \wedge$ *finite*$)$
using *assms SChopstar-sim2*
by (*metis FiniteImportSChopRight Prop12 SChopImpFinite SChopstar-finite inteq-reflection*)

lemma *WPowerstar-inductr-var*:
assumes ⊢ g ; $f \longrightarrow g$
shows ⊢ g ; (*wpowerstar f*) $\longrightarrow g$
using *assms*
by (*simp add*: *WPowerstarInductR*)

lemma *SChopstar-inductr-var*:
assumes ⊢ $g \frown f \longrightarrow g$
shows ⊢ $g \frown$ (*schopstar f*) $\longrightarrow g$
using *assms*

by (*simp add*: *SChopstarInductR*)

lemma *SChopstarMore-inductr-var*:
assumes $\vdash g \frown (f \wedge more) \longrightarrow g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*
by (*metis SChopstar-and-more SChopstar-inductr-var inteq-reflection*)

lemma *SChopstar-finite-absorb*:
$\vdash schopstar\ (f \wedge finite) = schopstar\ f$
by (*metis Prop10 SChopstar-ext SChopstar-finite SChopstar-WPowerstar int-eq lift-imp-trans*)

lemma *SChopstar-inductr-finite-var*:
assumes $\vdash g \frown (f \wedge finite) \longrightarrow g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*
by (*metis SChopstar-finite-absorb SChopstar-inductr-var inteq-reflection*)

lemma *SChopstarMore-inductr-finite-var*:
assumes $\vdash g \frown ((f \wedge more) \wedge finite) \longrightarrow g$
shows $\vdash g \frown (schopstar\ f) \longrightarrow g$
using *assms*
by (*metis SChopstar-and-more SChopstar-inductr-finite-var inteq-reflection*)

lemma *WPowerstar-inductr-var-equiv*:
$(\vdash g\ ;\ f \longrightarrow g\ ) = (\vdash g\ ;\ (wpowerstar\ f) \longrightarrow g)$
proof –
have *1*: $(\vdash g\ ;\ f \longrightarrow g\ ) \Longrightarrow (\vdash g\ ;\ (wpowerstar\ f) \longrightarrow g)$
by (*simp add*: *WPowerstar-inductr-var*)
have *2*: $(\vdash g\ ;\ (wpowerstar\ f) \longrightarrow g) \Longrightarrow (\vdash g\ ;\ f \longrightarrow g\ )$
by (*metis ChopAndB Prop10 WPowerstar-ext int-eq lift-imp-trans*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *SChopstar-inductr-var-equiv*:
$(\vdash g \frown (f \wedge finite) \longrightarrow g\ ) = (\vdash g \frown (schopstar\ f) \longrightarrow g)$
proof –
have *1*: $(\vdash g \frown (f \wedge finite) \longrightarrow g\ ) \Longrightarrow (\vdash g \frown (schopstar\ f) \longrightarrow g)$
by (*simp add*: *SChopstar-inductr-finite-var*)
have *2*: $(\vdash g \frown (schopstar\ f) \longrightarrow g) \Longrightarrow (\vdash g \frown (f \wedge finite) \longrightarrow g\ )$
by (*metis Prop10 Prop12 SChopAndB SChopstar-ext inteq-reflection*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *SChopstarMore-inductr-var-equiv*:
$(\vdash g \frown ((f \wedge more) \wedge finite) \longrightarrow g\ ) = (\vdash g \frown (schopstar\ f) \longrightarrow g)$
using *SChopstar-inductr-var-equiv*[*of g LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-sim3*:
assumes $\vdash h\ ;\ f = g\ ;\ h$
shows $\vdash h\ ;\ (wpowerstar\ f) = (wpowerstar\ g)\ ;h$
using *assms*
by (*simp add*: *Prop11 WPowerstar-sim1 WPowerstar-sim2*)

lemma *SChopstar-sim3*:
assumes $\vdash h \frown f = g \frown h$
shows $\vdash h \frown (schopstar\ f) = (schopstar\ g) \frown (h \wedge finite)$
using *SChopstar-sim1 SChopstar-sim2-finite*
by (*metis Prop11 assms*)

lemma *SChopstarMore-sim3*:
assumes $\vdash h \frown (f \wedge more) = (g \wedge more) \frown h$
shows $\vdash h \frown (schopstar\ f) = (schopstar\ g) \frown (h \wedge finite)$
by (*metis SChopstar-and-more SChopstar-sim3 assms inteq-reflection*)

lemma *WPowerstar-sim4*:
assumes $\vdash f\ ;\ g \longrightarrow g\ ;\ f$
shows $\vdash (wpowerstar\ f);(wpowerstar\ g) \longrightarrow (wpowerstar\ g)\ ;\ (wpowerstar\ f)$
using *assms*
by (*simp add*: *WPowerstar-sim1 WPowerstar-sim2*)

lemma *SChopstar-sim4*:
assumes $\vdash f \frown g \longrightarrow g \frown f$
shows $\vdash (schopstar\ f)\frown(schopstar\ g) \longrightarrow (schopstar\ g) \frown (schopstar\ f)$
using *assms*
by (*metis Prop10 SChopstar-finite SChopstar-sim1 SChopstar-sim2 inteq-reflection*)

lemma *WPowerstar-inductr-eq*:
assumes $\vdash (h \vee g\ ;\ f) = g$
shows $\vdash h;(wpowerstar\ f) \longrightarrow g$
using *assms*
using *WPowerstarInductR int-iffD1* **by** *blast*

lemma *SChopstar-inductr-eq*:
assumes $\vdash (h \vee g \frown f) = g$
shows $\vdash h\frown(schopstar\ f) \longrightarrow g$
using *assms* **using** *SChopstarInductR int-iffD1* **by** *blast*

lemma *SChopstarMore-inductr-eq*:
assumes $\vdash (h \vee g \frown (f \wedge more)) = g$
shows $\vdash h\frown(schopstar\ f) \longrightarrow g$
using *assms*
by (*metis SChopstar-and-more SChopstar-inductr-eq inteq-reflection*)

lemma *WPowerstar-inductr-var-eq*:

assumes $\vdash$ (g ; f) = g
shows $\vdash$ g;(*wpowerstar* f) $\longrightarrow$ g
using *assms*
using *WPowerstar-inductr-var int-iffD1* **by** *blast*

lemma *SChopstar-inductr-var-eq*:
assumes $\vdash$ ($g \frown f$) = g
shows $\vdash$ $g\frown$(*schopstar* f) $\longrightarrow$ g
using *assms*
using *SChopstar-inductr-var int-iffD1* **by** *blast*

lemma *SChopstarMore-inductr-var-eq*:
assumes $\vdash$ ($g \frown$ (f $\wedge$ *more*)) = g
shows $\vdash$ $g\frown$(*schopstar* f) $\longrightarrow$ g
using *assms*
by (*simp add*: *SChopstarMore-inductr-var int-iffD1*)

lemma *WPowerstar-inductr-var-eq2*:
assumes $\vdash$ (g ; f) = g
shows $\vdash$ g;(*wpowerstar* f) = g
using *assms*
by (*metis Prop11 RightChopImpChop WPowerstar-ext WPowerstar-inductr-var-eq inteq-reflection*)

lemma *SChopstar-inductr-var-eq2*:
assumes $\vdash$ ($g \frown$ (f $\wedge$ *finite*)) = g
shows $\vdash$ $g\frown$(*schopstar* f) = g
using *assms*
by (*metis Prop11 RightSChopImpSChop SChopstar-ext SChopstar-inductr-finite-var inteq-reflection*)

lemma *SChopstarMore-inductr-var-eq2*:
assumes $\vdash$ ($g \frown$ ((f $\wedge$ *more*) $\wedge$ *finite*)) = g
shows $\vdash$ $g\frown$(*schopstar* f) = g
using *assms*
SChopstar-inductr-var-eq2[*of g LIFT f* $\wedge$ *more*]
SChopstar-and-more[*of f*]
by (*metis inteq-reflection*)

lemma *WPowerstar-bubble-sort*:
assumes $\vdash$ g ; f $\longrightarrow$ f ; g
shows $\vdash$ (*wpowerstar* (f $\vee$ g)) = (*wpowerstar* f) ; (*wpowerstar* g)
using *assms*
using *WPowerstar-church-rosser WPowerstar-sim4* **by** *blast*

lemma *SChopstar-bubble-sort*:
assumes $\vdash$ $g \frown f \longrightarrow f \frown g$
shows $\vdash$ (*schopstar* (f $\vee$ g)) = (*schopstar* f) $\frown$ (*schopstar* g)
using *assms* **by** (*simp add*: *SChopstar-church-rosser SChopstar-sim4*)

lemma *WPowerstar-indepence1*:

assumes $\vdash f \; ; \; g = \#False$
shows $\vdash (wpowerstar\ f);g = g$
using *assms*
by (*metis EmptyOrChopEqv WPowerstar-induct-lvar-eq2 WPowerstar-star2 assms int-eq int-simps*(*25*))

lemma *SChopRightFalse*:
$\vdash f \frown \#False = \#False$
by (*simp add*: *intI itl-defs*)

lemma *SChopstar-indepence1*:
assumes $\vdash f \frown g = \#False$
shows $\vdash (schopstar\ f) \frown g = g$
proof −
have *1*: $\vdash (schopstar\ f) \frown g = (g \lor (schopstar\ f) \frown (f \frown g))$
by (*metis* (*no-types*, *lifting*) *ChopAssoc EmptyOrSChopEqv Prop10 SChopstar-1L SChopstar-finite SChopstar-unfoldr-eq SChopstar-WPowerstar WPowerstar-slide-var inteq-reflection lift-imp-trans schop-d-def*)
have *2*: $\vdash (schopstar\ f) \frown (f \frown g) = (schopstar\ f) \frown (\#False)$
by (*simp add*: *RightSChopEqvSChop assms*)
have *3*: $\vdash (schopstar\ f) \frown (\#False) = \#False$
by (*simp add*: *SChopRightFalse*)
show *?thesis*
using *1 2 3* **by** *fastforce*
qed

lemma *WPowerstar-indepence2*:
assumes $\vdash f \; ; \; g = \#False$
shows $\vdash f \; ; \; (wpowerstar\ g) = f$
using *assms WPowerstar-inductr-var-eq2*[*of f g*] *WPowerstar-star2*[*of g*]
by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty WPowerstar-inductr-var-equiv int-eq int-iffI int-simps*(*11*))

lemma *SChopstar-indepence2*:
assumes $\vdash f \frown g = \#False$
shows $\vdash f \frown (schopstar\ g) = (f \land finite)$
proof −
have *1*: $\vdash f \frown (schopstar\ g) = ((f \land finite) \lor (f \frown g) \frown (schopstar\ g))$
by (*metis ChopEmpty FPowerstarEqv FPowerstar-more-absorb SChopAssoc SChopOrEqv int-eq schop-d-def schopstar-d-def*)
have *2*: $\vdash (f \frown g) \frown (schopstar\ g) = \#False$
by (*metis AndInfChopEqvAndInf assms int-eq int-simps*(*19*) *schop-d-def*)
show *?thesis*
by (*metis 1 2 int-simps*(*25*) *inteq-reflection*)
qed

lemma *WPowerstar-lazy-comm*:
$(\vdash g;f \longrightarrow f;(wpowerstar\ (f \lor g)) \lor g) =$

($\vdash$ *g* ;(*wpowerstar f*) $\longrightarrow$ *f* ; (*wpowerstar* (*f* $\vee$ *g*)) $\vee$ *g*)
proof
 assume *1*: $\vdash$ *g*;*f* $\longrightarrow$ *f*;(*wpowerstar* (*f* $\vee$ *g*)) $\vee$ *g*
 show $\vdash$ *g* ;(*wpowerstar f*) $\longrightarrow$ *f* ; (*wpowerstar* (*f* $\vee$ *g*)) $\vee$ *g*
  proof −
   have *2*: $\vdash$ (*f*;(*wpowerstar* (*f* $\vee$ *g*)) $\vee$ *g*);*f* =
        ((*f*;(*wpowerstar* (*f* $\vee$ *g*)));*f* $\vee$ *g*;*f*)
    by (*simp add*: *OrChopEqv*)
   have *3*: $\vdash$ ((*f*;(*wpowerstar* (*f* $\vee$ *g*)));*f* $\vee$ *g*;*f*) $\longrightarrow$
        ((*f*;(*wpowerstar* (*f* $\vee$ *g*)));*f* $\vee$ (*f*;(*wpowerstar* (*f* $\vee$ *g*))) $\vee$ *g*)
      using *1* **by** *fastforce*
   have *4*: $\vdash$ (*f*;(*wpowerstar* (*f* $\vee$ *g*)));*f* $\longrightarrow$ (*f*;(*wpowerstar* (*f* $\vee$ *g*)))
    by (*metis ChopAssoc RightChopImpChop WPowerstar-subdist-var-1 WPowerstar-trans-eq inteq-reflection*)
   have *5*: $\vdash$ ((*f*;(*wpowerstar* (*f* $\vee$ *g*)));*f* $\vee$ (*f*;(*wpowerstar* (*f* $\vee$ *g*))) $\vee$ *g*) $\longrightarrow$
        ((*f*;(*wpowerstar* (*f* $\vee$ *g*))) $\vee$ *g*)
    using *4* **by** *fastforce*
   have *6*: $\vdash$ *g* $\vee$ ((*f*;(*wpowerstar* (*f* $\vee$ *g*))) $\vee$ *g*) $\longrightarrow$ ((*f*;(*wpowerstar* (*f* $\vee$ *g*))) $\vee$ *g*)
      by *fastforce*
   show *?thesis* **using** *WPowerstarInductR*[*of g LIFT* (*f*;*wpowerstar* (*f* $\vee$ *g*) $\vee$ *g*) *f*]
    using *2 3 5* **by** *fastforce*
 qed
next
 assume *7*: $\vdash$ *g* ;(*wpowerstar f*) $\longrightarrow$ *f* ; (*wpowerstar* (*f* $\vee$ *g*)) $\vee$ *g*
 show $\vdash$ *g*;*f* $\longrightarrow$ *f*;(*wpowerstar* (*f* $\vee$ *g*)) $\vee$ *g*
 proof −
 have *80*: $\vdash$ *f*;(*wpowerstar f*) = (*f* $\vee$ *f*;(*wpowerstar f*))
   by (*meson ChopEmpty Prop02 Prop05 Prop11 RightChopImpChop WPowerstar-imp-empty lift-imp-trans*)

  have *8*: $\vdash$ (*wpowerstar f*) = (*empty* $\vee$ *f* $\vee$ *f*;(*wpowerstar f*))
    by (*meson 80 Prop06 WPowerstarEqv*)
  have *9*: $\vdash$ *g* ;(*wpowerstar f*) = (*g*;*empty* $\vee$ *g*;*f* $\vee$ *g*;(*f*;(*wpowerstar f*)))
    by (*metis 8 ChopOrEqv int-eq*)
  show *?thesis*
   using *7 9* **by** *fastforce*
 qed
qed

lemma *SChopstar-lazy-comm*:
($\vdash$ *g*⌢(*f* $\wedge$ *finite*) $\longrightarrow$ *f*⌢(*schopstar* (*f* $\vee$ *g*)) $\vee$ *g*) =
 ($\vdash$ *g* ⌢(*schopstar f*) $\longrightarrow$ *f* ⌢ (*schopstar* (*f* $\vee$ *g*)) $\vee$ *g*)
proof
 assume *1*: $\vdash$ *g*⌢(*f* $\wedge$ *finite*) $\longrightarrow$ *f*⌢(*schopstar* (*f* $\vee$ *g*)) $\vee$ *g*
 show $\vdash$ *g* ⌢(*schopstar f*) $\longrightarrow$ *f* ⌢ (*schopstar* (*f* $\vee$ *g*)) $\vee$ *g*
  proof −
   have *2*: $\vdash$ (*f*⌢(*schopstar* (*f* $\vee$ *g*)) $\vee$ *g*)⌢(*f* $\wedge$ *finite*) =
        ((*f*⌢(*schopstar* (*f* $\vee$ *g*)))⌢(*f* $\wedge$ *finite*) $\vee$ *g*⌢(*f* $\wedge$ *finite*))
    by (*simp add*: *OrSChopEqv*)
   have *3*: $\vdash$ ((*f*⌢(*schopstar* (*f* $\vee$ *g*)))⌢(*f* $\wedge$ *finite*) $\vee$ *g*⌢(*f*$\wedge$ *finite*)) $\longrightarrow$
        ((*f*⌢(*schopstar* (*f* $\vee$ *g*)))⌢(*f* $\wedge$ *finite*) $\vee$ (*f*⌢(*schopstar* (*f* $\vee$ *g*))) $\vee$ *g*)
      using *1*

using *SChopAndA* **by** *fastforce*
have *4*: ⊢ $(f \frown (schopstar\ (f \vee g))) \frown (f \wedge finite) \longrightarrow (f \frown (schopstar\ (f \vee g)))$
using *SChopstar-subdist-var-1*
by (*metis Prop11 RightSChopImpSChop SChopAssoc SChop-SChopstar-Closure SChopstardef lift-imp-trans*)
have *5*: ⊢ $((f \frown (schopstar\ (f \vee g))) \frown (f \wedge finite) \ \vee (f \frown (schopstar\ (f \vee g))) \vee g) \longrightarrow$
$((f \frown (schopstar\ (f \vee g))) \vee g)$
using *4* **by** *fastforce*
have *6*: ⊢ $g \vee ((f \frown (schopstar\ (f \vee g))) \vee g) \longrightarrow ((f \frown (schopstar\ (f \vee g))) \vee g)$
by *fastforce*
show *?thesis*
by (*metis* (*mono-tags*, *lifting*) *3 5 OrSChopEqv Prop03 SChopstar-inductr-var-equiv int-eq lift-imp-trans*)

qed
next
assume *7*: ⊢ $g \frown (schopstar\ f) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g$
show ⊢ $g \frown (f \wedge finite) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g$
proof −
have *8*: ⊢ $(schopstar\ f) = (empty \vee (f \wedge finite) \vee f \frown (schopstar\ f))$
by (*meson AndSChopA Prop02 Prop05 Prop08 SChopstarMore-unfoldl-eq SChopstar-1L SChopstar-ext SChopstar-imp-empty int-iffD2 int-iffI*)
have *9*: ⊢ $g \frown (schopstar\ f) = (g \frown empty \vee g \frown (f \wedge finite) \vee g \frown (f \frown (schopstar\ f)))$
by (*metis 8 SChopOrEqv inteq-reflection*)
show *?thesis*
using *7 9* **by** *fastforce*
qed
qed

8.6 WPowerstar

lemma *WPowerstar-and-inf*:
⊢ $(wpowerstar\ (f \wedge inf)) = (empty \vee (f \wedge inf))$
by (*simp add*: *AndInfChopEqvAndInf WPowerstar-boffa*)

lemma *WPowerstar-chop-and-finite-inf*:
⊢ $(wpowerstar\ f) = (wpowerstar\ (f \wedge finite));(wpowerstar\ (f \wedge inf))$
by (*metis AndInfChopEqvAndInf OrFiniteInf WPowerstar-denest-var inteq-reflection*)

lemma *WPowerstar-empty*:
⊢ $(wpowerstar\ empty) = empty$
by (*simp add*: *WPowerstar-subid*)

lemma *WPowerstar-more*:
⊢ $(wpowerstar\ more)$
by (*metis ChopImpDi DiMoreEqvMore TrueW WPowerstar-boffa-var empty-d-def int-simps*(*29*) *inteq-reflection*)

lemma *WPowerstar-false*:
⊢ $(wpowerstar\ \#False) = empty$
by (*simp add*: *WPowerstar-subid*)

lemma *WPowerstar-true*:
$\vdash$ (*wpowerstar #True*)
by (*metis MP TrueW WPowerstar-ext*)

lemma *WPowerstar-inf*:
$\vdash$ (*wpowerstar inf*) = (*empty* $\vee$ *inf*)
by (*simp add*: *InfChopInfEqvInf WPowerstar-boffa*)

lemma *WPowerstar-finite*:
$\vdash$ (*wpowerstar finite*) = *finite*
by (*meson EmptyImpFinite FiniteChopFiniteEqvFinite Prop02 Prop11 WPowerstar-ext WPowerstar-induct-lvar-emp*

lemma *WPowerstar-imp-finite*:
assumes $\vdash$ *f* $\longrightarrow$ *finite*
shows $\vdash$ *wpowerstar f* $\longrightarrow$ *finite*
using *assms*
by (*metis WPowerstar-finite WPowerstar-iso inteq-reflection*)

lemma *WPowerstar-and-empty*:
$\vdash$ (*wpowerstar* (*f* $\wedge$ *empty*)) = *empty*
by (*metis AndEmptyChopAndEmptyEqvAndEmpty Prop11 Prop12 WPowerstar-subid inteq-reflection*)

lemma *WPowerstarIntro*:
assumes $\vdash$ *f* $\longrightarrow$ (*g* $\wedge$ *more*); *f*
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *wpowerstar g*
proof –
have *14*: $\vdash$ *wpowerstar g* = (*empty* $\vee$ (*g* $\wedge$ *more*);(*wpowerstar g*))
using *WPowerstar-more-absorb*[*of g*] *WPowerstarEqv*[*of LIFT* (*g* $\wedge$ *more*)]
by (*metis int-eq*)
have *15*: $\vdash$ ($\neg$(*wpowerstar g*)) = (*more* $\wedge$ $\neg$((*g* $\wedge$ *more*);*wpowerstar g*))
using *14* **unfolding** *empty-d-def* **by** *fastforce*
have *20*: $\vdash$ *f* $\wedge$ $\neg$(*wpowerstar g*) $\longrightarrow$ (*g* $\wedge$ *more*);*f* $\wedge$ $\neg$((*g* $\wedge$ *more*);*wpowerstar g*)
using *assms 15* **by** *fastforce*
have *21*: $\vdash$ *g* $\wedge$ *more* $\longrightarrow$ *more*
by *auto*
from *20 21* **show** *?thesis* **using** *ChopContraB*[*of f LIFT wpowerstar g LIFT* (*g* $\wedge$ *more*)]
by *blast*
qed

lemma *WPowerstarIntroMore*:
assumes $\vdash$ *f* $\wedge$ *more* $\longrightarrow$ (*g* $\wedge$ *more*); *f*
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *wpowerstar g*
proof –
have *14*: $\vdash$ *wpowerstar g* = (*empty* $\vee$ (*g* $\wedge$ *more*);(*wpowerstar g*))
using *WPowerstar-more-absorb*[*of g*] *WPowerstarEqv*[*of LIFT* (*g* $\wedge$ *more*)]
by (*metis int-eq*)
have *15*: $\vdash$ ($\neg$(*wpowerstar g*)) = (*more* $\wedge$ $\neg$((*g* $\wedge$ *more*);*wpowerstar g*))
using *14* **unfolding** *empty-d-def* **by** *fastforce*

have *20*: $\vdash$ *f* $\wedge$ $\neg$(*wpowerstar g*) $\longrightarrow$ (*g* $\wedge$ *more*);*f* $\wedge$ $\neg$((*g* $\wedge$ *more*);*wpowerstar g*)
using *assms 15* **by** *fastforce*
have *21*: $\vdash$ *g* $\wedge$ *more* $\longrightarrow$ *more*
by *auto*
from *20 21* **show** *?thesis* **using** *ChopContraB*[*of f LIFT wpowerstar g LIFT* (*g* $\wedge$ *more*)]
by *blast*
qed

lemma *Chopstar-WPowerstar*:
$\vdash$ *chopstar f* = *wpowerstar* (*f* $\wedge$ *more*)
by (*metis FPowerstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf chopstar-d-def inteq-reflection powerstar-d-def*)

lemma *Chopstar-FPowerstar*:
$\vdash$ *chopstar f* = (*fpowerstar f*);(*empty* $\vee$ (*f* $\wedge$ *inf*))
by (*metis Chopstar-WPowerstar FPowerstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf WPowerstar-more-absorb inteq-reflection*)

lemma *SChopstar-WPowerstar-more*:
$\vdash$ *schopstar f* = *wpowerstar* ((*f* $\wedge$ *more*) $\wedge$ *finite*)
by (*simp add*: *FPowerstar-WPowerstar schopstar-d-def*)

lemma *SChopstar-FPowerstar*:
$\vdash$ *schopstar f* = (*fpowerstar f*)
by (*simp add*: *FPowerstar-more-absorb schopstar-d-def*)

lemma *WPowerstar-skip-finite*:
$\vdash$ (*wpowerstar skip*) = *finite*
by (*metis EmptyImpFinite EmptyNextInducta Prop02 SkipChopFiniteImpFinite WPowerstar-1L WPowerstar-imp-empty WPowerstar-induct-lvar-empty int-iffI next-d-def*)

lemma *SPSEqvEmptyOrSChopSPS*:
$\vdash$ *fpowerstar f* = (*empty* $\vee$ *f*$\frown$*fpowerstar f*)
by (*simp add*: *FPowerstarEqv schop-d-def*)

lemma *ChopstarEqvPowerstar*:
$\vdash$ $f^\star$ = *powerstar f*
by (*metis Chopstar-FPowerstar powerstar-d-def*)

lemma *ChopplusCommute*:
$\vdash$ *f*;$f^\star$ = $f^\star$;*f*
by (*metis Chopstar-WPowerstar WPowerstar-more-absorb WPowerstar-slide-var int-eq*)

lemma *SChopplusCommute*:
$\vdash$ *f*$\frown$*schopstar f* = *schopstar f*$\frown$(*f* $\wedge$ *finite*)
by (*meson Prop04 SChopstar-sim3 int-simps*(*27*))

lemma *CSEqvOrChopCSB*:
$\vdash \; f^\star = \; (empty \;\lor\; (f^\star;f))$
by (*metis ChopplusCommute ChopstarEqvPowerstar PowerstarEqvSem intI inteq-reflection*)

lemma *SCSEqvOrChopSCSB*:
$\vdash \; schopstar\, f = \; (empty \;\lor\; (schopstar\, f \frown (f \land finite)))$
by (*metis SChopplusCommute SChopstar-FPowerstar SPSEqvEmptyOrSChopSPS inteq-reflection*)

lemma *CSChopCS*:
$\vdash f^\star \; ; f^\star = f^\star$
by (*metis Chopstar-WPowerstar WPowerstar-trans-eq inteq-reflection*)

lemma *SCSSChopSCS*:
$\vdash schopstar\, f \frown schopstar\, f = schopstar\, f$
by (*metis SChopstar-imp-empty SChopstar-invol SChopstar-sup-id-star1 inteq-reflection*)

lemma *CSCSEqvCS*:
$\vdash (f^\star)^\star \; = f^\star$
by (*metis* (*no-types*, *lifting*) *Chopstar-WPowerstar WPowerstar-invol WPowerstar-more-absorb int-eq*)

lemma *ChopPlusEqvOrChopChopPlus*:
$\vdash \; (f;f^\star) \; = (f \lor \; f; \, (f;f^\star))$
by (*metis CSEqvOrChopCSB ChopEmpty ChopOrEqv ChopplusCommute inteq-reflection*)

lemma *SChopPlusEqvOrSChopSChopPlus*:
$\vdash \; (f \frown schopstar\, f) \; = ((f \land finite) \lor \; f \frown (f \frown schopstar\, f))$
by (*metis ChopEmpty SChopOrEqvRule SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *ChopPlusEqv*:
$\vdash (f;f^\star) \; = (f \lor \; (f \land \; more \,); \, (f;f^\star))$
by (*metis ChopAssoc ChopplusCommute ChopstarEqv EmptyOrChopEqv inteq-reflection*)

lemma *SChopPlusEqv*:
$\vdash (f \frown schopstar\, f) \; = ((f \land finite) \lor \; (f \land \; more \,) \frown (f \frown schopstar\, f))$
by (*metis* (*no-types*, *opaque-lifting*) *ChopAssoc EmptyOrChopEqv Prop10 SChopstarEqv SChopplusCommute SChopstar-finite inteq-reflection schop-d-def*)

lemma *CSIntro*:
assumes $\vdash \; f \longrightarrow (g \land \; more \,); \, f$
shows $\vdash f \; \land finite \longrightarrow g^\star$
by (*metis Chopstar-WPowerstar WPowerstarIntro WPowerstar-more-absorb assms inteq-reflection*)

lemma *CSIntroMore*:
assumes $\vdash \; f \land more \longrightarrow (g \land \; more \,); \, f$

shows $\vdash f \ \wedge$ *finite* $\longrightarrow g^\star$
by (*metis Chopstar-WPowerstar WPowerstarIntroMore WPowerstar-more-absorb assms inteq-reflection*)

lemma *SCSIntro*:
assumes $\vdash$ $f \longrightarrow (g \wedge$ *more* $)\frown f$
shows $\vdash f \ \wedge$ *finite* $\longrightarrow$ *schopstar g*
proof −
have *1*: $\vdash((g \wedge$ *more*$) \wedge$ *finite*$) = ((g \wedge$ *finite*$) \wedge$ *more*$)$
by *auto*
show *?thesis*
using *assms SChopstar-WPowerstar*[*of g*] *WPowerstarIntro*[*of f LIFT* $(g \wedge$ *finite*$)$] *1*
by (*metis inteq-reflection schop-d-def*)
qed

lemma *SCSIntroMore*:
assumes $\vdash$ $f \wedge$ *more* $\longrightarrow (g \wedge$ *more* $)\frown f$
shows $\vdash f \ \wedge$ *finite* $\longrightarrow$ *schopstar g*
proof −
have *1*: $\vdash((g \wedge$ *more*$) \wedge$ *finite*$) = ((g \wedge$ *finite*$) \wedge$ *more*$)$
by *auto*
show *?thesis*
using *assms SChopstar-WPowerstar*[*of g*] *WPowerstarIntroMore*[*of f LIFT* $(g \wedge$ *finite*$)$] *1*
by (*metis inteq-reflection schop-d-def*)
qed

lemma *CSElim*:
assumes $\vdash$ *empty* $\longrightarrow g$
$\vdash (f \wedge$ *more* $);\ g \longrightarrow g$
shows $\vdash f^\star \longrightarrow g$
using *assms*
by (*metis Chopstar-WPowerstar Prop02 WPowerstar-induct-lvar-empty inteq-reflection*)

lemma *SCSElim*:
assumes $\vdash$ *empty* $\longrightarrow g$
$\vdash (f \wedge$ *more* $)\frown g \longrightarrow g$
shows $\vdash$ *schopstar f* $\longrightarrow g$
using *assms*
by (*metis Prop02 SChopstar-WPowerstar-more WPowerstar-induct-lvar-empty inteq-reflection schop-d-def*)

lemma *CSElimWithoutMore*:
assumes $\vdash$ *empty* $\longrightarrow g$
$\vdash f;\ g \longrightarrow g$
shows $\vdash f^\star \longrightarrow g$
using *assms*
by (*metis AndChopA CSElim Prop10 Prop12 int-eq*)

lemma *SCSElimWithoutMore*:
assumes $\vdash$ *empty* $\longrightarrow g$
$\vdash f\frown g \longrightarrow g$
shows $\vdash$ *schopstar f* $\longrightarrow g$

using *assms*
by (*metis Prop02 SChopstar-WPowerstar WPowerstar-induct-lvar-empty inteq-reflection schop-d-def*)

lemma *WPowerstarChopEqvChopOrRule*:
assumes $\vdash$ f= ((*wpowerstar g*); *h*)
shows $\vdash f = ((g;\, f) \vee\ h)$
proof –
have *1*: $\vdash f$ = ((*wpowerstar g*); *h*) **using** *assms* **by** *auto*
have *2*: $\vdash$ (*wpowerstar g*) = (*empty* $\vee$ (*g*; (*wpowerstar g*))) **by** (*simp add*: *WPowerstarEqv*)
hence *3*: $\vdash$ (*wpowerstar g*); *h* = (*h* $\vee$ ((*g*; (*wpowerstar g*)); *h*)) **by** (*rule EmptyOrChopEqvRule*)
have *4*: $\vdash$ (*g*; (*wpowerstar g*)); *h* = *g*; ((*wpowerstar g*); *h*) **by** (*meson ChopAssoc Prop11*)
hence *41*: $\vdash$ (*wpowerstar g*); *h* = (*h* $\vee$ *g*; ((*wpowerstar g*); *h*)) **using** *3* **by** *fastforce*
have *5*: $\vdash$ *g*; *f* = *g*; ((*wpowerstar g*); *h*) **using** *1* **by** (*rule RightChopEqvChop*)
hence *6*: $\vdash$ ((*wpowerstar g*); *h*) = (*h* $\vee$ *g*; *f*) **using** *41* **by** *fastforce*
hence *61*: $\vdash$ ((*wpowerstar g*); *h*) = ((*g*; *f*) $\vee$ *h*) **by** *auto*
from *1 61* **show** *?thesis* **by** *fastforce*
qed

lemma *CSChopEqvChopOrRule*:
assumes $\vdash$ f= ($g^\star$; *h*)
shows $\vdash f = ((g;\, f) \vee\ h)$
using *assms*
by (*metis Chopstar-WPowerstar WPowerstarChopEqvChopOrRule WPowerstar-more-absorb int-eq*)

lemma *SCSChopEqvSChopOrRule*:
assumes $\vdash$ f= (*schopstar g* $\frown$ *h*)
shows $\vdash f = ((g \frown f) \vee\ h)$
using *assms*
by (*metis* (*no-types*, *lifting*) *FPowerstar-WPowerstar FPowerstar-more-absorb Prop10 SChopstar-finite WPowerstarChopEqvChopOrRule int-eq schop-d-def schopstar-d-def*)

lemma *WPowerstarChopIntroRule*:
assumes $\vdash f \wedge \neg\ h \longrightarrow g;\, f$
$\vdash g \longrightarrow$ *more*
shows $\vdash f \wedge$ *finite* $\longrightarrow$ (*wpowerstar g*); *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*: $\vdash g;\, f \longrightarrow (g\ );\, f$
by *auto*
have *5*: $\vdash f \longrightarrow (g\ );\, f \vee\ h$
using *1 4* **by** *fastforce*
have *6*: $\vdash$ *wpowerstar g* = (*empty* $\vee$ *g* ; *wpowerstar g*)
by (*simp add*: *WPowerstarEqv*)

hence *7*: $\vdash$ (*wpowerstar g*); *h* = (*h* $\vee$ (*g*; *wpowerstar g*); *h*)
  by (*rule EmptyOrChopEqvRule*)
have *8*: $\vdash$ (*g* ; *wpowerstar g*); *h* = *g* ; (*wpowerstar g*; *h*)
  by (*meson ChopAssoc Prop11*)
have *9*: $\vdash$ (*wpowerstar g*); *h* = (*h* $\vee$ *g* ; (*wpowerstar g*; *h*))
  using *7 8* **by** *fastforce*
have *10*: $\vdash$ *f* $\wedge$ $\neg$ (*wpowerstar g*; *h*) $\longrightarrow$ (*g*); *f* $\wedge$ $\neg$ ((*g*); (*wpowerstar g*; *h*))
  using *5 9* **by** *fastforce*
have *11*: $\vdash$ *g* $\wedge$ *more* $\longrightarrow$ *more*
  by *fastforce*
from *10 11* **show** *?thesis* **using** *ChopContraB* **using** *2* **by** *blast*
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*
using *assms*
by (*metis Chopstar-WPowerstar Prop10 WPowerstarChopIntroRule inteq-reflection*)

lemma *SCSChopIntroRule*:
assumes $\vdash$ *f* $\wedge$ $\neg$ *h* $\longrightarrow$ $g\frown f$
$\vdash$ *g* $\longrightarrow$ *more*
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *schopstar* $g\frown h$
using *assms SChopstar-WPowerstar*[*of g*] *WPowerstarChopIntroRule*[*of f h LIFT* (*g* $\wedge$ *finite*)]
unfolding *schop-d-def*
by (*metis Prop05 Prop07 Prop10 SChopstar-finite finite-d-def inteq-reflection*)

lemma *DiamondAndEmptyEqvAndEmpty*:
$\vdash$ ($\Diamond$ *f* $\wedge$ *empty*) = (*f* $\wedge$ *empty*)
by (*metis ChopAndEmptyEqvEmptyChopEmpty EmptyChop FiniteAndEmptyEqvEmpty int-eq sometimes-d-def*)

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* $\wedge$ $\neg$($\square$ (*init w*)) $\longrightarrow$ $\neg$ *empty*
proof –
have *1*: $\vdash$ ((*init w*) $\wedge$ *empty*) = (*w* $\wedge$ *empty*)
by (*rule InitAndEmptyEqvAndEmpty*)
have *2*: $\vdash$ ($\neg$($\square$ (*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*: $\vdash$ ($\neg$($\square$ (*init w*)) $\wedge$ *empty*) = ($\neg$ *w* $\wedge$ *empty*)
using *2 3 5* **by** *fastforce*
have *7*: $\vdash$ $\neg$(*init w* $\wedge$ $\neg$($\square$ (*init w*)) $\wedge$ *empty*)
using *1 6* **by** *fastforce*
from *7* **show** *?thesis* **by** *auto*
qed

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

lemma *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*:
$\vdash$ $\square$(*init w*) $\wedge$ *more* $\longrightarrow$ ($\square$ (*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*: $\vdash$ $\square$(*init w*) $\longrightarrow$ *#True*;(*init w* $\wedge$ *empty*) **by** (*rule BoxImpTrueChopAndEmpty*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpTrueSChopAndEmpty*:
$\vdash$ $\square$ *f* $\wedge$ *finite*$\longrightarrow$ *#True*$\frown$(*f* $\wedge$ *empty*)
by (*metis BoxAndSChopImport DiamondEmptyEqvFinite TrueSChopEqvDiamond inteq-reflection*)

lemma *BoxInitAndMoreImpBoxInitAndMoreAndSFinInit*:
$\vdash$ $\square$(*init w*) $\wedge$ *more* $\wedge$ *finite* $\longrightarrow$ ($\square$ (*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*: $\vdash$ $\square$(*init w*) $\wedge$ *finite* $\longrightarrow$ *#True*$\frown$(*init w* $\wedge$ *empty*) **by** (*rule BoxImpTrueSChopAndEmpty*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *CSImpBox*:
assumes $\vdash$ *f* $\longrightarrow$ *empty* $\vee$ (($\square$ (*init w*) $\wedge$ *more*) $\wedge$ *finite*); *f*
shows $\vdash$ (*init w* $\wedge$ *f*) $\wedge$ *finite* $\longrightarrow$ $\square$ (*init w*)
proof –
have *1*: $\vdash$ *f* $\longrightarrow$ *empty* $\vee$ (($\square$(*init w*) $\wedge$ *more*) $\wedge$ *finite*); *f*
using *assms* **by** *auto*
have *2*: $\vdash$ *init w* $\wedge$ $\neg$($\square$ (*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 *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*)
using *assms* **by** (*simp add*: *CSImpBox schop-d-def*)

lemma *ChopstarInductR*:
assumes $\vdash$ *g* $\vee$ *h*;*f* $\longrightarrow$ *h*
shows $\vdash$ *g*;(*chopstar f*) $\longrightarrow$ *h*
by (*metis Chopstar-WPowerstar WPowerstarInductR WPowerstar-more-absorb assms int-eq*)

lemma *BoxWPowerstarEqvBox*:
$\vdash$ (*init w* $\wedge$ *wpowerstar* ($\Box$(*init w*))) = $\Box$ (*init w*)

proof –
have *1*: $\vdash$ □(*init w*);□(*init w*) ⟶ □(*init w*)
by (*simp add*: *BoxStateChopBoxEqvBox int-iffD1*)
have *2*: $\vdash$ (*init w* ∧ *empty*) ⟶ □(*init w*)
by (*simp add*: *StateAndEmptyImpBoxState*)
have *3*: $\vdash$ (*init w* ∧ *empty*) ∨ □(*init w*);□(*init w*) ⟶ □(*init w*)
using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*init w* ∧ *empty*);*wpowerstar* (□(*init w*)) ⟶ □ (*init w*)
using *3 WPowerstarInductR* **by** *blast*
have *5*: $\vdash$ *init w* ∧ *wpowerstar* (□(*init w*)) ⟶ □ (*init w*)
using *4 StateAndEmptyChop* **by** *fastforce*
have *11*: $\vdash$ □ (*init w*) ⟶ (*init w*)
using *BoxElim* **by** *blast*
have *12*: $\vdash$ □(*init w*) ⟶ *wpowerstar* (□ (*init w*))
by (*simp add*: *WPowerstar-ext*)
have *13*: $\vdash$ □ (*init w*) ⟶ *init w* ∧ *wpowerstar* (□ (*init w*))
using *11 12* **by** *fastforce*
from *5 13* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxCSEqvBox*:
$\vdash$ (*init w* ∧ (□(*init w*))*) = □ (*init w*)
by (*metis BoxWPowerstarEqvBox Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)

lemma *BoxSCSEqvBox*:
$\vdash$ (*init w* ∧ *schopstar* (□(*init w*))) = (□ (*init w*) ∧ *finite*)
proof –
have *1*: $\vdash$ □(*init w*)⌢(□(*init w*) ∧ *finite*)⟶ □(*init w*) ∧ *finite*
by (*metis BoxStateAndChopEqvChop FiniteChopFiniteEqvFinite int-iffD2 inteq-reflection schop-d-def*)
have *2*: $\vdash$ (*init w* ∧ *empty*) ⟶ □(*init w*) ∧ *finite*
using *EmptyImpFinite StateAndEmptyImpBoxState* **by** *fastforce*
have *3*: $\vdash$ (*init w* ∧ *empty*) ∨ □(*init w*)⌢(□(*init w*) ∧ *finite*) ⟶ □(*init w*) ∧ *finite*
using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*init w* ∧ *empty*)⌢*schopstar* (□(*init w*)) ⟶ □ (*init w*) ∧ *finite*
using *SChopstarInductR 3*
by (*metis Prop12 SChopImpFinite SChopstar-finite SChopstar-finite-absorb inteq-reflection*)
have *5*: $\vdash$ *init w* ∧ *schopstar* (□(*init w*)) ⟶ □ (*init w*) ∧ *finite*
using *4 StateAndEmptySChop* **by** *fastforce*
have *11*: $\vdash$ □ (*init w*) ⟶ (*init w*)
using *BoxElim* **by** *blast*
have *12*: $\vdash$ □(*init w*) ∧ *finite* ⟶ *schopstar* (□ (*init w*))
by (*metis SChopstar-WPowerstar WPowerstar-ext inteq-reflection*)
have *13*: $\vdash$ □ (*init w*) ∧ *finite* ⟶ *init w* ∧ *schopstar* (□ (*init w*))
using *11 12* **by** *fastforce*
from *5 13* **show** *?thesis* **by** *fastforce*
qed

lemma *WpowerstarAndMoreEqvAndMoreChop*:

$\vdash$ (*wpowerstar f* $\wedge$ *more*) = (*f* $\wedge$ *more*); *wpowerstar f*
proof –
have *1*: $\vdash$ (*empty* $\vee$ (*f* $\wedge$ *more*); *wpowerstar f*) $\wedge$ *more* $\longrightarrow$ (*f* $\wedge$ *more*); *wpowerstar f*
by (*auto simp*: *empty-d-def*)
have *2*: $\vdash$ *wpowerstar f* = (*empty* $\vee$ (*f* $\wedge$ *more*);*wpowerstar f*)
by (*metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
have *3*: $\vdash$ *wpowerstar f* $\wedge$ *more* $\longrightarrow$ (*f* $\wedge$ *more*); *wpowerstar f*
using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*f* $\wedge$ *more*); *wpowerstar f* $\longrightarrow$ *wpowerstar f*
using *2* **by** *fastforce*
have *5*: $\vdash$ (*f* $\wedge$ *more*) $\longrightarrow$ *more*
by *auto*
hence *6*: $\vdash$ (*f* $\wedge$ *more*); *wpowerstar f* $\longrightarrow$ *more*
by (*rule LeftChopImpMoreRule*)
have *7*: $\vdash$ (*f* $\wedge$ *more*); *wpowerstar f* $\longrightarrow$ *wpowerstar f* $\wedge$ *more*
using *4 6* **by** *fastforce*
from *3 7* **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndMoreEqvAndMoreChop*:
$\vdash$ ($f^{\star}$ $\wedge$ *more*) = (*f* $\wedge$ *more*); $f^{\star}$
by (*metis Chopstar-WPowerstar WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop int-eq*)

lemma *SCSAndMoreEqvAndMoreSChop*:
$\vdash$ (*schopstar f* $\wedge$ *more*) = (*f* $\wedge$ *more*)$\frown$ *schopstar f*
proof –
have *1*: $\vdash$((*f* $\wedge$ *more*) $\wedge$ *finite*) = ((*f* $\wedge$ *finite*) $\wedge$ *more*)
by *auto*
show *?thesis*
using *SChopstar-WPowerstar*[*of f*] *WPowerstar-more-absorb*[*of LIFT* (*f* $\wedge$ *finite*)]
WpowerstarAndMoreEqvAndMoreChop[*of LIFT* (*f* $\wedge$ *finite*)]
by (*metis 1 int-eq schop-d-def*)
qed

lemma *SCSAndMoreEqvAndFMoreSChop*:
$\vdash$ (*schopstar f* $\wedge$ *fmore*) = (*f* $\wedge$ *more*)$\frown$ *schopstar f*
by (*metis AndMoreAndFiniteEqvAndFmore Prop10 SCSAndMoreEqvAndMoreSChop SChopImpFinite SChopstar-finite*
inteq-reflection)

lemma *BoxStateAndWPowerstarEqvWPowerstar*:
$\vdash$ ($\Box$(*init w*) $\wedge$ *wpowerstar f* $\wedge$ *finite*) = (*init w* $\wedge$ *wpowerstar* ($\Box$(*init w*) $\wedge$ *f*) $\wedge$ *finite*)
proof –
have *1*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ *init w*
using *BoxElim* **by** *blast*
have *2*: $\vdash$ (*wpowerstar f* $\wedge$ *more*) = (*f* $\wedge$ *more*); *wpowerstar f*
by (*simp add*: *WpowerstarAndMoreEqvAndMoreChop*)
have *3*: $\vdash$ ($\Box$(*init w*) $\wedge$ ((*f* $\wedge$ *more*); *wpowerstar f*)) =

$((\Box\ (init\ w) \wedge f \wedge\ \ more\ ); (\Box\ (init\ w) \wedge wpowerstar\ f))$
by (*rule BoxStateAndChopEqvChop*)
have *4*: $\vdash \Box\ (init\ w) \wedge f \wedge\ \ more\ \longrightarrow (\Box\ (init\ w) \wedge f) \wedge\ \ more$
by *auto*
hence *5*: $\vdash (\Box\ (init\ w) \wedge f \wedge\ \ more\ ); (\Box\ (init\ w) \wedge wpowerstar\ f) \longrightarrow$
$((\Box\ (init\ w) \wedge f) \wedge\ \ more\ ); (\Box\ (init\ w) \wedge wpowerstar\ f)$
by (*rule LeftChopImpChop*)
have *6*: $\vdash (\Box(\ init\ w) \wedge wpowerstar\ f) \wedge\ \ more\ \ \longrightarrow$
$((\Box\ (init\ w) \wedge f) \wedge\ \ more\ ); (\Box\ (init\ w) \wedge wpowerstar\ f)$
using *2 3 5* **by** *fastforce*
hence *7*: $\vdash (\Box\ (init\ w) \wedge wpowerstar\ f) \wedge finite \longrightarrow wpowerstar\ (\Box\ (init\ w) \wedge f)$
using *WPowerstarIntroMore*[*of LIFT* $(\Box\ (init\ w) \wedge wpowerstar\ f)$ *LIFT* $(\Box\ (init\ w) \wedge f)$]
by *blast*
have *71*: $\vdash init\ w \wedge \Box\ (init\ w) \wedge wpowerstar\ f \wedge finite \longrightarrow init\ w \wedge wpowerstar\ (\Box\ (init\ w) \wedge f) \wedge finite$
using *7* **by** *fastforce*
have *8*: $\vdash \Box(\ init\ w) \wedge wpowerstar\ f \wedge finite \longrightarrow init\ w \wedge wpowerstar\ (\Box\ (init\ w) \wedge f) \wedge finite$
using *1 71* **by** *fastforce*
have *11*: $\vdash wpowerstar\ (\Box\ (init\ w) \wedge f) \longrightarrow wpowerstar\ (\Box\ (init\ w))$
by (*meson Prop12 WPowerstar-iso int-iffD2 lift-and-com*)
have *12*: $\vdash (init\ w \wedge wpowerstar\ (\Box\ (init\ w))) = \Box\ (init\ w)$
by (*simp add*: *BoxWPowerstarEqvBox*)
have *13*: $\vdash wpowerstar\ (\Box\ (init\ w) \wedge f) \longrightarrow wpowerstar\ f$
by (*meson Prop12 WPowerstar-iso int-iffD2 lift-and-com*)
have *14*: $\vdash init\ w \wedge wpowerstar\ (\Box\ (init\ w) \wedge f) \longrightarrow init\ w \wedge wpowerstar\ (\Box\ (init\ w)) \wedge wpowerstar\ f$
using *11 13* **by** *fastforce*
have *15*: $\vdash init\ w \wedge wpowerstar\ (\Box\ (init\ w)) \wedge wpowerstar\ f \longrightarrow \Box\ (init\ w) \wedge wpowerstar\ f$
using *12* **by** *auto*
have *16*: $\vdash init\ w \wedge wpowerstar\ (\Box\ (init\ w) \wedge f) \longrightarrow \Box\ (init\ w) \wedge wpowerstar\ f$
using *14 15 lift-imp-trans* **by** *blast*
from *8 16* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateAndCSEqvCS*:
$\vdash\ \ (\Box(\ init\ w) \wedge f^{\star} \wedge finite) = (init\ w \wedge (\Box(\ init\ w) \wedge f)^{\star} \wedge finite)$
by (*metis BoxStateAndWPowerstarEqvWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *BoxStateAndSCSEqvSCS*:
$\vdash\ \ (\Box(\ init\ w) \wedge schopstar\ f) = (init\ w \wedge schopstar\ (\Box(\ init\ w) \wedge f))$
proof –
have *1*: $\vdash (\Box\ (init\ w) \wedge f \wedge finite) = ((\Box\ (init\ w) \wedge f) \wedge finite)$
by *auto*
show *?thesis*
using *BoxStateAndWPowerstarEqvWPowerstar*[*of w LIFT* $(f \wedge finite)$]
SChopstar-WPowerstar[*of f*] *SChopstar-WPowerstar*[*of LIFT* $(\Box(\ init\ w) \wedge f)$]
SChopstar-finite[*of f*] *SChopstar-finite*[*of LIFT* $(\Box(\ init\ w) \wedge f)$]
by (*metis 1 Prop10 inteq-reflection*)
qed

lemma *BaWPowerstarImpWPowerstar*:

$\vdash$ *ba* $(f \longrightarrow g) \wedge$ *finite* $\longrightarrow$ *wpowerstar f* $\longrightarrow$ *wpowerstar g*
proof –
have *1*: $\vdash$ *wpowerstar f* = (*empty* $\vee$ (*f* $\wedge$ *more*); *wpowerstar f*)
by (*metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
have *2*: $\vdash$ *wpowerstar g* = (*empty* $\vee$ (*g* $\wedge$ *more*); *wpowerstar g*)
by (*metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
have *21*: $\vdash$ $\neg$(*wpowerstar g*) = ($\neg$*empty* $\wedge$ $\neg$((*g* $\wedge$ *more*); *wpowerstar g*))
using *2* **by** *fastforce*
hence *22*: $\vdash$ $\neg$(*wpowerstar g*) = (*more* $\wedge$ $\neg$((*g* $\wedge$ *more*); *wpowerstar g*))
by (*simp add*: *empty-d-def*)
have *3*: $\vdash$ *wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*) $\longrightarrow$
(*empty* $\vee$ (*f* $\wedge$ *more*); *wpowerstar f*) $\wedge$ *more* $\wedge$ $\neg$ ((*g* $\wedge$ *more*); *wpowerstar g*)
using *1 22* **by** *fastforce*
have *31*: $\vdash$ ((*empty* $\vee$ (*f* $\wedge$ *more*); *wpowerstar f*) $\wedge$ *more*) = ((*f* $\wedge$ *more*); *wpowerstar f* $\wedge$ *more*)
by (*auto simp*: *empty-d-def*)
have *32*: $\vdash$ *wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*) $\longrightarrow$ (*f* $\wedge$ *more*); *wpowerstar f* $\wedge$ $\neg$ ((*g* $\wedge$ *more*); *wpowerstar g*)
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*); *wpowerstar f* $\longrightarrow$ (*g* $\wedge$ *more*); *wpowerstar f*
by (*rule BaLeftChopImpChop*)
have *7*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ (*f* $\wedge$ *more*); *wpowerstar f* $\longrightarrow$ (*g* $\wedge$ *more*); *wpowerstar f*
using *5 6* **by** *fastforce*
have *8*: $\vdash$ (*g* $\wedge$ *more*); *wpowerstar f* $\wedge$ $\neg$ ((*g* $\wedge$ *more*); *wpowerstar g*)
$\longrightarrow$ (*g* $\wedge$ *more*); (*wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*))
by (*rule ChopAndNotChopImp*)
have *9*: $\vdash$ (*g* $\wedge$ *more*); (*wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*)) $\longrightarrow$ *more* ; (*wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*))
by (*rule AndChopB*)
have *10*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ *more* ; (*wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*)) $\longrightarrow$
more ; (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*))
by (*rule BaChopImpChopBa*)
have *11*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\wedge$ *wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*) $\longrightarrow$
more ; (*ba* (*f* $\longrightarrow$ *g*) $\wedge$ *wpowerstar f* $\wedge$ $\neg$ (*wpowerstar g*))
using *32 7 8 9 10* **by** *fastforce*
hence *12*: $\vdash$ *finite* $\longrightarrow$ $\neg$ ((*ba* (*f* $\longrightarrow$ *g*)) $\wedge$ (*wpowerstar f*) $\wedge$ ($\neg$ (*wpowerstar g*)))
using *MoreChopLoopFiniteB* **by** *blast*
from *12* **show** *?thesis* **by** (*simp add*: *Valid-def*)
qed

lemma *BaCSImpCS*:
$\vdash$ *ba* $(f \longrightarrow g) \wedge$ *finite* $\longrightarrow f^\star \longrightarrow g^\star$
by (*metis BaWPowerstarImpWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SDa-Da*:
⊢ *sda f* = *da* (*f* ∧ *finite*)
unfolding *sda-d-def da-d-def schop-d-def*
by *simp*

lemma *SBa-Ba*:
⊢ *sba f* = *ba* (*finite* ⟶ *f*)
proof –
have *1*: ⊢ (¬ (*finite* ⟶ *f*)) = (*finite* ∧ ¬ *f*)
by *auto*
show *?thesis*
unfolding *sba-d-def ba-d-def sda-d-def da-d-def schop-d-def*
by (*metis 1 AndChopCommute int-eq int-simps*(*17*))
qed

lemma *SBaSCSImpSCS*:
⊢ *sba* (*f* ⟶ *g*) ⟶ *schopstar f* ⟶ *schopstar g*
proof –
have *1*: ⊢ *ba* (*finite* ⟶ *f* ⟶ *g*) ⟶ *ba* (*f* ∧ *finite* ⟶ *g* ∧ *finite*)
by (*simp add*: *BaImpBa intI*)
have *2*: ⊢ *schopstar f* ⟶ *finite*
by (*simp add*: *SChopstar-finite*)
show *?thesis*
using *BaWPowerstarImpWPowerstar*[*of LIFT* (*f* ∧ *finite*) *LIFT* (*g* ∧ *finite*)]
SChopstar-WPowerstar[*of f*] *SChopstar-WPowerstar*[*of g*]
SBa-Ba[*of LIFT* (*f* ⟶ *g*)] *1 2* **by** *fastforce*
qed

lemma *BaWPowerstarEqvWPowerstar*:
⊢ *ba* (*f* = *g*) ∧ *finite* ⟶ (*wpowerstar f* = *wpowerstar g*)
proof –
have *0*: ⊢ (*f* = *g*) = ((*f* ⟶ *g*) ∧ (*g* ⟶ *f*))
by *fastforce*
have *1*: ⊢ *ba* (*f* = *g*) = (*ba* (*f* ⟶ *g*) ∧ *ba* (*g* ⟶ *f*))
by (*metis 0 BaAndEqv int-eq*)
have *2*: ⊢ *ba* (*f* ⟶ *g*) ∧ *finite* ⟶ (*wpowerstar f* ⟶ *wpowerstar g*)
by (*rule BaWPowerstarImpWPowerstar*)
have *3*: ⊢ *ba* (*g* ⟶ *f*) ∧ *finite* ⟶ (*wpowerstar g* ⟶ *wpowerstar f*)
by (*rule BaWPowerstarImpWPowerstar*)
have *4*: ⊢ *ba* (*f* = *g*) ∧ *finite* ⟶ (*wpowerstar f* ⟶ *wpowerstar g*) ∧ (*wpowerstar g* ⟶ *wpowerstar f*)
using *1 2 3* **by** *fastforce*
have *5*: ⊢ ((*wpowerstar f* ⟶ *wpowerstar g*) ∧ (*wpowerstar g* ⟶ *wpowerstar f*)) = (*wpowerstar f* = *wpowerstar g*)
by *auto*
from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *BaCSEqvCS*:

$\vdash \quad ba\ (f = g) \wedge finite \longrightarrow (f^{\star} = g^{\star})$
by (*metis BaWPowerstarEqvWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SBaSCSEqvSCS*:
$\vdash \quad sba\ (f = g) \longrightarrow (schopstar\ f = schopstar\ g)$
proof –
have *0*: $\vdash (f = g) = (\ (f \longrightarrow g) \wedge (g \longrightarrow f))$
by *fastforce*
have *1*: $\vdash sba\ (f = g) = (sba\ (f \longrightarrow g) \wedge sba\ (g \longrightarrow f))$
by (*metis 0 SBaAndEqv int-eq*)
show *?thesis* **using** *SBaSCSImpSCS*[*of f g*] *SBaSCSImpSCS*[*of g f*]
1 **by** *fastforce*
qed

lemma *BaAndWPowerstarImport*:
$\vdash \quad ba\ \ f \wedge wpowerstar\ g \wedge finite \longrightarrow wpowerstar\ (f \wedge g)$
proof –
have *1*: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
hence *2*: $\vdash\ \ ba\ \ f \longrightarrow\ \ ba\ \ (g \longrightarrow f \wedge g)$ **by** (*rule BaImpBa*)
have *3*: $\vdash\ \ ba\ \ (g \longrightarrow f \wedge g) \wedge finite \longrightarrow wpowerstar\ g \longrightarrow wpowerstar\ (f \wedge g)$
by (*rule BaWPowerstarImpWPowerstar*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndCSImport*:
$\vdash \quad ba\ \ f \wedge g^{\star} \wedge finite \longrightarrow (f \wedge g)^{\star}$
by (*metis BaAndWPowerstarImport Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SBaAndSCSImport*:
$\vdash \quad sba\ \ f \wedge schopstar\ g \longrightarrow schopstar\ (f \wedge g)$
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

8.7 Len

lemma *wpower-len*:
$\vdash wpower\ skip\ k = len\ k$
by (*simp add*: *len-d-def*)

lemma *wpowerstar-skip-finite*:
$\vdash finite = wpowerstar\ skip$
using *WPowerstar-skip-finite* **by** *fastforce*

lemma *schopstar-skip-finite*:
$\vdash finite = schopstar\ skip$
by (*metis Prop10 SChopstar-WPowerstar WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)

lemma *wpowerstar-skip-fmore*:
$\vdash$ *fmore* = *skip*;*wpowerstar skip*
by (*metis FmoreEqvSkipChopFinite integ-reflection wpowerstar-skip-finite*)

lemma *schopstar-skip-fmore*:
$\vdash$ *fmore* = *skip*$\frown$*schopstar skip*
by (*metis NextSChopdef WPowerstar-skip-finite integ-reflection next-d-def schopstar-skip-finite wpowerstar-skip-fmore*)

lemma *len-k-finite*:
$\vdash$ *len k* $\longrightarrow$ *finite*
proof (*induct k*)
case *0*
then show *?case*
by (*metis EmptyImpFinite len-d-def wpow-0*)
next
case (*Suc k*)
then show *?case*
proof –
have *1*: $\vdash$ *len* (*Suc k*) = *skip*;*len k*
by (*simp add*: *len-d-def*)
have *2*: $\vdash$ *skip* $\longrightarrow$ *finite*
by (*metis WPowerstar-ext WPowerstar-skip-finite integ-reflection*)
show *?thesis*
by (*metis 1 2 ChopImpChop FiniteChopFiniteEqvFinite Suc integ-reflection*)
qed
qed

lemma *len-k-schop*:
$\vdash$ *len Suc k* = *len k* $\frown$ *skip*
unfolding *len-d-def* **unfolding** *schop-d-def*
by (*metis Prop10 WPowerCommute int-eq len-k-finite wpow-Suc wpower-len*)

lemma *SkipChopAnd*:
$\vdash$ ((*skip*;*f*) $\wedge$ (*skip*;*g*)) = *skip*;(*f* $\wedge$ *g*)
proof –
have *1*: $\vdash$ *skip*;(*f* $\wedge$ *g*) $\longrightarrow$ ((*skip*;*f*) $\wedge$ (*skip*;*g*))
by (*simp add*: *ChopAndA ChopAndB Prop12*)
have *2*: $\vdash$ ((*skip*;*f*) $\wedge$ (*skip*;*g*)) $\longrightarrow$ *skip*;(*f* $\wedge$ *g*)
by (*metis NextAndEqvNextAndNext SkipChopEqvNext int-eq int-iffD2*)
show *?thesis*
using *1 2 int-iffI* **by** *blast*
qed

lemma *SkipSChopAnd*:
$\vdash$ ((*skip*$\frown$*f*) $\wedge$ (*skip*$\frown$*g*)) = *skip*$\frown$(*f* $\wedge$ *g*)
by (*metis NextAndNextEqvNextRule NextSChopdef integ-reflection lift-and-com*)

lemma *LenChopAnd*:
⊢ (*len k*;*f* ∧ *len k*;*g*) = *len k*;(*f* ∧ *g*)
proof –
 have *2*: ⊢ *len k*;(*f* ∧ *g*) ⟶ (*len k*;*f* ∧ *len k*;*g*)
  by (*simp add*: *ChopAndA ChopAndB Prop12*)
 have *1*: ⊢ (*len k*;*f* ∧ *len k*;*g*) ⟶ *len k*;(*f* ∧ *g*)
  proof (*induct k arbitrary*: *f g*)
  case *0*
  then show *?case*
  by (*metis EmptyChop int-iffD2 inteq-reflection wpow-0 wpower-len*)
  next
  case (*Suc k*)
  then show *?case*
   proof –
    have *1*: ⊢ *len Suc k*;*f* = *len k*;(*skip*;*f*)
     using *WPowerCommute*[*of LIFT skip k*] *ChopAssoc*[*of LIFT len k LIFT skip f*]
     by (*metis inteq-reflection wpow-Suc wpower-len*)
    have *2*: ⊢ *len Suc k*;*g* = *len k*;(*skip*;*g*)
     using *WPowerCommute*[*of LIFT skip k*] *ChopAssoc*[*of LIFT len k LIFT skip g*]
     by (*metis inteq-reflection wpow-Suc wpower-len*)
    have *3*: ⊢ (*len k*;(*skip*;*f*) ∧ *len k*;(*skip*;*g*)) ⟶ *len k*;((*skip*;*f*) ∧ (*skip*;*g*))
     by (*simp add*: *Suc*)
    have *4*: ⊢ ((*skip*;*f*) ∧ (*skip*;*g*)) = *skip*;(*f* ∧ *g*)
     by (*simp add*: *SkipChopAnd*)
    have *5*: ⊢ *len k*;((*skip*;*f*) ∧ (*skip*;*g*)) = *len* (*Suc k*);(*f* ∧ *g*)
     using *WPowerCommute*[*of LIFT skip k*] *ChopAssoc*[*of LIFT len k LIFT skip LIFT* (*f* ∧ *g*)]
     by (*metis 4 inteq-reflection wpow-Suc wpower-len*)
    show *?thesis*
    by (*metis 1 2 3 5 int-eq*)
   qed
  qed
 show *?thesis*
 using *1 2 int-iffI* **by** *blast*
qed

lemma *LenSChopAnd*:
⊢ (*len k*⌢*f* ∧ *len k*⌢*g*) = *len k*⌢(*f* ∧ *g*)
proof –
 have *1*: ⊢ *len k* ⟶ *finite*
  using *len-k-finite* **by** *blast*
 show *?thesis* **unfolding** *schop-d-def*
 by (*metis 1 LenChopAnd Prop10 int-eq*)
qed

lemma *LenEqvLenChopLen*:
⊢ *len*(*i*+*j*) = *len*(*i*);*len*(*j*)
proof
 (*induct i*)
 case *0*
 then show *?case*

by (*metis EmptyChop Prop11 add-cancel-left-left len-d-def wpow-0*)
next
case (*Suc i*)
then show *?case*
by (*metis ChopAssoc add-Suc int-eq len-d-def wpow-Suc*)
qed

lemma *LenChopFalse*:
$\vdash$ *len k* ;*#False* $\longrightarrow$ *#False*
by (*metis AndInfEqvChopFalse ChopImpChop InfEqvNotFinite int-iffD1 int-simps*(*21*) *inteq-reflection len-k-finite*)

lemma *LenSChopFalse*:
$\vdash$ *len k* $\frown$*#False* $\longrightarrow$ *#False*
by (*metis AndChopB InfEqvNotFinite TrueChopAndFiniteEqvAndFiniteChopFinite infinite-d-def int-eq int-simps*(*19*) *int-simps*(*22*) *schop-d-def*)

lemma *len-len-suc-not*:
$\vdash$ ¬(*len k* ∧ *len* (*Suc k*);*f*)
proof –
have *1*: $\vdash$ *len k* = *len k*;*empty*
by (*meson ChopEmpty Prop11*)
have *2*: $\vdash$ *len* (*Suc k*);*f* = *len k*;(*skip*;*f*)
using *ChopAssoc*[*of LIFT len k LIFT skip f*] *WPowerCommute*[*of LIFT skip k*]
by (*metis inteq-reflection wpow-Suc wpower-len*)
have *3*: $\vdash$ (*len k*;*empty* ∧ *len k*;(*skip*;*f*)) = *len k*;(*empty* ∧ *skip*;*f*)
by (*simp add*: *LenChopAnd*)
have *4*: $\vdash$ (*empty* ∧ *skip*;*f*) $\longrightarrow$ *#False*
unfolding *empty-d-def more-d-def next-d-def*
by (*metis ChopAndB Prop01 int-simps*(*16*) *int-simps*(*25*) *int-simps*(*4*) *inteq-reflection*)
have *5*: $\vdash$ (*len k* ∧ *len* (*Suc k*);*f*) $\longrightarrow$ *len k*;*#False*
by (*metis 1 2 3 4 RightChopImpChop inteq-reflection*)
have *6*: $\vdash$ *len k* ;*#False* $\longrightarrow$ *#False*
using *LenChopFalse* **by** *blast*
show *?thesis*
by (*metis 5 6 int-simps*(*14*) *inteq-reflection lift-imp-trans*)
qed

lemma *len-len-suc-not-schop*:
$\vdash$ ¬(*len k* ∧ *len* (*Suc k*)$\frown$*f*)
unfolding *schop-d-def*
by (*metis AndInfEqvChopFalse LenChopFalse Prop09 Prop10 int-eq int-simps*(*14*) *itl-def*(*8*) *len-len-suc-not*)

lemma *Finite-exist-len*:
$\vdash$ *finite* = (∃ *k*. *len k*)
by (*metis ExEqvRule WPowerstar-skip-finite int-eq wpower-len wpowerstar-d-def*)

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** (*simp add*: *ChopEmpty len-d-def*)
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*
by (*simp add*: *WPowerCommute len-d-def*)

lemma *NotFixedChop*:
$\vdash$ (¬((*g* ∧ *len*(*k*));*f*)) = (¬(*di*(*g* ∧ *len*(*k*))) ∨ ((*g* ∧ *len*(*k*));(¬*f*)))
by (*auto simp add*: *itl-defs len-defs Valid-def min-def nlength-eq-enat-nfiniteD*)

8.8 Properties of While

lemma *SChopstar-Chopstar*:
$\vdash$ *schopstar f* = *chopstar* (*f* ∧ *finite*)
by (*metis Chopstar-WPowerstar SChopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)

lemma *SWhile-While*:
$\vdash$ *swhile g do f* = *while g do* (*f* ∧ *finite*)
proof –
have *1*: $\vdash$ ((*g* ∧ *f*) ∧ *finite*) = (*g* ∧ *f* ∧ *finite*)
by *auto*
have *2*: $\vdash$ *chopstar* ((*g* ∧ *f*) ∧ *finite*) = *chopstar* (*g* ∧ *f* ∧ *finite*)
using *1* **by** (*metis Chopstardef int-eq*)
show *?thesis*
unfolding *swhile-d-def while-d-def*
using *SFinEQvFinAndFinite*[*of LIFT* ¬ *g*] *SChopstar-Chopstar*[*of LIFT* (*g* ∧ *f*)]
2 SChopstar-finite[*of LIFT* (*g* ∧ *f*)] **by** *fastforce*
qed

lemma *IfAndFiniteDist*:
$\vdash$ (*if$_i$* (*init w*) *then* (*f*;*g*) *else empty* ∧ *finite*) =
(*if$_i$* (*init w*) *then* ((*f* ∧ *finite*);(*g* ∧ *finite*)) *else empty*)
proof –
have *1*: $\vdash$ (*if$_i$* (*init w*) *then* (*f*;*g*) *else empty* ∧ *finite*) =
(((*init w* ∧ (*f*;*g*)) ∨ (¬(*init w*) ∧ *empty*)) ∧ *finite*)
by (*auto simp*: *ifthenelse-d-def*)
have *2*: $\vdash$ (((*init w* ∧ (*f*;*g*)) ∨ (¬(*init w*) ∧ *empty*)) ∧ *finite*) =
(((*init w* ∧ (*f*;*g*) ∧ *finite*) ∨ (¬(*init w*) ∧ *empty* ∧ *finite*)))
by *auto*
have *3*: $\vdash$ (*init w* ∧ (*f*;*g*) ∧ *finite*) = (*init w* ∧ (*f* ∧ *finite*);(*g* ∧ *finite*))
using *ChopAndFiniteDist* **by** *fastforce*
have *4*: $\vdash$ (¬(*init w*) ∧ *empty* ∧ *finite*) = (¬(*init w*) ∧ *empty*)
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have *5*: $\vdash$ (((*init w* ∧ (*f*;*g*) ∧ *finite*) ∨ (¬(*init w*) ∧ *empty* ∧ *finite*))) =
((*init w* ∧ (*f* ∧ *finite*);(*g* ∧ *finite*)) ∨ (¬(*init w*) ∧ *empty*))

by (*metis 2 3 4 inteq-reflection*)
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 *WhileEqvIf*:
$\vdash$ ((*while* (*init w*) *do* *f*) $\wedge$ *finite*) =
(*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*) =
((((*init w*) $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*))) $\wedge$ *finite*)
by (*simp add*: *while-d-def*)
have *2*: $\vdash$ (*init w* $\wedge$ *f*)* = (*empty* $\vee$ ((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*))
by (*metis CSEqvOrChopCSB ChopplusCommute int-eq*)
have *21*: $\vdash$ (((*init w*) $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
((*empty* $\vee$ ((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*)) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)
using *2* **by** *fastforce*
have *22*: $\vdash$ ((*empty* $\vee$ ((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*)) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
((*empty* $\wedge$ *fin* ($\neg$(*init w*)) $\wedge$ *finite*) $\vee$
(((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*))
by *auto*
have *23*: $\vdash$ ((((*init w*) $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*))) $\wedge$ *finite*) =
((*empty* $\wedge$ *fin* ($\neg$(*init w*)) $\wedge$ *finite*) $\vee$
(((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*) $\wedge$ *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*) $\vee$
(((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)) =
(($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*))
using *23 33* **by** *auto*
have *4*: $\vdash$ (*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)* = (*init w* $\wedge$ (*f*; (*init w* $\wedge$ *f*)*))
by (*rule StateAndChop*)
have *41*: $\vdash$ (((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(*init w* $\wedge$ (*f*; (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)
using *4* **by** *auto*
have *42*: $\vdash$ (*init w* $\wedge$ ((*f*; (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)) =
(*init w* $\wedge$ ((*f*; (*init w* $\wedge$ *f*)*) $\wedge$ *fin* (*init* ($\neg$ *w*)) $\wedge$ *finite*))
using *Initprop*(*2*) **by** (*metis StateAndEmptyChop int-eq*)
have *5*: $\vdash$ ((*f*; ((*init w* $\wedge$ *f*)*)) $\wedge$ (*fin* (*init* ($\neg$ *w*))) $\wedge$ *finite*)
= ((*f* $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ (*fin* (*init* ($\neg$ *w*))) $\wedge$ *finite*))

using *ChopAndFin* **by** *fastforce*
hence *49*: $\vdash$ (*init w* $\wedge$ (*f*; (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(*init w* $\wedge$ (*f* $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ (*fin* (*init* ($\neg$ *w*))) $\wedge$ *finite*))
using *42* **by** *fastforce*
have *50*: $\vdash$ (((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(*init w* $\wedge$ (*f* $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ (*fin* (*init* ($\neg$ *w*))) $\wedge$ *finite*))
by (*meson 41 49 Prop04 lift-and-com*)
have *51*: $\vdash$ (*init w* $\wedge$ (*f* $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ (*fin* (*init* ($\neg$ *w*))) $\wedge$ *finite*)) =
(*init w* $\wedge$ (*f* $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ (*fin* ($\neg$ (*init w*))) $\wedge$ *finite*))
by (*metis* (*no-types*) *EmptyChop Initprop*(*2*) *inteq-reflection*)
have *52*: $\vdash$ (((*init w* $\wedge$ *f*); (*init w* $\wedge$ *f*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
(*init w* $\wedge$ ((*f* $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *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*)*) $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)) =
(($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*))))
using *52 34* **by** *auto*
have *6*: $\vdash$ ((*f* $\wedge$ *finite*); (((*init w* $\wedge$ *f*)* $\wedge$ *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*)* $\wedge$ *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*)* $\wedge$ *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*)))
proof −
have $\vdash$(*while* (*init w*) *do f* $\wedge$ *finite*) =
($\neg$ *init w* $\wedge$ *empty* $\vee$ (*init w* $\wedge$ *f*);(*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ *init w*) $\wedge$ *finite*)
by (*metis 1 23 34 inteq-reflection*)
then show *?thesis*
by (*metis 21 22 34 52 ChopAndFiniteDist int-eq*)
qed
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 *WPower-import-finite*:
$\vdash$ (*wpower f k* $\wedge$ *finite*) = *wpower* (*f* $\wedge$ *finite*) *k*
proof (*induct k*)
case *0*
then show *?case*

using *FiniteAndEmptyEqvEmpty* **by** *fastforce*
next
case (*Suc k*)
then show *?case*
by (*metis ChopAndFiniteDist inteq-reflection wpow-Suc*)
qed

lemma *WPowerstar-import-finite*:
$\vdash$ (*wpowerstar f* $\wedge$ *finite*) = (*wpowerstar* (*f* $\wedge$ *finite*))
proof –
 have *1*: $\vdash$ (*wpowerstar* (*f* $\wedge$ *finite*)) $\longrightarrow$ (*wpowerstar f* $\wedge$ *finite*)
 by (*metis OrFiniteInf Prop12 SChopstar-finite SChopstar-WPowerstar WPowerstar-subdist inteq-reflection*)
 have *2*: $\vdash$ *wpowerstar f* $\wedge$ *finite* $\longrightarrow$ (*wpowerstar* (*f* $\wedge$ *finite*))
 unfolding *wpowerstar-d-def* **using** *WPower-import-finite*[*of f*] **by** *fastforce*
 show *?thesis*
 using *1 2 int-iffI* **by** *blast*
qed

lemma *Chopstar-import-finite*:
$\vdash$ (*chopstar f* $\wedge$ *finite*) = (*chopstar* (*f* $\wedge$ *finite*))
using *Chopstar-WPowerstar*[*of f*] *Chopstar-WPowerstar*[*of LIFT* (*f* $\wedge$ *finite*)]
 by (*metis WPowerstar-import-finite WPowerstar-more-absorb int-eq*)

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 *WPowerstar-chopstar*:
$\vdash$ (*wpowerstar* (*f* $\wedge$ *more*)) = (*chopstar f*)
by (*meson Chopstar-WPowerstar Prop11*)

lemma *While-import-finite*:
$\vdash$ (*while g do f* $\wedge$ *finite*) = (*while g do* (*f* $\wedge$ *finite*))
proof –
 have *1*: $\vdash$(*g* $\wedge$ *f* $\wedge$ *finite*) = ((*g* $\wedge$ *f*) $\wedge$ *finite*)
 by *auto*
 have *2*: $\vdash$ *chopstar* (*g* $\wedge$ *f* $\wedge$ *finite*) = *chopstar* ((*g* $\wedge$ *f*) $\wedge$ *finite*)
 by (*metis 1 Chopstardef int-eq*)
 show *?thesis*
 unfolding *while-d-def*
 using *Chopstar-import-finite*[*of LIFT* (*g* $\wedge$ *f*)] *2* **by** *fastforce*
qed

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*: ⊢ ((*init w* ∧ *f*) ∧ *finite*) = (*init w* ∧ *f* ∧ *finite*)
by *auto*
have *2*: ⊢ *chopstar* ((*init w* ∧ *f*) ∧ *finite*) = *chopstar* (*init w* ∧ *f* ∧ *finite*)
using *1* **by** (*metis Chopstardef inteq-reflection*)
have *3*: ⊢ ((*while* (*init w*) *do* *f*) ∧ *finite*) = *while* (*init w*) *do* (*f* ∧ *finite*)
unfolding *while-d-def*
using *Chopstar-import-finite*[*of LIFT* (*init w* ∧ *f*)] *2* **by** *fastforce*
have *4*: ⊢ *while* (*init w*) *do* (*f* ∧ *finite*) = *swhile* (*init w*) *do* *f*
by (*metis Prop04 SWhile-While lift-and-com swhile-d-def*)
have *5*: ⊢ ((*init w* ∧ (*f* ∧ *finite*);*while* (*init w*) *do* (*f* ∧ *finite*) ∨ ¬ *init w* ∧ *empty*) ∧ *finite*) =
((*init w* ∧ (*f* ∧ *finite*);*while* (*init w*) *do* (*f* ∧ *finite*) ∧ *finite* ∨ ¬ *init w* ∧ *empty* ∧ *finite*))
by *fastforce*
have *6*: ⊢ ((*f* ∧ *finite*);*while* (*init w*) *do* (*f* ∧ *finite*) ∧ *finite*) =
(*f* ∧ *finite*);*while* (*init w*) *do* (*f* ∧ *finite*)
by (*metis 3 ChopAndFiniteDist Prop10 Prop12 int-eq int-iffD2*)
have *7*: ⊢ (¬ *init w* ∧ *empty* ∧ *finite*) = (¬ *init w* ∧ *empty*)
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have *8*: ⊢ (if_i (*init w*) *then* ((*f* ∧ *finite*); (*while* (*init w*) *do* (*f* ∧ *finite*))) *else* *empty* ∧ *finite*) =
(if_i (*init w*) *then* (*f*⌢ (*swhile* (*init w*) *do* *f*)) *else* *empty*)
unfolding *ifthenelse-d-def schop-d-def*
by (*metis 4 5 6 7 inteq-reflection*)
have *9*: ⊢ ((*while* (*init w*) *do* (*f* ∧ *finite*)) ∧ *finite*) =
(if_i (*init w*) *then* ((*f* ∧ *finite*); (*while* (*init w*) *do* (*f* ∧ *finite*))) *else* *empty* ∧ *finite*)
by (*simp add*: *WhileEqvIf*)
show *?thesis*
by (*metis 3 4 8 9 Prop10 Prop12 int-iffD2 inteq-reflection*)
qed

lemma *WhileChopEqvIf*:
⊢ ((*while* (*init w*) *do f*) ∧ *finite*); *g* =
if_i (*init w*) *then* ((*f* ∧ *finite*); (((*while* (*init w*) *do f*) ∧ *finite*) ; *g*)) *else* *g*
proof −
have *1*: ⊢ (*while* (*init w*) *do* *f* ∧ *finite*)=
(if_i (*init w*) *then* (*f*; (*while* (*init w*) *do* *f*)) *else* *empty* ∧ *finite*)
by (*rule WhileEqvIf*)
have *11*: ⊢ (if_i (*init w*) *then* (*f*; (*while* (*init w*) *do* *f*)) *else* *empty* ∧ *finite*) =
(if_i (*init w*) *then* ((*f* ∧ *finite*); (*while* (*init w*) *do* *f* ∧ *finite*)) *else* *empty*)
using *IfAndFiniteDist* **by** *fastforce*
have *12*: ⊢ (*while* (*init w*) *do* *f* ∧ *finite*) =
(if_i (*init w*) *then* ((*f* ∧ *finite*); (*while* (*init w*) *do* *f* ∧ *finite*)) *else* *empty*)
using *1 11* **by** *fastforce*
hence *2*: ⊢ ((*while* (*init w*) *do* *f*) ∧ *finite*); *g* =
(if_i (*init w*)
then (((*f* ∧ *finite*); (*while* (*init w*) *do* *f* ∧ *finite*));*g*)
else (*empty*;*g*))
by (*rule IfChopEqvRule*)
have *3*: ⊢ *empty* ; *g* = *g*

by (*rule EmptyChop*)
have *4*: $\vdash$ (*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 (*meson ChopAssoc Prop11*)
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*
show *?thesis*
using *2 4 6* **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*
unfolding *schop-d-def*
by (*metis* (*no-types*, *opaque-lifting*) *ChopEmpty EmptySChop SChopAssoc SWhile-While WhileChopEqvIf inteq-reflection schop-d-def*)

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 *SWhileSChopEqvIfRule*:
assumes $\vdash$ *f* = (*swhile* (*init w*) *do* *g*)$\frown$ *h*

shows $\vdash f = \ if_i \ (init\ w) \ then \ (g \frown f) \ else \ h$
using *assms*
by (*metis SWhileSChopEqvIf inteq-reflection*)

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 *SWhileImpFin*:
$\vdash \ \ swhile \ (init\ w) \ do \ f \longrightarrow \ sfin\ (\neg\ (\ init\ w))$
by (*simp add*: *Prop01 Prop05 swhile-d-def*)

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 *40*: $\vdash fin\ (\neg\ (\ init\ w)) = fin\ (\neg\ w)$
by (*metis FinEqvTrueChopAndEmpty InitAndEmptyEqvAndEmpty Initprop*(*2*) *inteq-reflection*)
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 *60*: $\vdash ((\ empty \ \wedge \ fin \ (\neg\ (init\ w)) \wedge finite) \vee$
$((init\ w \wedge f \wedge \ more\ );\ (init\ w \wedge f)^{\star} \wedge fin\ (\neg\ (\ init\ w)) \wedge finite)) =$
$((\neg\ (\ init\ w) \wedge \ empty) \vee$
$((init\ w \wedge f \wedge \ more\ );\ (init\ w \wedge f)^{\star} \wedge fin\ (\neg\ (\ init\ w)) \wedge finite))$
using *6* **by** *fastforce*
have *61*: $\vdash \ ((init\ w \wedge f)^{\star} \wedge \ fin\ (\neg\ (\ init\ w)) \wedge finite) \ =$
$((\neg\ (\ init\ w) \wedge \ empty) \vee$
$((init\ w \wedge f \wedge \ more\ );\ (init\ w \wedge f)^{\star} \wedge fin\ (\neg\ (\ init\ w)) \wedge finite))$
by (*metis 5 60 inteq-reflection*)

have *70*: $\vdash$ (*init w* $\wedge$ *f* $\wedge$ *more*); (*init w* $\wedge$ *f*)* = (*init w* $\wedge$ (*f* $\wedge$ *more*); (*init w* $\wedge$ *f*)*)
  by (*rule StateAndChop*)
have *7*: $\vdash$ ((*init w* $\wedge$ *f* $\wedge$ *more*); (*init w* $\wedge$ *f*)* $\wedge$ *fin* (*init* ($\neg$ *w*)) $\wedge$ *finite*) =
  (*init w* $\wedge$ (*f* $\wedge$ *more*); (*init w* $\wedge$ *f*)* $\wedge$ *fin* (*init* ($\neg$ *w*)) $\wedge$ *finite*)
  using *70* **by** *auto*
have *8*: $\vdash$ (((*f* $\wedge$ *more*); (*init w* $\wedge$ *f*)*) $\wedge$ *fin* (*init* ($\neg$ *w*)) $\wedge$ *finite*) =
  (((*f* $\wedge$ *more*) $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *fin* (*init* ($\neg$ *w*)) $\wedge$ *finite*))
  using *ChopAndFin* **by** *fastforce*
have *81*: $\vdash$ *fin* (*init* ($\neg$ *w*)) = *fin* ($\neg$ (*init w*))
  by (*meson FinEqvFin Initprop*(*2*) *Prop11*)
have *82*: $\vdash$ ((*f* $\wedge$ *more*); (*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
  (((*f* $\wedge$ *more*) $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *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*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
  (*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *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*)* $\wedge$ *fin* ($\neg$(*init* (*w*))) $\wedge$ *finite*)) =
  (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
  (*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)))
    using *82* **by** *auto*
have *9*: $\vdash$ ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
  (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
    (*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);
      ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*)))
    by (*metis 61 7 81 84 inteq-reflection*)
have *10*: $\vdash$ ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
  (((*init w* $\wedge$ *f*)* $\wedge$ *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*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*))) =
  (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
    (*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);
      (((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*))) $\wedge$ *finite*)))
 by (*metis EmptyChop int-eq*)
have *12*: $\vdash$ ((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*)) $\wedge$ *finite*) =
  (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
    (*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);
      (((*init w* $\wedge$ *f*)* $\wedge$ *fin* ($\neg$ (*init w*))) $\wedge$ *finite*)))
  by (*metis 11 9 inteq-reflection*)
from *12* **show** *?thesis*
by (*metis 10 inteq-reflection while-d-def*)
qed

lemma *SWhileEqvEmptyOrSChopSWhile*:
$\vdash$ *swhile* (*init w*) *do f* = (($\neg$ (*init w*) $\wedge$ *empty*) $\vee$ (*init w* $\wedge$ (*f* $\wedge$ *more*)$\frown$*swhile* (*init w*) *do f*))
proof –
have *1*: $\vdash$ (((*f* $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *finite*) = ((*f* $\wedge$ *more*) $\wedge$ *finite*)
  by *auto*

show *?thesis*
unfolding *schop-d-def*
using *WhileEqvEmptyOrChopWhile*[*of w LIFT* (*f*)] *SWhile-While*[*of LIFT* (*init w*) *f*]
by (*metis While-import-finite inteq-reflection*)
qed

lemma *WhileIntro*:
assumes $\vdash$ $\neg$ (*init w*) $\wedge$ *f* $\longrightarrow$ *empty*
$\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); *f*
shows $\vdash$ *f* $\wedge$ *finite* $\longrightarrow$ *while* (*init w*) *do g*
proof –
have *1*: $\vdash$ $\neg$ (*init w*) $\wedge$ *f* $\longrightarrow$ *empty*
using *assms* **by** *blast*
have *2*: $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); *f*
using *assms* **by** *blast*
have *3*: $\vdash$ (*while* (*init w*) *do g* $\wedge$ *finite*)=
(($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))
by (*rule WhileEqvEmptyOrChopWhile*)
hence *31*: $\vdash$ $\neg$ (*while* (*init w*) *do g* $\wedge$ *finite*) =
($\neg$(($\neg$ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))))
by *fastforce*
hence *32*: $\vdash$ (*f* $\wedge$ $\neg$ (*while* (*init w*) *do g* $\wedge$ *finite*)) =
(*f* $\wedge$ $\neg$(($\neg$(*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))))
by *fastforce*
have *33*: $\vdash$ (*f* $\wedge$ $\neg$(($\neg$(*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))) =
(*f* $\wedge$ $\neg$($\neg$(*init w*) $\wedge$ *empty*) $\wedge$
$\neg$(*init w* $\wedge$ ((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))
by *auto*
have *34*: $\vdash$ (*f* $\wedge$ $\neg$($\neg$(*init w*) $\wedge$ *empty*) $\wedge$
$\neg$((*init w*) $\wedge$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*)))) =
(*f* $\wedge$ ((*init w*) $\vee$ *more*) $\wedge$
($\neg$(*init w*) $\vee$ $\neg$(((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))))
by (*auto simp*: *empty-d-def*)
have *35*: $\vdash$ (*f* $\wedge$ ((*init w*) $\vee$ *more*) $\wedge$
($\neg$(*init w*) $\vee$ $\neg$(((*g* $\wedge$ *more*) $\wedge$ *finite*);(*while* (*init w*) *do g* $\wedge$ *finite*)))) =
((*f* $\wedge$ (*init w*) $\wedge$ $\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ (*init w*) $\wedge$ $\neg$(*init w*)) $\vee$
(*f* $\wedge$ *more* $\wedge$ $\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ *more* $\wedge$ $\neg$(*init w*)))
by *auto*
have *36*: $\vdash$ (*f* $\wedge$ $\neg$ (*while* (*init w*) *do g* $\wedge$ *finite*)) =
((*f* $\wedge$ (*init w*) $\wedge$ $\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ (*init w*) $\wedge$ $\neg$(*init w*)) $\vee$
(*f* $\wedge$ *more* $\wedge$ $\neg$ (((*g* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do g* $\wedge$ *finite*))) $\vee$
(*f* $\wedge$ *more* $\wedge$ $\neg$(*init w*)))
by (*metis 32 33 34 35 int-eq*)

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*))
  by (*meson 36 40 Prop11 lift-imp-trans*)
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 *SWhileIntro*:
assumes ⊢ ¬ (*init w*) ∧ *f* ⟶ *empty*
  ⊢ *init w* ∧ *f* ⟶ (*g* ∧ *more*)⌢ *f*
shows ⊢ *f* ∧ *finite* ⟶ *swhile* (*init w*) *do* *g*
proof −
have *1*: ⊢ *f* ∧ *finite* ⟶ *while* (*init w*) *do* *g*
  using *assms*
  using *assms*
  unfolding *schop-d-def*
  using *WhileIntro*[*of w f g*]
  by *blast*
have *2*: ⊢ *f* ∧ *finite* ⟶ *while* (*init w*) *do* *g* ∧ *finite*
  using *1* **by** *auto*
show *?thesis*
using *2*
SWhile-While[*of LIFT* (*init w*) *g*]
While-import-finite[*of LIFT* (*init w*) *g*]
by *fastforce*
qed

lemma *WhileElim*:
assumes $\vdash$ $\neg$ (*init w*) $\wedge$ *empty* $\longrightarrow$ *g*
$\vdash$ *init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); *g* $\longrightarrow$ *g*
shows $\vdash$ *while* (*init w*) *do* *f* $\wedge$ *finite* $\longrightarrow$ *g*
proof –
have *1*: $\vdash$ (*while* (*init w*) *do* *f* $\wedge$ *finite*)=
((¬ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*)))
by (*rule WhileEqvEmptyOrChopWhile*)
hence *11*: $\vdash$ ((*while* (*init w*) *do* *f* $\wedge$ *finite*) $\wedge$ ¬ *g*) =
(((¬(*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);(*while* (*init w*) *do f* $\wedge$ *finite*))) $\wedge$ ¬ *g*)
by (*metis inteq-reflection lift-and-com*)
have *2*: $\vdash$ ¬ (*init w*) $\wedge$ *empty* $\longrightarrow$ *g*
using *assms* **by** *blast*
hence *21*: $\vdash$ ¬ *g* $\longrightarrow$ ¬(¬ (*init w*) $\wedge$ *empty*)
by *auto*
have *22*: $\vdash$ ((¬ (*init w*) $\wedge$ *empty*) $\vee$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*);(*while* (*init w*) *do f* $\wedge$ *finite*))) $\wedge$ ¬ *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*) $\wedge$ ¬ *g* $\longrightarrow$
(*init w* $\wedge$ ((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*)) $\wedge$ ¬ *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*: $\vdash$ ¬ *g* $\longrightarrow$ ¬((*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*)) $\wedge$ ¬ *g* $\longrightarrow$
((((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*)) $\wedge$
¬ (((*f* $\wedge$ *more*) $\wedge$ *finite*); *g*)) $\wedge$ ¬*g*
using *31* **by** *fastforce*
have *4*: $\vdash$ (*while* (*init w*) *do* *f* $\wedge$ *finite*) $\wedge$ ¬ *g* $\longrightarrow$
(((*f* $\wedge$ *more*) $\wedge$ *finite*); (*while* (*init w*) *do* *f* $\wedge$ *finite*)) $\wedge$
¬ (((*f* $\wedge$ *more*) $\wedge$ *finite*); *g*)
by (*meson 23 32 Prop12 lift-imp-trans*)
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 *SWhileElim*:
assumes $\vdash$ ¬ (*init w*) $\wedge$ *empty* $\longrightarrow$ *g*
$\vdash$ *init w* $\wedge$ (*f* $\wedge$ *more*)⁀ *g* $\longrightarrow$ *g*
shows $\vdash$ *swhile* (*init w*) *do* *f* $\longrightarrow$ *g*
using *assms*
unfolding *schop-d-def*
using *WhileElim*[*of w g f*] *SWhile-While*[*of LIFT* (*init w*) *f*]

While-import-finite[*of LIFT* (*init w*) *f*]
by *fastforce*

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*)* $\wedge$ *fin* ($\neg$ (*init w*))
$\longrightarrow$ (*init w* $\wedge$ *g*)* $\wedge$ *fin* ($\neg$ (*init w*)))
using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *while-d-def*)
qed

lemma *SBaSWhileImpSWhile*:
$\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*swhile* (*init w*) *do* *f*) $\longrightarrow$ (*swhile* (*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*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ *sba* ((*init w* $\wedge$ *f*) $\longrightarrow$ (*init w* $\wedge$ *g*))
by (*rule SBaImpSBa*)
have *3*: $\vdash$ *sba* ((*init w* $\wedge$ *f*) $\longrightarrow$ (*init w* $\wedge$ *g*)) $\longrightarrow$
(*schopstar* (*init w* $\wedge$ *f*) $\longrightarrow$ *schopstar* (*init w* $\wedge$ *g*))
by (*rule SBaSCSImpSCS*)
have *4*: $\vdash$ *sba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*schopstar* (*init w* $\wedge$ *f*) $\wedge$ *sfin* ($\neg$ (*init w*)) $\longrightarrow$
schopstar (*init w* $\wedge$ *g*) $\wedge$ *sfin* ($\neg$ (*init w*)))
using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *swhile-d-def*)
qed

lemma *WhileImpWhile*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ (*while* (*init w*) *do* *f*) $\wedge$ *finite* $\longrightarrow$ (*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*) $\wedge$ *finite* $\longrightarrow$ (*while* (*init w*) *do* *f*) $\longrightarrow$ (*while* (*init w*) *do* *g*)
by (*rule BaWhileImpWhile*)
have *4*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*while* (*init w*) *do* *f* $\wedge$ *finite*) $\longrightarrow$ (*while* (*init w*) *do* *g*)
using *3* **by** (*auto simp*: *Valid-def*)
from *2 4* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *SWhileImpSWhile*:
assumes ⊢ $f \longrightarrow g$
shows ⊢ (*swhile* (*init w*) *do* *f*) $\longrightarrow$ (*swhile* (*init w*) *do* *g*)
proof −
have *1*: ⊢ $f \longrightarrow g$
using *assms* **by** *auto*
hence *2*: ⊢ *sba* ($f \longrightarrow g$)
by (*rule SBaGen*)
have *3*: ⊢ *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

end

theory *Omega*
imports *Chopstar*
begin

This theory defines the omega operator for infinite ITL and provides a library of lemmas. We also define a weak version womega that corresponds to the omega operator from Omega Algebra [1]. We also provide a semantic version aomega and provide lemmas that express various relationships between them. We also ported the numerous omega algebra lemmas from [1] to ITL.

9 Omega and variants

9.1 Definitions

9.1.1 Omega

lemma *FMoreSem-var* [*mono*]:
(*w* ⊨ ((*f* ∧ *more*) ∧ *finite*);*g*) =
((*0*< *nlength w* ∧ (∃ *n*. *f* ((*ntaken n w*)) ∧ *0*<*n* ∧ *g* (*ndropn n w*))))
by (*simp add*: *itl-defs*)
(*metis enat-0-iff*(*1*) *linorder-not-less ndropn-all neq0-conv*
nfinite-nlength-enat nfinite-ntaken ntaken-all order-le-less)

coinductive *omega-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
for *F* **where**
(*s* ⊨ ((*F* ∧ *more*) ∧ *finite*);(*omega-d F*)) $\Longrightarrow$ (*s* ⊨ (*omega-d F*))

syntax
-omega-d :: *lift* ⇒ *lift* ((-$^\omega$) [*85*] *85*)

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

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

lemma *OmegaIntros*:
$\vdash$ $((f \wedge more) \wedge finite);(omega\ f) \longrightarrow (omega\ f)$
using *omega-d.intros* **using** *unl-lift2* **by** *blast*

lemma *OmegaCases*:
$(w \models (omega\ F)) \Longrightarrow$
$(w \models \ ((F \wedge more) \wedge finite);(omega\ F) \Longrightarrow P) \Longrightarrow P$
using *omega-d.cases*[*of F w P*] **by** *auto*

lemma *OmegaUnrollSem*:
$(s\models (omega\ f)) = (s \models ((f \wedge more) \wedge finite);(omega\ f))$
using *omega-d.cases*[*of f*]
by (*metis omega-d.simps*)

lemma *OmegaCoinductSem*:
assumes $\bigwedge x.\ \ X\ x \Longrightarrow x \models\ ((F \wedge more) \wedge finite);(X\ \vee omega\ F)$
shows $x\ \models X\ \ \longrightarrow omega\ F$
using *assms omega-d.coinduct*[*of X x F*]
by (*auto simp add*: *chop-defs*)

lemma *OmegaWeakCoinductSem*:
assumes $\bigwedge x.\ \ X\ x \Longrightarrow x \models\ ((F \wedge more) \wedge finite);X$
shows $x\ \models X\ \ \longrightarrow omega\ F$
using *assms omega-d.coinduct*[*of X x F*]
by (*auto simp add*: *chop-defs*)

lemma *OmegaUnroll*:
$\vdash f^{\omega} = ((f \wedge more) \wedge finite);f^{\omega}$
using *OmegaUnrollSem unl-lift2* **by** *blast*

lemma *OmegaCoinduct*:
assumes $\vdash X \longrightarrow\ ((F \wedge more) \wedge finite);(X \vee (omega\ F))$
shows $\vdash X\ \longrightarrow (omega\ F)$
using *assms OmegaCoinductSem*[*of X F*]
by (*simp add*: *Valid-def*)

lemma *OmegaWeakCoinduct*:
assumes $\vdash X \longrightarrow\ ((F \wedge more) \wedge finite);X$
shows $\vdash X\ \longrightarrow (omega\ F)$
using *assms OmegaWeakCoinductSem*[*of X F*]
by (*simp add*: *Valid-def*)

9.1.2 Alternative definition for Omega

lemma *infinite-nidx-imp-infinite-interval*:
assumes $\neg$ *nfinite l*

nidx l
(nnth l 0) = 0
∀ i. (nnth l i) ≤ nlength s
shows ¬ *nfinite s*
proof
assume *nfinite s*
thus *False*
using *assms*
proof (*induct s rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis dual-order.antisym enat-ord-simps*(*1*) *linorder-linear nfinite-ntaken nidx-gr-first nlength-NNil not-gr-zero ntaken-all zero-le zero-less-Suc*)
next
case (*NCons x nell*)
then show *?case*
proof −
have *1*: ⋀*j*. (*nnth l j*) < (*nnth l* (*Suc j*))
by (*metis assms*(*1*) *assms*(*2*) *linorder-le-cases nfinite-ntaken nidx-expand ntaken-all*)
have *2*: ∀ *i*. *enat* (*nnth l i*) ≤ *nlength* (*NCons x nell*) ⟹ *False*
by (*metis 1 NCons.hyps*(*2*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *dual-order.strict-iff-order enat-ord-simps*(*1*) *iless-Suc-eq linorder-not-le nlength-NCons*)
show *?thesis*
using *2 NCons.prems*(*4*) **by** *auto*
qed
qed
qed

definition *aomega-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
where *aomega-d F* ≡
(*λs*.
(∃ (*l*:: *nat nellist*).
¬*nfinite l* ∧ (*nnth l 0*) = *0* ∧ *nidx l* ∧
(∀ *i*. (*nnth l i*) ≤ *nlength s*) ∧
(∀ *i*. ((*nsubn s* (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *F*))
)
)

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

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

translations
-aomega-d ⇌ *CONST aomega-d*

lemma *aomega-unroll-chain-a*:

assumes $(\exists\, l.\ \neg\ nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
$(\forall\, i.\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ f\ (\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))))$
shows $(\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (\ (ntaken\ n\ \sigma)) \wedge$
$0 < n \wedge$
$enat\ 0 < nlength\ \sigma \wedge$
$(\exists\, l.$
$\neg\ nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l\ \wedge$
$(\forall\, i.\ (nnth\ l\ i) \leq nlength\ \sigma - enat\ n) \wedge$
$(\forall\, i.\ f\ (\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
)
)
proof –
obtain *l* **where** *1*: $\neg nfinite\ l \wedge\ (nnth\ l\ 0) = 0 \wedge nidx\ \ l \wedge$
$(\forall\, i.\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge (\forall\, i.\ f\ (\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
using *assms* **by** *auto*
have *2*: $l = NCons\ (nnth\ l\ 0)\ (ndropn\ 1\ l)$
by (*metis 1 One-nat-def gr-zeroI ndropn-0 ndropn-Suc-conv-ndropn nlength-eq-enat-nfiniteD zero-enat-def*)
have *3*: $(nnth\ l\ 0) = 0$
using *1* **by** *blast*
have *4*: $(nnth\ l\ 0) < (nnth\ l\ 1)$
using *1*
by (*metis One-nat-def Suc-ile-eq enat-defs*(*1*) *nidx-gr-first nlength-eq-enat-nfiniteD not-gr-zero zero-less-one*)
have *5*: $nidx\ (ndropn\ 1\ l)$
using *1 nidx-expand*[*of l*] *nidx-expand*[*of* (*ndropn 1 l*)]
by (*metis dual-order.order-iff-strict enat-defs*(*1*) *ndropn-all ndropn-nnth nfinite-ndropn-b nlength-NNil nlength-eq-enat-nfiniteD not-le-imp-less plus-1-eq-Suc*)
have *6*: $f\ (\ (nsubn\ \sigma\ (nnth\ l\ 0)\ (nnth\ l\ 1)))$
by (*metis 1 One-nat-def*)
have *7*: $(\forall\, i.\ f\ (\ (nsubn\ \sigma\ \ (nnth\ (ndropn\ 1\ l)\ i)\ (nnth\ (ndropn\ 1\ l)\ (Suc\ i)))))$
by (*simp add*: *1*)
have *8*: $f\ (\ (ntaken\ (nnth\ l\ 1)\ \sigma))$
by (*metis 1 4 One-nat-def Suc-diff-1 Suc-diff-Suc ndropn-0 nsubn-def1*)
have *81*: $\neg\ nfinite\ (nmap\ (\lambda\ e.\ e - (nnth\ l\ 1))\ (ndropn\ 1\ l))$
by (*simp add*: *1*)
have *82*: $\bigwedge j.\ 0 < j \longrightarrow (nnth\ l\ 1) < nnth\ (ndropn\ 1\ l)\ j$
by (*metis 1 Suc-diff-1 enat-defs*(*1*) *linorder-le-cases ndropn-all ndropn-nnth nfinite-ndropn-b nidx-less nlength-NNil nlength-eq-enat-nfiniteD plus-1-eq-Suc*)
have *83*: $\bigwedge j.\ nnth\ (nmap\ (\lambda e.\ e - nnth\ l\ 1)\ (ndropn\ 1\ l))\ j =$
$(nnth\ l\ (Suc\ j)) - (nnth\ l\ 1)$
by (*metis 81 le-cases ndropn-nnth nfinite-ntaken nlength-nmap nnth-nmap ntaken-all plus-1-eq-Suc*)
have *84*: $\bigwedge j.\ nnth\ (nmap\ (\lambda e.\ e - nnth\ l\ 1)\ (ndropn\ 1\ l))\ (Suc\ j) =$
$(nnth\ l\ (Suc\ (Suc\ j))) - (nnth\ l\ 1)$
using *83* **by** *blast*
have *840*: $\bigwedge j.\ (nnth\ l\ 1) \leq (nnth\ l\ (Suc\ j))$
by (*metis 1 diff-add-zero diff-is-0-eq linorder-le-cases nfinite-ntaken nidx-less-eq ntaken-all plus-1-eq-Suc*)

have *85*: $\bigwedge j.$ (*nnth l* (*Suc j*)) − (*nnth l 1*) < (*nnth l* (*Suc* (*Suc j*))) − (*nnth l 1*)
by (*metis 1 840 diff-less-mono linorder-le-cases nfinite-ntaken nidx-expand ntaken-all*)
have *86*: (∀ *i*. *enat* (*Suc i*) ≤ *nlength* (*nmap* (λ*e*. *e* − *nnth l 1*) (*ndropn 1 l*)) ⟶
nnth (*nmap* (λ*e*. *e* − *nnth l 1*) (*ndropn 1 l*)) *i* <
nnth (*nmap* (λ*e*. *e* − *nnth l 1*) (*ndropn 1 l*)) (*Suc i*))
using *83 85* **by** *presburger*
have *9*: *nidx* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*))
using *nidx-expand*[*of* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*))]
using *83 85* **by** *presburger*
have *91*: $\bigwedge j.$ (*nnth* (*nmap* (λ*e*. *e* + (*nnth l 1*)) (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*))) *j*) =
(*nnth l* (*Suc j*))
by (*metis 81 82 83 One-nat-def add-Suc diff-Suc-1 diff-Suc-eq-diff-pred diff-add diff-is-0-eq le-refl linorder-le-cases ndropn-nnth nfinite-ntaken nlength-nmap nnth-nmap not-le-imp-less ntaken-all order-less-imp-le plus-1-eq-Suc*)
have *10*: (∀ *i*. *f* ((*nsubn σ*
(*nnth* (*nmap* (λ*e*. *e* + (*nnth l 1*)) (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*))) *i*)
(*nnth* (*nmap* (λ*e*. *e* + (*nnth l 1*)) (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*))) (*Suc i*)))))
using *1 91* **by** *presburger*
have *11*: (∀ *i*. *f* ((*nsubn σ*
((*nnth* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) *i*) + (*nnth l 1*))
((*nnth* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) (*Suc i*)) + (*nnth l 1*))
)))
using *7 83 840* **by** *fastforce*
have *12*: (∀ *i*. *f* ((*nsubn* (*ndropn* (*nnth l 1*) *σ*)
((*nnth* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) *i*))
((*nnth* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) (*Suc i*)))
)))
by (*metis 11 83 85 nsubn-ndropn*)
have *121*: *nnth* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) *0* = *0*
using *83 One-nat-def* **by** *presburger*
have *122*: ¬ *nfinite* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) ∧
nnth (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) *0* = *0* ∧
nidx (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) ∧
(∀ *i*. *f* ((*nsubn* (*ndropn* (*nnth l 1*) *σ*)
((*nnth* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) *i*))
((*nnth* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) (*Suc i*)))
)))
using *12 121 81 9* **by** *blast*
have *123*: (∀ *i*. (*nnth* (*nmap* (λ *e*. *e* − (*nnth l 1*)) (*ndropn 1 l*)) *i*) ≤ *nlength* (*ndropn* (*nnth l 1*) *σ*))
by (*meson 1 enat-ile infinite-nidx-imp-infinite-interval le-cases nfinite-ndropn-b nlength-eq-enat-nfiniteD*)
have *13*: (∃ *ls*.
¬ *nfinite ls* ∧ (*nnth ls 0*) = *0* ∧ *nidx ls* ∧
(∀ *i*. (*nnth ls i*) ≤ *nlength* (*ndropn* (*nnth l 1*) *σ*)) ∧
(∀ *i*. *f* ((*nsubn* (*ndropn* (*nnth l 1*) *σ*) (*nnth ls i*) (*nnth ls* (*Suc i*)))))
)
using *122 123* **by** *blast*
from *13* **show** *?thesis* **using** *4 8*
by (*metis 1 enat-ord-simps*(*1*) *linorder-not-less ndropn-nlength not-gr-zero zero-enat-def*)
qed

lemma *aomega-unroll-chain-b*:
assumes $(\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$f\ (\ (ntaken\ n\ \sigma)) \wedge$
$0 < n \wedge$
$enat\ 0 < nlength\ \sigma \wedge$
$(\exists l.$
$\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
$(\forall i.\ (nnth\ l\ i) \le nlength\ \sigma - enat\ n) \wedge$
$(\forall i.\ f\ (\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
$)$
$)$
shows $(\exists l.\ \neg\ nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
$(\forall i.\ (nnth\ l\ i) \le nlength\ \sigma) \wedge$
$(\forall i.\ f\ (\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))))$
proof −
obtain n **where** *1*: $enat\ n \le nlength\ \sigma \wedge f\ (\ (ntaken\ n\ \sigma)) \wedge$
$0 < n \wedge\ \ enat\ 0 < nlength\ \sigma \wedge$
$(\exists l.$
$\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
$(\forall i.\ (nnth\ l\ i) \le nlength\ \sigma - enat\ n) \wedge$
$(\forall i.\ f\ (\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
$)$
using *assms* **by** *auto*
have *2*: $(\exists l.$
$\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
$(\forall i.\ (nnth\ l\ i) \le nlength\ (ndropn\ n\ \sigma)) \wedge$
$(\forall i.\ f\ (\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
$)$
using *1* **by** *auto*
obtain l **where** *3*: $\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
$(\forall i.\ (nnth\ l\ i) \le nlength\ (ndropn\ n\ \sigma)) \wedge$
$(\forall i.\ f\ (\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
using *2* **by** *auto*
have *4*: $nidx\ (nmap\ (\lambda e.\ e + n)\ l)$
using *3*
by (*simp add*: *Suc-ile-eq nidx-expand*)
have *42*: $\bigwedge j.\ \ nnth\ (NCons\ 0\ (nmap\ (\lambda e.\ e + n)\ l))\ j =$
$(case\ j\ of\ \ 0 \Rightarrow 0\ |$
$(Suc\ k) \Rightarrow nnth\ (nmap\ (\lambda e.\ e + n)\ l)\ k\ )$
by (*simp add*: *nnth-NCons*)
have *43*: $\bigwedge j.\ nnth\ (NCons\ 0\ (nmap\ (\lambda e.\ e + n)\ l))\ (Suc\ j) =$
$nnth\ (nmap\ (\lambda e.\ e + n)\ l)\ j$
by *simp*
have *44*: $0 < nnth\ (nmap\ (\lambda e.\ e + n)\ l)\ 0$
using *1 enat-defs*(*1*) **by** *auto*
have *45*: $\bigwedge j.\ \ nnth\ (nmap\ (\lambda e.\ e + n)\ l)\ j < nnth\ (nmap\ (\lambda e.\ e + n)\ l)\ (Suc\ j)$
by (*metis 3 4 add.right-neutral le-cases nfinite-ntaken nidx-less nlength-nmap ntaken-all*)
have *46*: $(\forall i.\ enat\ (Suc\ i) \le nlength\ (NCons\ 0\ (nmap\ (\lambda e.\ e + n)\ l)) \longrightarrow$

nnth (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) *i* <
nnth (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) (*Suc i*))
by (*metis 43 44 45 lessI less-Suc-eq-0-disj nnth-0*)
have *5*: *nidx* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*))
using *46 nidx-expand* **by** *blast*
have *6*: ((*ntaken n σ*)) =
((*nsubn σ* (*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) *0*)
(*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) (*Suc 0*))))
by (*metis 3 43 44 One-nat-def Suc-diff-1 Suc-diff-Suc add-0 ndropn-0 nnth-0 nnth-nmap nsubn-def1 zero-enat-def zero-le*)
have *7*: (∀ *i*. *f* ((*nsubn* (*ndropn n σ*) (*nnth l i*) (*nnth l* (*Suc i*)))))
using *3* **by** *auto*
have *8*: (∀ *i*. *f* ((*nsubn σ* ((*nnth l i*)+*n*) ((*nnth l* (*Suc i*)) +*n*))))
using *7*
by (*simp add*: *add.commute ndropn-ndropn nsubn-def1*)
have *9*: (∀ *i*. *f* ((*nsubn σ* ((*nnth* (*nmap* (*λe. e* + *n*) *l*) *i*))
((*nnth* (*nmap* (*λe. e* + *n*) *l*) (*Suc i*))))))
using *8*
by (*metis 45 Suc-ile-eq dual-order.order-iff-strict linorder-le-cases nat-neq-iff nlength-nmap nnth-beyond nnth-nmap*)
have *10*: *f* ((*nsubn σ* (*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) *0*)
(*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) (*Suc 0*))))
using *1 6* **by** *auto*
have *11*: (∀ *i*. *i*>*0* ⟶
f ((*nsubn σ* (*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) *i*)
(*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) (*Suc i*)))))
by (*metis 9 Suc-diff-1 nnth-Suc-NCons*)
have *12*: (∀ *i*.
f ((*nsubn σ* (*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) *i*)
(*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) (*Suc i*)))))
using *10 11 neq0-conv* **by** *blast*
have *13*: ∀ *i*. (*nnth* (*NCons 0* (*nmap* (*λe. e* + *n*) *l*)) *i*) ≤ *nlength σ*
by (*metis 3 add.commute enat-ile infinite-nidx-imp-infinite-interval linorder-le-cases ndropn-all ndropn-ndropn nfinite-ndropn-b nlength-NNil nlength-eq-enat-nfiniteD zero-le*)
show *?thesis* **using** *12 5*
by (*metis 13 3 nfinite-NCons nfinite-nmap nnth-0*)
qed

lemma *aomega-unroll-chain*:
(∃ *l*. ¬ *nfinite l* ∧ (*nnth l 0*) = *0* ∧ *nidx l* ∧
(∀ *i*. (*nnth l i*) ≤ *nlength σ*) ∧
(∀ *i*. *f* ((*nsubn σ* (*nnth l i*) (*nnth l* (*Suc i*))))))
=
(∃ *n*. *enat n* ≤ *nlength σ* ∧
f ((*ntaken n σ*)) ∧
0 < *n* ∧
enat 0 < *nlength σ* ∧
(∃ *l*.
¬ *nfinite l* ∧ (*nnth l 0*) = *0* ∧ *nidx l* ∧
(∀ *i*. (*nnth l i*) ≤ *nlength σ* − *enat n*) ∧

$(\forall\, i.\ f\ (\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
)
)

using *aomega-unroll-chain-a*[*of σ f*] *aomega-unroll-chain-b*[*of σ f*]
by *blast*

lemma *aomega-unroll-sem*:
$(\sigma \models ((f \wedge more) \wedge finite);(aomega\ f) = (aomega\ f))$
proof
(*simp add*: *itl-defs zero-enat-def aomega-d-def*)
show $((\exists\, n.\ enat\ n \leq nlength\ \sigma \wedge$
$f\ (ntaken\ n\ \sigma) \wedge$
$0 < n \wedge$
$enat\ 0 < nlength\ \sigma \wedge$
$(\exists\, l.\ \neg\ nfinite\ l \wedge nnth\ l\ 0 = 0 \wedge nidx\ l \wedge$
$(\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma - enat\ n) \wedge$
$(\forall\, i.\ f\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))) \vee$
$\neg\ nfinite\ \sigma \wedge f\ \sigma \wedge enat\ 0 < nlength\ \sigma \wedge nfinite\ \sigma) =$
$(\exists\, l.\ \neg\ nfinite\ l \wedge nnth\ l\ 0 = 0 \wedge nidx\ l \wedge (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ f\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$
using *aomega-unroll-chain*[*of σ f*] **by** *blast*
qed

lemma *AOmegaUnroll*:
$\vdash (aomega\ f) = ((f \wedge more) \wedge finite);(aomega\ f)$
unfolding *Valid-def*
using *aomega-unroll-sem* **by** *auto*

lemma *AOmegaInductSem-help*:
$(\sigma \models\ inf \wedge g \wedge \Box(g \longrightarrow ((f \wedge more) \wedge finite);g)) =$
$(\ \neg\ nfinite\ \sigma \wedge g\ \sigma \wedge$
$(\forall\, n.\ g\ (\ (ndropn\ n\ \sigma)) \longrightarrow$
$(\exists\, na.\ f\ (\ (nsubn\ \sigma\ n\ (na{+}n))) \wedge$
$0 < na \wedge g\ (\ (ndropn\ (n + na)\ \sigma))))$
)

by (*simp add*: *itl-defs zero-enat-def min-def ndropn-ndropn nsubn-def1*)
(*metis linorder-le-cases ndropn-all ndropn-nlength nfinite-NNil nfinite-ndropn-b*)

primrec *cpoint* :: $('a{::}world)\ formula \Rightarrow 'a\ formula \Rightarrow nat \Rightarrow 'a\ intervals \Rightarrow nat$
where $cpoint\ f\ g\ 0\ \sigma = 0$
$|\ cpoint\ f\ g\ (Suc\ n)\ \sigma =$
$(\epsilon\ \ x.\ (\exists\ \ m.\ (\ (nsubn\ \sigma\ (cpoint\ f\ g\ n\ \sigma)\ (m{+}(cpoint\ f\ g\ n\ \sigma))\ \ ) \models f)$
$\wedge\ m{>}0\ \wedge((ndropn\ (m{+}(cpoint\ f\ g\ n\ \sigma))\ \sigma) \models g) \wedge$
$x{=}m{+}(cpoint\ f\ g\ n\ \sigma)$
)
)

lemma *cpoint-expand-0*:
 ($cpoint\ f\ g\ 0\ \sigma$) $=0$
by *simp*

lemma *cpoint-expand-1*:
 ($cpoint\ f\ g\ 1\ \sigma$) $=$
   ($SOME\ x.$ ($\exists\ m.\ f$ (($nsubn\ \sigma\ 0$ (m)))
                 $\wedge$ $m>0$ $\wedge$ g (($ndropn$ (m) σ))
                 $\wedge$ $x=m$))
by (*simp add*: *itl-defs*)

lemma *cpoint-expand-n*:
 ($cpoint\ f\ g$ ($Suc\ n$) σ) $=$
  ($SOME\ x.$ ($\exists\ m.\ f$ (($nsubn\ \sigma$ ($cpoint\ f\ g\ n\ \sigma$) ($m+$($cpoint\ f\ g\ n\ \sigma$))))
                $\wedge$ $m>0$ $\wedge$ g (($ndropn$ ($m+$($cpoint\ f\ g\ n\ \sigma$)) σ))
                $\wedge$ $x=m+$($cpoint\ f\ g\ n\ \sigma$))
)
by (*simp add*: *itl-defs*)

lemma *cpoint-0*:
 assumes $\neg nfinite\ \sigma \wedge g\ \sigma \wedge$
            ($\forall k.\ g$ (($ndropn\ k\ \sigma$)) $\longrightarrow$
                ($\exists m.\ f$ (($nsubn\ \sigma\ k$ ($m+k$))) $\wedge$
                   $0 < m \wedge g$ (($ndropn$ ($m+k$) σ))))

 shows g (($ndropn$ ($cpoint\ f\ g\ i\ \sigma$) σ))
proof
 (*induct i*)
 case *0*
 then show *?case* **by** (*simp add*: *assms*)
 next
 case (*Suc i*)
 then show *?case*
 proof −
  have *1*: g (($ndropn$ ($cpoint\ f\ g\ i\ \sigma$) σ))
  by (*simp add*: *Suc.hyps*)
  have *2*: g (($ndropn$ ($cpoint\ f\ g\ i\ \sigma$) σ)) $\longrightarrow$
        ($\exists m.\ f$ (($nsubn\ \sigma$ ($cpoint\ f\ g\ i\ \sigma$) ($m+$($cpoint\ f\ g\ i\ \sigma$)))) $\wedge$
                 $0 < m \wedge g$ (($ndropn$ ($m+$($cpoint\ f\ g\ i\ \sigma$)) σ)))
  using *assms* **by** *blast*
  have *3*: ($\exists m.\ f$ (($nsubn\ \sigma$ ($cpoint\ f\ g\ i\ \sigma$) ($m+$($cpoint\ f\ g\ i\ \sigma$)))) $\wedge$
                 $0 < m \wedge g$ (($ndropn$ ($m+$($cpoint\ f\ g\ i\ \sigma$)) σ)))
  using *1 2* **by** *auto*
  have *4*: ($cpoint\ f\ g$ ($Suc\ i$) σ) $=$
        ($SOME\ x.$ ($\exists\ m.\ f$ (($nsubn\ \sigma$ ($cpoint\ f\ g\ i\ \sigma$) ($m+$($cpoint\ f\ g\ i\ \sigma$))))
           $\wedge$ $m>0$ $\wedge$ g (($ndropn$ ($m+$($cpoint\ f\ g\ i\ \sigma$)) σ))

$\wedge$ *x=m+(cpoint f g i σ)))*
by *simp*
have *5*: *g ((ndropn ((cpoint f g (Suc i) σ)) σ))*
using *3 4 someI-ex[of λx. (∃ m. f ((nsubn σ (cpoint f g i σ) (m+(cpoint f g i σ))))*
$\wedge$ *m>0* $\wedge$ *g ((ndropn (m+(cpoint f g i σ)) σ))*
$\wedge$ *x=m+(cpoint f g i σ))]* **by** *auto*
from *5* **show** *?thesis* **by** *auto*
qed
qed

lemma *cpoint-1*:
assumes ¬ *nfinite σ* ∧ *g σ* ∧
(∀ *k. g ((ndropn k σ))* ⟶
(∃ *m. f ((nsubn σ k (m+k)))* ∧
0 < m ∧ *g ((ndropn (m+k) σ))))*

shows (*g ((ndropn (cpoint f g i σ) σ))*
⟹ *g ((ndropn (cpoint f g (Suc i) σ) σ)))*
using *assms cpoint-0* **by** *blast*

lemma *cpoint-2*:
assumes ¬*nfinite σ* ∧ *g σ* ∧
(∀ *k. g ((ndropn k σ))* ⟶
(∃ *m. f ((nsubn σ k (m+k)))* ∧
0 < m ∧ *g ((ndropn (m+k) σ))))*

shows *f ((nsubn σ (cpoint f g i σ) (cpoint f g (Suc i) σ)))*

proof
(*induct i*)
case *0*
then show *?case*
proof −
have *1*: *g ((ndropn 0 σ))*
using *assms cpoint-0 cpoint-expand-0* **by** *force*
have *2*: (∃ *m. f ((nsubn σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ))))* ∧
0 < m ∧ *g ((ndropn (m+(cpoint f g 0 σ)) σ)))*
using *assms 1* **by** (*metis cpoint-expand-0*)
have *3*: (*cpoint f g 1 σ*) =
(*SOME x.* (∃ *m. f ((nsubn σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ))))*
∧ *m>0* ∧ *g ((ndropn (m+(cpoint f g 0 σ)) σ))*
∧ *x=m+(cpoint f g 0 σ))*
)

by *simp*
have *4*: *f ((nsubn σ (cpoint f g 0 σ) ((cpoint f g 1 σ))))*
using *2 3 someI-ex[of λx.* (∃ *m. f ((nsubn σ (cpoint f g 0 σ)*
(m+(cpoint f g 0 σ))))

$\wedge$ *m*>*0* $\wedge$ *g* ((*ndropn* (*m*+(*cpoint f g 0* σ)) σ))
$\wedge$ *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* ((*ndropn* (*cpoint f g* (*Suc i*) σ) σ))
using *assms cpoint-0* **by** *blast*
have *n2*: ($\exists$ *m*. *f* ((*nsubn* σ (*cpoint f g* (*Suc i*) σ) (*m*+(*cpoint f g* (*Suc i*) σ)))) $\wedge$
0 < *m* $\wedge$ *g* ((*ndropn* (*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* ((*nsubn* σ (*cpoint f g* (*Suc i*) σ) (*m*+(*cpoint f g* (*Suc i*) σ))))
$\wedge$ *m*>*0* $\wedge$ *g* ((*ndropn* (*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*: *f* ((*nsubn* σ (*cpoint f g* (*Suc i*) σ) ((*cpoint f g* (*Suc* (*Suc i*)) σ))))
using *n2 n3 someI-ex*[*of* λ*x*. ($\exists$ *m*. *f* ((*nsubn* σ (*cpoint f g* (*Suc i*) σ)
(*m*+(*cpoint f g* (*Suc i*) σ))))
$\wedge$ *m*>*0* $\wedge$ *g* ((*ndropn* (*m*+(*cpoint f g* (*Suc i*) σ)) σ))
$\wedge$ *x*=*m*+(*cpoint f g* (*Suc i*) σ))] **by** *auto*
from *n4* **show** *?thesis* **by** *auto*
qed
qed

lemma *cpoint-3a*:
m>*0* $\wedge$ *x*=*m*+(*cpoint f g* (*Suc i*) σ) $\Longrightarrow$ (*cpoint f g* (*Suc i*) σ) < *x*
by *auto*

lemma *cpoint-3*:
assumes $\neg$*nfinite* σ $\wedge$ *g* σ $\wedge$
($\forall$ *k*. *g* ((*ndropn k* σ)) $\longrightarrow$
($\exists$ *m*. *f* ((*nsubn* σ *k* (*m*+*k*))) $\wedge$
0 < *m* $\wedge$ *g* ((*ndropn* (*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* ((*ndropn 0* σ))
using *assms cpoint-0 cpoint-expand-0* **by** *force*
have *2*: ($\exists$ *m*. *f* ((*nsubn* σ (*cpoint f g 0* σ) (*m*+(*cpoint f g 0* σ)))) $\wedge$
0 < *m* $\wedge$ *g* ((*ndropn* (*m*+(*cpoint f g 0* σ)) σ)))

using *assms 1* **by** (*metis cpoint-expand-0*)
have *3*: (*cpoint f g 1 σ*) =
(*SOME x*. (∃ *m*. *f* ((*nsubn σ* (*cpoint f g 0 σ*) (*m*+(*cpoint f g 0 σ*))))
∧ *m*>*0* ∧ *g* ((*ndropn* (*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* λ*x*. (∃ *m*. *f* ((*nsubn σ* (*cpoint f g 0 σ*) (*m*+(*cpoint f g 0 σ*))))
∧ *m*>*0* ∧ *g* ((*ndropn* (*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* ((*ndropn* (*cpoint f g* (*Suc i*) *σ*) *σ*))
using *assms cpoint-0* **by** *blast*
have *n2*: (∃ *m*. *f* ((*nsubn σ* (*cpoint f g* (*Suc i*) *σ*) (*m*+(*cpoint f g* (*Suc i*) *σ*)))) ∧
0 < *m* ∧ *g* ((*ndropn* (*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* ((*nsubn σ* (*cpoint f g* (*Suc i*) *σ*) (*m*+(*cpoint f g* (*Suc i*) *σ*))))
∧ *m*>*0* ∧ *g* ((*ndropn* (*m*+(*cpoint f g* (*Suc i*) *σ*)) *σ*))
∧ *x*=*m*+(*cpoint f g* (*Suc i*) *σ*))
)
using *cpoint-expand-n* **by** *blast*
have *n4*: (∃ *m*. *f* ((*nsubn σ* (*cpoint f g* (*Suc i*) *σ*) (*m*+(*cpoint f g* (*Suc i*) *σ*))))
∧ *m*>*0* ∧ *g* ((*ndropn* (*m*+(*cpoint f g* (*Suc i*) *σ*)) *σ*))
∧ (*cpoint f g* (*Suc* (*Suc i*)) *σ*)=*m*+(*cpoint f g* (*Suc i*) *σ*))
using *n2 n3 someI-ex*[*of* λ*x*. (∃ *m*. *f* ((*nsubn σ* (*cpoint f g* (*Suc i*) *σ*) (*m*+(*cpoint f g* (*Suc i*) *σ*))))
∧ *m*>*0* ∧ *g* ((*ndropn* (*m*+(*cpoint f g* (*Suc i*) *σ*)) *σ*))
∧ *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

primcorec *cpl* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *nat* ⇒ *'a intervals* ⇒ *nat nellist*
where *cpl f g x σ* = *NCons* (*cpoint f g x σ*) (*cpl f g* (*Suc x*) *σ*)

lemma
nnth (*cpl f g 0 σ*) *0* = *0*
by (*metis cpl.disc-iff cpl.simps*(*2*) *cpoint-expand-0 nhd-conv-nnth*)

lemma

nnth (*cpl f g 0 σ*) *1* = *cpoint f g 1 σ*
by (*metis One-nat-def cpl.code cpl.disc-iff cpl.simps*(*2*) *nhd-conv-nnth nnth-Suc-NCons*)

lemma *nnth-cpl*:
nnth (*cpl f g x σ*) *i* = *cpoint f g* (*x*+*i*) *σ*
proof (*induct i arbitrary*: *x*)
case *0*
then show *?case* **by** (*simp add*: *nnth-0-conv-nhd*)
next
case (*Suc i*)
then show *?case* **by** (*metis add-Suc-shift cpl.simps*(*3*) *nnth-ntl*)
qed

lemma *cpl-infinite*:
¬*nfinite* (*cpl f g x σ*)
proof
assume *nfinite* (*cpl f g x σ*)
thus *False*
proof (*induct zs*≡(*cpl f g x σ*) *arbitrary*: *x rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*metis cpl.disc-iff nellist.disc*(*1*))
next
case (*NCons x nell*)
then show *?case* **by** (*metis cpl.simps*(*3*) *nellist.sel*(*5*))
qed
qed

lemma *AOmegaInductSem*:
(*w* ⊨ (*inf* ∧ *g* ∧ □(*g* ⟶ ((*f* ∧ *more*) ∧ *finite*);*g*)) ⟶ *aomega f*)
proof –
have *1*: (*w* ⊨ (*inf* ∧ *g* ∧ □(*g* ⟶ ((*f* ∧ *more*) ∧ *finite*);*g*))) =
(¬*nfinite w* ∧ *g w* ∧
(∀ *k*. *g* ((*ndropn k w*)) ⟶
(∃ *m*. *f* ((*nsubn w k* (*m*+*k*))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+*k*) *w*)))))
using *AOmegaInductSem-help*[*of g f w*]
by (*simp add*: *add.commute*)
have *2*: (¬ *nfinite w* ∧ *g w* ∧
(∀ *k*. *g* ((*ndropn k w*)) ⟶
(∃ *m*. *f* ((*nsubn w k* (*m*+*k*))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+*k*) *w*))))) ⟶
nidx (*cpl f g 0 w*)
using *nidx-expand*[*of* (*cpl f g 0 w*)] *nnth-cpl*[*of f g 0 w*]
by (*metis add-0 cpoint-3*)
have *3*: (¬*nfinite w* ∧ *g w* ∧
(∀ *k*. *g* ((*ndropn k w*)) ⟶
(∃ *m*. *f* ((*nsubn w k* (*m*+*k*))) ∧
0 < *m* ∧ *g* ((*ndropn* (*m*+*k*) *w*))))) ⟶

$(\forall\, i.\ f\ (\ (nsubn\ w\ (nnth\ (cpl\ f\ g\ 0\ w)\ i)$
$(nnth\ (cpl\ f\ g\ 0\ w)\ \ (Suc\ i)))))$
using *1 cpoint-2* **by** (*metis add-0 nnth-cpl*)
have *31*: $(\neg nfinite\ w \wedge g\ w \wedge$
$(\forall\, k.\ g\ (\ (ndropn\ k\ w)) \longrightarrow$
$(\exists\, m.\ f\ (\ (nsubn\ w\ k\ (m{+}k))) \wedge$
$0 < m \wedge g\ (\ (ndropn\ (m{+}k)\ w))))) \longrightarrow$
$(\forall\, i.\ (nnth\ (cpl\ f\ g\ 0\ w)\ i) \le nlength\ w)$
by (*simp add*: *nfinite-conv-nlength-enat*)
have *4*: $(\neg\ nfinite\ w \wedge g\ w \wedge$
$(\forall\, k.\ g\ (\ (ndropn\ k\ w)) \longrightarrow$
$(\exists\, m.\ f\ (\ (nsubn\ w\ k\ (m{+}k))) \wedge$
$0 < m \wedge g\ (\ (ndropn\ (m{+}k)\ w))))) \longrightarrow$
$\neg nfinite\ w \wedge \neg nfinite\ (cpl\ f\ g\ 0\ w) \wedge nnth\ (cpl\ f\ g\ 0\ w)\ 0 = 0 \wedge$
$nidx\ \ (cpl\ f\ g\ 0\ w)\ \ \wedge$
$(\forall\, i.\ (nnth\ (cpl\ f\ g\ 0\ w)\ i) \le nlength\ w) \wedge$
$(\forall\, i.\ f\ (\ (nsubn\ w\ (nnth\ (cpl\ f\ g\ 0\ w)\ \ i)$
$(nnth\ (cpl\ f\ g\ 0\ w)\ \ (Suc\ i)))))$
using *2 3 31*
by (*simp add*: *cpl-infinite nnth-0-conv-nhd*)
have *5*: $(\neg nfinite\ w \wedge g\ w \wedge$
$(\forall\, k.\ g\ (\ (ndropn\ k\ w)) \longrightarrow$
$(\exists\, m.\ f\ (\ (nsubn\ w\ k\ (m{+}k))) \wedge$
$0 < m \wedge g\ (\ (ndropn\ (m{+}k)\ w))))) \longrightarrow$
$\neg nfinite\ w \wedge$
$(\exists\, (ls).\ \ \neg nfinite\ ls \wedge nnth\ ls\ 0 = 0 \wedge\ \ nidx\ ls \wedge$
$(\forall\, i.\ (nnth\ ls\ i) \le nlength\ w) \wedge$
$(\forall\, i.\ f\ (\ (nsubn\ w\ (nnth\ ls\ \ i)$
$(nnth\ ls\ \ (Suc\ i))))) \wedge$
$ls = (cpl\ f\ g\ 0\ w))$
using *4* **by** *auto*
have *6*: $(\neg nfinite\ w \wedge g\ w \wedge$
$(\forall\, k.\ g\ (\ (ndropn\ k\ w)) \longrightarrow$
$(\exists\, m.\ f\ (\ (nsubn\ w\ k\ (m{+}k))) \wedge$
$0 < m \wedge g\ (\ (ndropn\ (m{+}k)\ w))))) \longrightarrow (w \models aomega\ f)$
using *3 5* **unfolding** *aomega-d-def* **by** *blast*
have *7*: $(w \models (inf \wedge g \wedge \Box(g \longrightarrow ((f \wedge more) \wedge finite);g)) \longrightarrow (aomega\ f)\ \ )$
using *6 1* **by** *auto*
from *7* **show** *?thesis* **by** *blast*
qed

lemma *AOmegaInduct*:
$\vdash (inf \wedge g \wedge \Box(g \longrightarrow ((f \wedge more) \wedge finite);g)) \longrightarrow aomega\ f$
unfolding *Valid-def* **using** *AOmegaInductSem*[*of g f*] **by** *blast*

lemma *AOmegaWeakCoInduct*:
assumes $\vdash g \longrightarrow ((f \wedge more) \wedge finite);g$
shows $\vdash inf \wedge g \longrightarrow aomega\ f$
proof −

have *1*: $\vdash \Box(g \longrightarrow ((f \wedge \mathit{more}) \wedge \mathit{finite});g)$
using *assms* **by** (*simp add*: *BoxGen*)
show *?thesis* **using** *assms AOmegaInduct*[*of g f*]
using *1* **by** *fastforce*
qed

9.1.3 Weak Omega

lemma *ChopSemMono* [*mono*]:
$(w \models\ f\ ;g) =$
$(\ (\exists\, n.\ \mathit{enat}\ n \le \mathit{nlength}\ w \wedge f\ (\ (\mathit{ntaken}\ n\ w))\ \wedge g\ \ (\mathit{ndropn}\ n\ \ w)) \vee$
$(\neg \mathit{nfinite}\ w \wedge f\ w))$
by (*simp add*: *itl-defs*)

coinductive *womega-d* :: (*'a*::*world*) *formula* ⇒ *'a formula*
for *F* **where**
$(s \models\ F;(\mathit{womega\text{-}d}\ F)) \Longrightarrow\ (s \models (\mathit{womega\text{-}d}\ F))$

syntax
-womega-d :: *lift* ⇒ *lift* ((-$^{\mathcal{W}}$) [*85*] *85*)

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

translations
-womega-d ⇌ *CONST womega-d*

lemma *WOmegaIntros*:
$\vdash\ f;(\mathit{womega}\ f) \longrightarrow (\mathit{womega}\ f)$
using *womega-d.intros* **using** *unl-lift2* **by** *blast*

lemma *WOmegaCases*:
$(w \models (\mathit{womega}\ F)) \Longrightarrow$
$(w \models\ \ F;(\mathit{womega}\ F) \Longrightarrow P) \Longrightarrow P$
using *womega-d.cases*[*of F w P*] **by** *auto*

lemma *WOmegaUnrollSem*:
$(s \models (\mathit{womega}\ f)) = (s \models f;(\mathit{womega}\ f))$
using *womega-d.cases*[*of f*]
by (*metis womega-d.simps*)

lemma *WOmegaUnroll*:
$\vdash (\mathit{womega}\ f) = f;(\mathit{womega}\ f)$
using *WOmegaUnrollSem*
using *unl-lift2* **by** *blast*

lemma *WOmegaCoinductSem*:
assumes $\bigwedge x.\ \ X\ x \Longrightarrow x \models\ \ F;(X\ \ \vee\ womega\ F)$
shows $x\ \models X\ \ \ \longrightarrow womega\ F$
using *assms womega-d.coinduct*[*of X x F*]
by (*auto simp add*: *chop-defs*)

lemma *WOmegaCoinduct*:
assumes $\vdash X\ \longrightarrow\ \ F;(X \vee (womega\ F))$
shows $\vdash X\ \longrightarrow (womega\ F)$
using *assms WOmegaCoinductSem*[*of X F*]
by (*simp add*: *Valid-def*)

lemma *WOmegaWeakCoinductSem*:
assumes $\bigwedge x.\ \ x \models X \Longrightarrow x \models\ \ F;X$
shows $x\ \models X\ \ \ \longrightarrow womega\ F$
using *assms womega-d.coinduct*[*of X x F*]
by (*auto simp add*: *chop-defs*)

lemma *WOmegaWeakCoinduct*:
assumes $\vdash X\ \longrightarrow\ \ F;X$
shows $\vdash X\ \longrightarrow (womega\ F)$
using *assms WOmegaWeakCoinductSem*[*of X F*]
by (*simp add*: *Valid-def*)

lemma *Omega-WOmega*:
$\vdash (omega\ f) = (\ (womega\ ((f\ \wedge\ more)\ \wedge\ finite)))$
proof –
have *3*: $\vdash (\ (womega\ ((f\ \wedge\ more)\ \wedge\ finite))) \longrightarrow (omega\ f)$
using *OmegaWeakCoinduct*[*of LIFT* (*womega* ((*f* ∧ *more*) ∧ *finite*)) *f*]
by (*simp add*: *WOmegaUnroll int-iffD1*)
have *4*: $\vdash\ \ (omega\ f\ ) \longrightarrow ((f\ \wedge\ more)\ \wedge\ finite);(omega\ f)$
by (*simp add*: *OmegaUnroll int-iffD1*)
have *5*:$\vdash (omega\ f) \longrightarrow (womega\ ((f\ \wedge\ more)\ \wedge\ finite))$
using *WOmegaWeakCoinduct*[*of LIFT* (*omega f*) *LIFT* (*f* ∧ *more*) ∧ *finite*]
4 **by** *blast*
show *?thesis*
by (*simp add*: *3 5 int-iffI*)
qed

lemma *WOmegaInducthelp*:
$\vdash (\ g \wedge \Box(g \longrightarrow f;g))$
$\longrightarrow$
$f;(\ (\ g \wedge \Box(g \longrightarrow f;g))\ )$
proof –
have *1*: $\vdash (\ g \wedge \Box(g \longrightarrow f;g)) = (\ g \wedge (g \longrightarrow f;g) \wedge \Box(g \longrightarrow f;g))$
using *BoxEqvAndBox*[*of LIFT* $g \longrightarrow f;g$] **by** *fastforce*
have *2*: $\vdash (\ g \wedge (g \longrightarrow f;g) \wedge \Box(g \longrightarrow f;g)) = (\ g \wedge (\ f;g) \wedge \Box(g \longrightarrow f;g))$
by *fastforce*
have *4*: $\vdash \Box(g \longrightarrow f;g) = \Box\ (\Box\ (g \longrightarrow f;g))$
by (*simp add*: *BoxEqvBoxBox*)

have *5*: $\vdash$ ($g \wedge$ (f;g) $\wedge$ $\Box$($g \longrightarrow f$;g)) = ($g \wedge$ (f;g) $\wedge\Box$($\Box$($g \longrightarrow f$;g)))
using *4* **by** *fastforce*
have *6*: $\vdash$ ($g \wedge$ (f;g) $\wedge$ $\Box$($\Box$($g \longrightarrow f$;g))) $\longrightarrow$ f;(($g \wedge \Box$($g \longrightarrow f$;g)))
using *ChopAndBoxImport*[*of f g LIFT* $\Box$($g \longrightarrow f$;g)]
by (*meson Prop01 Prop05*)
show *?thesis*
by (*metis 1 2 5 6 int-eq*)
qed

lemma *OmegaInducthelp*:
$\vdash$ ($g \wedge \Box$($g \longrightarrow$ (($f \wedge$ *more*) $\wedge$ *finite*);g))
$\longrightarrow$
(($f \wedge$ *more*) $\wedge$ *finite*);(($g \wedge \Box$($g \longrightarrow$ (($f \wedge$ *more*) $\wedge$ *finite*);g)))
using *WOmegaInducthelp* **by** *blast*

lemma *WOmegaInduct*:
$\vdash$ ($g \wedge \Box$($g \longrightarrow f$;g)) $\longrightarrow$ *womega f*
by (*simp add*: *WOmegaInducthelp WOmegaWeakCoinduct*)

lemma *OmegaInduct*:
$\vdash$ ($g \wedge \Box$($g \longrightarrow$ (($f \wedge$ *more*) $\wedge$ *finite*);g)) $\longrightarrow$ *omega f*
by (*simp add*: *OmegaInducthelp OmegaWeakCoinduct*)

lemma *WOmega-coinduct*:
assumes $\vdash g \longrightarrow h \vee f$;$g$
shows $\vdash g \longrightarrow$ (*womega f*) $\vee$ (*wpowerstar f*);h
proof −
have *1*: $\vdash$ (*wpowerstar f*);h = (*empty* $\vee$ f;(*wpowerstar f*));h
by (*simp add*: *LeftChopEqvChop WPowerstarEqv*)
have *2*: $\vdash$ (*empty* $\vee$ f;(*wpowerstar f*));h = ($h \vee$ (f;(*wpowerstar f*));h)
by (*simp add*: *EmptyOrChopEqv*)
have *3*: $\vdash$ ($\neg$($h \vee$ (f;(*wpowerstar f*));h)) = ($\neg$ $h \wedge \neg$(f;((*wpowerstar f*);h)))
by (*metis ChopAssoc int-eq int-simps*(*14*) *int-simps*(*33*))
have *31*: $\vdash$ ($\neg$((*wpowerstar f*);h)) = ($\neg$ $h \wedge \neg$(f;((*wpowerstar f*);h)))
by (*metis 1 2 3 int-eq*)
have *32*: $\vdash$ (($\neg$ $h \wedge \neg$(f;((*wpowerstar f*);h))) $\wedge$ ($h \vee f$;g)) $\longrightarrow$
($\neg$(f;((*wpowerstar f*);h)) $\wedge$ f;g)
by *force*
have *33*: $\vdash$ $g \wedge \neg$((*wpowerstar f*);h) $\longrightarrow$ (($\neg$ $h \wedge \neg$(f;((*wpowerstar f*);h))) $\wedge$ ($h \vee f$;g))
using *assms 31*
by (*metis Prop10 Prop12 int-eq int-iffD2 lift-and-com*)
have *4*: $\vdash g \wedge \neg$((*wpowerstar f*);h) $\longrightarrow$ $\neg$(f;((*wpowerstar f*);h)) $\wedge$ f;g
using *assms 31 32 33*
using *lift-imp-trans* **by** *blast*
have *5*: $\vdash$ $\neg$(f;((*wpowerstar f*);h)) $\wedge$ f;g $\longrightarrow$ f;($g \wedge \neg$ *wpowerstar f*;h)
by (*metis ChopAndNotChopImp int-eq lift-and-com*)
have *6*: $\vdash g \wedge \neg$((*wpowerstar f*);h) $\longrightarrow$ f;($g \wedge \neg$ *wpowerstar f*;h)
using *4 5 lift-imp-trans* **by** *blast*

have *7*: ⊢ $g \wedge \neg((wpowerstar\ f);h) \longrightarrow (womega\ f)$
using *WOmegaWeakCoinduct*[*of LIFT* $g \wedge \neg((wpowerstar\ f);h)$ *f*] *6*
by *blast*
show *?thesis* **using** *7* **by** *fastforce*
qed

lemma *Omega-coinduct*:
assumes ⊢ $g \longrightarrow h \vee (f \wedge more)\frown g$
shows ⊢ $g \longrightarrow (omega\ f) \vee (schopstar\ f)\frown h$
using *assms WOmega-coinduct*[*of g h LIFT* $((f \wedge more) \wedge finite)$]
Omega-WOmega[*of f*] *SChopstar-WPowerstar*[*of LIFT* $(f \wedge more)$]
by (*metis Prop10 SChopstar-finite SChopstar-WPowerstar-more inteq-reflection schop-d-def*)

9.2 Omega algebra

lemma *WOmega-coinduct-var1*:
assumes ⊢ $f \longrightarrow empty \vee\ g;f$
shows ⊢ $f \longrightarrow (womega\ g) \vee (wpowerstar\ g)$
using *assms WOmega-coinduct*[*of f LIFT empty g*]
by (*metis ChopEmpty inteq-reflection*)

lemma *Omega-coinduct-var1*:
assumes ⊢ $f \longrightarrow empty \vee ((g \wedge more)\frown f)$
shows ⊢ $f \longrightarrow (omega\ g) \vee (schopstar\ g)$
using *assms*
Omega-WOmega[*of g*] *SChopstar-WPowerstar*[*of LIFT* $(g \wedge more)$] *SChopstar-WPowerstar-more*[*of g*]
WOmega-coinduct-var1[*of f LIFT* $((g \wedge more) \wedge finite)$]
by (*metis inteq-reflection schop-d-def*)

lemma *WOmega-coinduct-var2*:
assumes ⊢ $f \longrightarrow\ g;f$
shows ⊢ $f \longrightarrow (womega\ g)$
using *assms WOmegaWeakCoinduct* **by** *blast*

lemma *Omega-coinduct-var2*:
assumes ⊢ $f \longrightarrow (g \wedge more)\frown f$
shows ⊢ $f \longrightarrow (omega\ g)$
using *assms*
Omega-WOmega[*of g*] *SChopstar-WPowerstar*[*of LIFT* $(g \wedge more)$] *SChopstar-WPowerstar-more*[*of g*]
WOmega-coinduct-var2[*of f LIFT* $((g \wedge more) \wedge finite)$]
by (*metis int-eq schop-d-def*)

lemma *WOmega-coinduct-eq*:
assumes ⊢ $f = (h \vee\ g;f)$
shows ⊢ $f \longrightarrow (womega\ g) \vee (wpowerstar\ g);h$
using *assms*
by (*simp add*: *Prop11 WOmega-coinduct*)

lemma *Omega-coinduct-eq*:
assumes ⊢ $f = (h \vee (g \wedge more)\frown f)$

shows $\vdash f \longrightarrow (omega\ g) \vee (schopstar\ g) \frown h$
using *assms*
Omega-WOmega[of g] SChopstar-WPowerstar[of LIFT $(g \wedge more)$ *] SChopstar-WPowerstar-more[of g]*
WOmega-coinduct-eq[of f h LIFT $((g \wedge more) \wedge finite)$*]*
SChopstar-finite[of g]
unfolding *schop-d-def*
by (*metis Prop10 inteq-reflection*)

lemma *WOmega-coinduct-eq-var1*:
assumes $\vdash f = (empty \vee g;f)$
shows $\vdash f \longrightarrow (womega\ g) \vee (wpowerstar\ g)$
using *assms*
using *WOmega-coinduct-var1 int-iffD1* **by** *blast*

lemma *Omega-coinduct-eq-var1*:
assumes $\vdash f = (empty \vee (g \wedge more) \frown f)$
shows $\vdash f \longrightarrow (omega\ g) \vee (schopstar\ g)$
using *assms*
Omega-WOmega[of g] SChopstar-WPowerstar[of LIFT $(g \wedge more)$ *] SChopstar-WPowerstar-more[of g]*
WOmega-coinduct-eq-var1[of f LIFT $((g \wedge more) \wedge finite)$ *]*
by (*metis int-eq schop-d-def*)

lemma *WOmega-coinduct-eq-var2*:
assumes $\vdash f = g;f$
shows $\vdash f \longrightarrow (womega\ g)$
using *assms WOmega-coinduct-var2*
using *int-iffD1* **by** *blast*

lemma *Omega-coinduct-eq-var2*:
assumes $\vdash f = (g \wedge more) \frown f$
shows $\vdash f \longrightarrow (omega\ g)$
using *assms*
Omega-WOmega[of g] SChopstar-WPowerstar[of LIFT $(g \wedge more)$ *] SChopstar-WPowerstar-more[of g]*
WOmega-coinduct-eq-var2[of f LIFT $((g \wedge more) \wedge finite)$ *]*
by (*metis int-eq schop-d-def*)

lemma *WOmega-subdist*:
$\vdash (womega\ f) \longrightarrow (womega\ (f \vee g))$
proof –
have *1*: $\vdash (womega\ f) \longrightarrow ((f \vee g));(womega\ f)$
by (*metis* (*mono-tags*, *lifting*) *OrChopEqv WOmegaUnroll intI inteq-reflection unl-lift2*)
have *2*: $\vdash (womega\ f) \longrightarrow ((f \vee g));(womega\ f)$
by (*simp add*: *1 Prop12*)
show *?thesis* **using** *WOmega-coinduct-var2 2* **by** *auto*
qed

lemma *Omega-subdist*:

$\vdash$ (*omega f*) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
proof –
 have *1*: $\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite* $\vee$ (*g* $\wedge$ *more*) $\wedge$ *finite*) =
  (((*f* $\vee$ *g*) $\wedge$ *more*) $\wedge$ *finite*)
  by *fastforce*
 show *?thesis*
 using *Omega-WOmega*[*of f*] *Omega-WOmega*[*of LIFT* (*f* $\vee$ *g*)]
 WOmega-subdist[*of LIFT* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *LIFT* ((*g* $\wedge$ *more*) $\wedge$ *finite*)] *1*
 by (*metis int-eq*)
qed

lemma *WOmega-iso*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ (*womega f*) $\longrightarrow$ (*womega g*)
using *assms WOmega-subdist*[*of f g*]
by (*metis LeftChopImpChop WOmegaUnroll WOmegaWeakCoinduct inteq-reflection*)

lemma *Omega-iso*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ (*omega f*) $\longrightarrow$ (*omega g*)
using *assms*
Omega-WOmega[*of f*] *Omega-WOmega*[*of g*]
WOmega-iso[*of LIFT* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *LIFT* ((*g* $\wedge$ *more*) $\wedge$ *finite*)]
by *fastforce*

lemma *WOmega-subdist-var*:
$\vdash$ (*womega f*) $\vee$ (*womega g*) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*meson EmptyChop Prop02 Prop04 Prop05 Prop11 WOmega-iso WOmega-subdist*)

lemma *Omega-subdist-var*:
$\vdash$ (*omega f*) $\vee$ (*omega g*) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
proof –
 have *1*: $\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite* $\vee$ (*g* $\wedge$ *more*) $\wedge$ *finite*) =
  (((*f* $\vee$ *g*) $\wedge$ *more*) $\wedge$ *finite*)
  by *fastforce*
 show *?thesis*
 using
 Omega-WOmega[*of f*] *Omega-WOmega*[*of g*] *Omega-WOmega*[*of LIFT* (*f* $\vee$ *g*)]
 WOmega-subdist-var[*of LIFT* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *LIFT* ((*g* $\wedge$ *more*) $\wedge$ *finite*)] *1*
 by (*metis int-eq*)
qed

lemma *WOmega-zero*:
$\vdash$ $\neg$(*womega* #*False*)
by (*metis AndInfChopEqvAndInf InfEqvNotFinite WOmegaUnroll int-iffD1 int-simps*(*14*) *int-simps*(*21*) *inteq-reflection*)

lemma *Omega-zero*:
$\vdash$ $\neg$(*omega* #*False*)
using *Omega-WOmega*[*of LIFT* #*False*] *WOmega-zero* **by** *fastforce*

lemma *WOmega-star-1*:
⊢ (*wpowerstar f*);(*womega f*) = (*womega f*)
proof –
have *1*: ⊢ *f*; (*womega f*) ⟶ (*womega f*)
by (*simp add*: *WOmegaUnroll int-iffD2*)
have *2*: ⊢ (*wpowerstar f*);(*womega f*) ⟶ (*womega f*)
by (*simp add*: *1 WPowerstar-inductL-var-equiv*)
have *3*: ⊢ (*womega f*) ⟶ (*wpowerstar f*);(*womega f*)
by (*meson Prop11 WOmegaUnroll WPowerstar-induct-lvar-eq2*)
show *?thesis*
by (*simp add*: *2 3 int-iffI*)
qed

lemma *Omega-star-1*:
⊢ (*schopstar f*)⌢(*omega f*) = (*omega f*)
using *Omega-WOmega*[*of f*]
WOmega-star-1[*of LIFT* ((*f* ∧ *more*) ∧ *finite*)]
SChopstar-WPowerstar[*of LIFT* (*f* ∧ *more*)]
SChopstar-WPowerstar-more[*of f*] *SChopstar-finite*[*of f*]
unfolding *schop-d-def*
by (*metis Prop10 inteq-reflection*)

lemma *WOmega-max-element*:
⊢ *f* ⟶ (*womega empty*)
by (*simp add*: *EmptyChop WOmegaWeakCoinduct int-iffD2*)

lemma *Omega-empty-zero*:
⊢ ¬(*omega empty*)
using *Omega-WOmega*[*of LIFT empty*] *WOmega-zero*
unfolding *empty-d-def* **by** *fastforce*

lemma *WOmega-star-3*:
⊢ (*womega* (*wpowerstar f*))
proof –
have *1*: ⊢ *empty* ⟶ (*wpowerstar f*)
using *WPowerstar-imp-empty* **by** *auto*
have *2*: ⊢ (*womega empty*) ⟶ (*womega* (*wpowerstar f*))
by (*simp add*: *1 WOmega-iso*)
show *?thesis*
by (*meson 2 MP WOmega-max-element*)
qed

lemma *WOmega-1*:
⊢ (*womega f*); *g* ⟶ (*womega f*)
proof –
have *1*: ⊢ (*womega f*); *g* ⟶ *f*;((*womega f*); *g*)
by (*metis ChopAssoc WOmegaUnroll int-eq int-iffD1*)
show *?thesis* **using** *WOmega-coinduct-var2*[*of LIFT* (*womega f*); *g f*]
using *1* **by** *auto*

qed

lemma *Omega-1*:
$\vdash$ (*omega f*); *g* $\longrightarrow$ (*omega f*)
using *Omega-WOmega*[*of f*]
WOmega-1[*of LIFT* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *g*]
by (*metis int-eq*)

lemma *WOmega-sup-id*:
assumes $\vdash$ *empty* $\longrightarrow$ *g*
shows $\vdash$ (*womega f*);*g* = (*womega f*)
by (*meson ChopEmpty Prop11 RightChopImpChop WOmega-1 assms lift-imp-trans*)

lemma *Omega-sup-id*:
assumes $\vdash$ *empty* $\longrightarrow$ *g*
shows $\vdash$ (*omega f*);*g* = (*omega f*)
using *assms*
by (*metis Omega-WOmega WOmega-sup-id int-eq*)

lemma *WOmega-top*:
$\vdash$ (*womega f*);(*womega empty*) = (*womega f*)
by (*simp add*: *WOmega-max-element WOmega-sup-id*)

lemma *supid-WOmega*:
assumes $\vdash$ *empty* $\longrightarrow$ *f*
shows $\vdash$ (*womega f*) = (*womega empty*)
using *assms*
by (*simp add*: *Prop11 WOmega-iso WOmega-max-element*)

lemma *WOmega-simulation*:
assumes $\vdash$ *h*; *f* $\longrightarrow$ *g* ; *h*
shows $\vdash$ *h* ; (*womega f*) $\longrightarrow$ (*womega g*)
proof –
have *1*: $\vdash$ *h* ; (*womega f*) = *h* ; (*f*; (*womega f*))
by (*simp add*: *RightChopEqvChop WOmegaUnroll*)
have *2*: $\vdash$ *h* ; (*f*; (*womega f*)) $\longrightarrow$
g ; (*h*; (*womega f*))
by (*meson ChopAssoc LeftChopImpChop Prop11 assms lift-imp-trans*)
have *3*: $\vdash$ *h* ; (*womega f*) $\longrightarrow$ *g* ; (*h* ; (*womega f*))
by (*metis 1 2 int-eq*)
show *?thesis*
using *WOmega-coinduct-var2*[*of LIFT*(*h* ; (*womega f*)) *g*] *3* **by** *blast*
qed

lemma *Omega-simulation*:
assumes $\vdash$ (*h*)$\frown$ (*f* $\wedge$ *more*) $\longrightarrow$ (*g* $\wedge$ *more*) $\frown$ (*h*)
shows $\vdash$ (*h*) $\frown$ (*omega f*) $\longrightarrow$ (*omega g*)
using *assms*

Omega-WOmega[of f] Omega-WOmega[of g]
WOmega-simulation[of LIFT ((h) ∧ finite) LIFT ((f ∧ more) ∧ finite) LIFT ((g ∧ more) ∧ finite)]
by (*metis* (*no-types, opaque-lifting*) *ChopEmpty LeftSChopImpSChop SChopAssoc int-eq schop-d-def*)

lemma *WOmega-WOmega*:
⊢ (*womega* (*womega f*)) ⟶ (*womega f*)
by (*metis WOmegaUnroll WOmega-1 int-eq*)

lemma *WOmega-Wagner-1*:
⊢ (*womega* (*f*;*wpowerstar f*)) = (*womega f*)
proof –
 have *1*: ⊢ (*womega* (*f*;*wpowerstar f*)) =
   ((*f*;(*wpowerstar f*));(*f*;(*wpowerstar f*)));(*womega* (*f*;*wpowerstar f*))
  by (*metis ChopAssoc WOmegaUnroll int-eq*)
 have *2*: ⊢ ((*f*;(*wpowerstar f*));(*f*;(*wpowerstar f*)));(*womega* (*f*;*wpowerstar f*)) =
   f; (*womega* (*f*;*wpowerstar f*))
  by (*metis* (*no-types, lifting*) *ChopAssoc WOmegaUnroll WPowerstar-slide-var WPowerstar-trans-eq inteq-reflection*)
 have *3*: ⊢ (*womega* (*f*;*wpowerstar f*)) ⟶ (*womega f*)
  by (*metis 1 2 WOmega-coinduct-eq-var2 int-eq*)
 have *4*: ⊢ (*womega f*) ⟶ (*womega* (*f*;*wpowerstar f*))
   by (*metis ChopAssoc WOmegaUnroll WOmega-coinduct-eq-var2 WOmega-star-1 int-eq*)
 show *?thesis* **by** (*simp add*: *3 4 int-iffI*)
qed

lemma *Omega-Wagner-1*:
⊢ (*omega* (*schopstar f*)) = (*omega f*)
using *Omega-WOmega[of f]*
WOmega-Wagner-1[of LIFT ((f ∧ more) ∧ finite)]
SChopstar-WPowerstar[of LIFT (f ∧ more)]
SChopstar-WPowerstar-more[of f]
by (*metis AndMoreAndFiniteEqvAndFmore Omega-WOmega SCSAndMoreEqvAndFMoreSChop int-eq schop-d-def*)

lemma *Omega-Wagner-1-var2*:
⊢ (*omega* ((*f* ∧ *more*)⌢*schopstar f*)) = (*omega f*)
by (*metis* (*no-types, lifting*) *LeftSChopImpMoreRule Omega-WOmega Omega-Wagner-1 OrFiniteInf Prop01 Prop05 Prop10 Prop11 SCSAndMoreEqvAndMoreSChop int-iffD1 inteq-reflection*)

lemma *WOmega-Wagner-2-var*:
⊢ *f*;(*womega* (*g*;*f*)) ⟶ (*womega* (*f*;*g*))
proof –
 have *1*: ⊢ *f*;(*g*;*f*) ⟶ *f*;(*g*;*f*)
  by *auto*
 show *?thesis*
 by (*meson ChopAssoc Prop11 WOmega-simulation*)
qed

lemma *Omega-Wagner-2-var*:
⊢ (*f* ∧ *more*)⌢(*omega* ((*g* ∧ *more*)⌢(*f* ∧ *more*))) ⟶ (*omega* ((*f* ∧ *more*)⌢(*g* ∧ *more*)))

using *WOmega-Wagner-2-var*[*of LIFT* ((*f* ∧ *more*) ∧ *finite*) *LIFT* ((*g* ∧ *more*) ∧ *finite*)]
Omega-WOmega[*of LIFT* (((*g* ∧ *more*) ∧ *finite*);((*f* ∧ *more*) ∧ *finite*))]
Omega-WOmega[*of LIFT* (((*f* ∧ *more*) ∧ *finite*);((*g* ∧ *more*) ∧ *finite*))]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Omega-simulation SChopAssoc int-eq int-iffD1*)

lemma *WOmega-Wagner-2*:
⊢ (*womega* (*f*;*g*)) = *f* ; (*womega* (*g*;*f*))
proof −
have *1*: ⊢ *f* ; (*womega* (*g*;*f*)) ⟶ (*womega* (*f*;*g*))
by (*simp add*: *WOmega-Wagner-2-var*)
have *2*: ⊢ (*womega* (*f*;*g*)) = (*f*;*g*); (*womega* (*f*;*g*))
by (*simp add*: *WOmegaUnroll*)
have *3*: ⊢ (*womega* (*f*;*g*)) ⟶ *f* ; (*womega* (*g*;*f*))
by (*metis 2 ChopAssocSem RightChopImpChop Valid-def WOmega-Wagner-2-var inteq-reflection*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2*:
⊢ (*omega* ((*f* ∧ *more*)⌢(*g* ∧ *more*))) = (*f* ∧ *more*) ⌢ (*omega* ((*g* ∧ *more*)⌢(*f* ∧ *more*)))
proof −
have *1*: ⊢ (*f* ∧ *more*) ⌢ (*omega* ((*g* ∧ *more*)⌢(*f* ∧ *more*))) ⟶
(*omega* ((*f* ∧ *more*)⌢(*g* ∧ *more*)))
by (*simp add*: *Omega-Wagner-2-var*)
have *2*: ⊢ (*omega* ((*f* ∧ *more*)⌢(*g* ∧ *more*))) = ((*f* ∧ *more*)⌢(*g* ∧ *more*))⌢ (*omega* ((*f* ∧ *more*)⌢(*g* ∧ *more*)))
by (*metis AndMoreSChopAndMoreEqvAndMoreSChop OmegaUnroll int-eq schop-d-def*)
have *3*: ⊢ (*omega* ((*f* ∧ *more*)⌢(*g* ∧ *more*))) ⟶ (*f* ∧ *more*) ⌢ (*omega* ((*g* ∧ *more*)⌢(*f* ∧ *more*)))
by (*metis* (*no-types*, *lifting*) *2 Omega-Wagner-2-var RightSChopImpSChop SChopAssoc inteq-reflection*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2-var1*:
⊢ (*omega* ((*f*)⌢(*g* ∧ *more*))) = (*f*) ⌢ (*omega* ((*g* ∧ *more*)⌢(*f*)))
proof −
have *1*: ⊢ (*f*) ⌢ (*omega* ((*g* ∧ *more*)⌢(*f*))) ⟶ (*omega* ((*f*)⌢(*g* ∧ *more*)))
using *Omega-simulation*[*of f LIFT* (*g* ∧ *more*) ⌢ *f LIFT f* ⌢ (*g* ∧ *more*)]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Prop10 SChopAndB SChopAssoc*
SChopMoreImpMore int-iffD1 inteq-reflection lift-imp-trans)
have *2*: ⊢ (*omega* ((*f*)⌢(*g* ∧ *more*))) = ((*f*)⌢(*g* ∧ *more*))⌢ (*omega* ((*f*)⌢(*g* ∧ *more*)))
by (*metis AndSChopA OmegaUnroll SChopstarMore-induct-lvar-eq SChopstar-inductL-var-equiv int-eq int-iffI schop-d-def*)
have *3*: ⊢ (*omega* ((*f*)⌢(*g* ∧ *more*))) ⟶ (*f*) ⌢ (*omega* ((*g* ∧ *more*)⌢(*f*)))
using *2 Omega-coinduct-eq-var2*[*of LIFT* (*g* ∧ *more*) ⌢ (*omega* ((*f*)⌢(*g* ∧ *more*))) *LIFT* ((*g* ∧ *more*)⌢(*f*))]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop RightSChopImpSChop SChopAs-*

soc inteq-reflection)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2-var2*:
$\vdash$ (*omega* ((*f* $\wedge$ *more*)$\frown$(*g*))) = (*f* $\wedge$ *more*) $\frown$ (*omega* ((*g*)$\frown$(*f* $\wedge$ *more*)))
proof –
have *1*: $\vdash$ (*f* $\wedge$ *more*) $\frown$ (*omega* ((*g*)$\frown$(*f* $\wedge$ *more*))) $\longrightarrow$ (*omega* ((*f* $\wedge$ *more*)$\frown$(*g*)))
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Omega-simulation SChopAndA SChopAssoc int-eq*)
have *2*: $\vdash$ (*omega* ((*f* $\wedge$ *more*)$\frown$(*g*))) = ((*f* $\wedge$ *more*)$\frown$(*g*))$\frown$ (*omega* ((*f* $\wedge$ *more*)$\frown$(*g*)))
by (*metis AndMoreSChopAndMoreEqvAndMoreSChop OmegaUnroll inteq-reflection schop-d-def*)
have *3*: $\vdash$ (*omega* ((*f* $\wedge$ *more*)$\frown$(*g*))) $\longrightarrow$ (*f* $\wedge$ *more*) $\frown$ (*omega* ((*g*)$\frown$(*f* $\wedge$ *more*)))
by (*meson 2 Omega-Wagner-2-var1 Prop04 RightSChopEqvSChop SChopAssoc int-iffD1 int-iffD2 int-iffI*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *WOmega-Wagner-3*:
assumes $\vdash$ (*f* ; (*womega* (*f* $\vee$ *g*)) $\vee$ *h*) = (*womega* (*f* $\vee$ *g*))
shows $\vdash$ (*womega* (*f* $\vee$ *g*)) = ((*womega f*) $\vee$ (*wpowerstar f*);*h*)
proof –
have *1*: $\vdash$ (*womega* (*f* $\vee$ *g*)) $\longrightarrow$ ((*womega f*) $\vee$ (*wpowerstar f*);*h*)
using *WOmega-coinduct*[*of LIFT* (*womega* (*f* $\vee$ *g*)) *h f*] *assms*
by *fastforce*
have *2*: $\vdash$ (*wpowerstar f*);*h* $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*metis ChopImpChop Prop03 WOmega-star-1 WPowerstar-subdist assms inteq-reflection*)
have *3*: $\vdash$ ((*womega f*) $\vee$ (*wpowerstar f*);*h*) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*meson 2 Prop02 WOmega-subdist*)
show *?thesis*
using *1 3 int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-3*:
assumes $\vdash$ ((*f* $\wedge$ *more*) $\frown$ (*omega* (*f* $\vee$ *g*)) $\vee$ (*h*)) = (*omega* (*f* $\vee$ *g*))
shows $\vdash$ (*omega* (*f* $\vee$ *g*)) = ((*omega f*) $\vee$ (*schopstar f*)$\frown$(*h*))
proof –
have *1*: $\vdash$ ((*f* $\wedge$ *more*) $\wedge$ *finite* $\vee$ (*g* $\wedge$ *more*) $\wedge$ *finite*) = (((*f* $\vee$ *g*) $\wedge$ *more*) $\wedge$ *finite*)
by *fastforce*
show *?thesis*
using *assms*
WOmega-Wagner-3[*of LIFT* ((*f* $\wedge$ *more*) $\wedge$ *finite*) *LIFT* ((*g* $\wedge$ *more*) $\wedge$ *finite*) *h*]
Omega-WOmega[*of f*] *Omega-WOmega*[*of LIFT* (*f* $\vee$ *g*)]
SChopstar-WPowerstar[*of LIFT* (*f* $\wedge$ *more*)]
SChopstar-WPowerstar-more[*of f*]
unfolding *schop-d-def*
by (*metis 1 Prop10 SChopstar-finite inteq-reflection*)
qed

lemma *WOmega-Wagner-1-var*:
⊢ (*womega* ((*wpowerstar f*);*f*)) = (*womega f*)
by (*metis WOmega-Wagner-1 WOmega-Wagner-2 WOmega-star-1 int-eq*)

lemma *Omega-Wagner-1-var3*:
⊢ (*omega* ((*schopstar f*)⌢(*f* ∧ *more*))) = (*omega f*)
by (*metis Omega-Wagner-1-var2 Omega-Wagner-2-var1 Omega-star-1 inteq-reflection*)

lemma *WOmega-star-4*:
⊢ (*wpowerstar* (*womega f*)) = (*empty* ∨ (*womega f*))
proof −
have *1*: ⊢ (*wpowerstar* (*womega f*)) = (*empty* ∨ (*womega f*);(*wpowerstar* (*womega f*)))
by (*simp add*: *WPowerstarEqv*)
have *2*: ⊢ (*empty* ∨ (*womega f*);(*wpowerstar* (*womega f*))) ⟶ (*empty* ∨ (*womega f*);(*womega empty*))
by (*metis 1 WOmega-sup-id WOmega-top WPowerstar-imp-empty int-eq int-iffD2*)
have *3*: ⊢ (*wpowerstar* (*womega f*)) ⟶ (*empty* ∨ (*womega f*))
by (*metis 2 WOmega-top WPowerstarEqv inteq-reflection*)
have *4*: ⊢ (*empty* ∨ (*womega f*)) ⟶ (*wpowerstar* (*womega f*))
by (*meson Prop02 WPowerstar-ext WPowerstar-imp-empty*)
show *?thesis*
using *3 4 int-iffI* **by** *blast*
qed

lemma *WOmega-star-5*:
⊢ (*womega f*);(*wpowerstar* (*womega f*)) = (*womega f*)
proof −
have *1*: ⊢ (*womega f*);(*wpowerstar* (*womega f*)) ⟶ (*womega f*)
by (*simp add*: *WOmega-1*)
have *2*: ⊢ (*womega f*) ⟶ (*womega f*);(*wpowerstar* (*womega f*))
by (*simp add*: *WOmega-sup-id WPowerstar-imp-empty int-iffD2*)
show *?thesis*
by (*simp add*: *1 2 int-iffI*)
qed

lemma *WOmega-or-unfold*:
⊢ ((*womega f*) ∨ ((*wpowerstar f*);*g*); (*womega* (*f* ∨ *g*))) = (*womega* (*f* ∨ *g*))
proof −
have *1*: ⊢ (*womega* (*f* ∨ *g*)) = (*f*;(*womega* (*f* ∨ *g*)) ∨ *g*;(*womega* (*f* ∨ *g*)))
using *WOmegaUnroll*[*of LIFT* (*f* ∨ *g*)] **by** (*metis OrChopEqv int-eq*)
have *2*: ⊢((*wpowerstar f*);*g*); (*womega* (*f* ∨ *g*)) = (*wpowerstar f*);(*g*; (*womega* (*f* ∨ *g*)))
by (*meson ChopAssoc int-iffD1 int-iffD2 int-iffI*)
show *?thesis* **using** *1 2 WOmega-Wagner-3*[*of f g LIFT g*;(*womega* (*f* ∨ *g*))]
by (*metis int-eq*)
qed

lemma *Omega-or-unfold*:
⊢ ((*omega f*) ∨ ((*schopstar f*)⌢(*g* ∧ *more*))⌢ (*omega* (*f* ∨ *g*))) = (*omega* (*f* ∨ *g*))

proof −

have *0*: ⊢ (((*f* ∨ *g*) ∧ *more*) ∧ *finite*) = (((*f* ∧ *more*) ∧ *finite*) ∨ ((*g* ∧ *more*) ∧ *finite*))
by *fastforce*
show *?thesis* **using** *WOmega-or-unfold*[*of LIFT* ((*f* ∧ *more*) ∧ *finite*) *LIFT* ((*g* ∧ *more*) ∧ *finite*)]
Omega-WOmega[*of f*] *Omega-WOmega*[*of LIFT* (*f* ∨ *g*)]
SChopstar-WPowerstar[*of LIFT* (*f* ∧ *more*)] *SChopstar-WPowerstar-more*[*of f*]
by (*metis* (*no-types*, *lifting*) *0 Prop10 ChopAssoc SChopstar-finite SChopAssoc inteq-reflection schop-d-def*)
qed

lemma *WOmega-or-unfold-coinduct*:
⊢ (*womega* (*f* ∨ *g*)) ⟶ (*womega* (((*wpowerstar f*);*g*)))
∨ (*wpowerstar* ((*wpowerstar f*);*g*));(*womega f*)
using *WOmega-or-unfold*[*of f g*]
WOmega-coinduct-eq[*of LIFT* (*womega* (*f* ∨ *g*)) *LIFT* (*womega f*)
LIFT (*wpowerstar f*);*g*]
using *WOmega-coinduct int-iffD2* **by** *blast*

lemma *Omega-or-unfold-coinduct*:
⊢ (*omega* (*f* ∨ *g*)) ⟶ (*omega* (((*schopstar f*)⌢(*g* ∧ *more*))))
∨ (*schopstar* ((*schopstar f*)⌢(*g* ∧ *more*)))⌢(*omega f*)
using *Omega-or-unfold*[*of f g*]
Omega-coinduct-eq[*of LIFT* (*omega* (*f* ∨ *g*)) *LIFT* (*omega f*)
LIFT (*schopstar f*)⌢(*g* ∧ *more*)]
by (*metis Prop10 Prop12 RightSChopImpMoreRule int-eq int-iffD2 lift-and-com*)

lemma *WOmega-or-unfold-induct*:
⊢ (*wpowerstar* ((*wpowerstar f*);*g*));(*womega f*) ⟶ (*womega* (*f* ∨ *g*))
using *WOmega-or-unfold*[*of f g*]
using *WPowerstar-induct-leq* **by** *blast*

lemma *Omega-or-unfold-induct*:
⊢ (*schopstar* ((*schopstar f*)⌢(*g* ∧ *more*)))⌢(*omega f*) ⟶ (*omega* (*f* ∨ *g*))
using *Omega-or-unfold*[*of f g*]
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-leq inteq-reflection lift-imp-trans schop-d-def*)

lemma *WOmega-Wagner-1-gen*:
⊢ (*womega* (*f*;(*wpowerstar g*))) ⟶ (*womega* (*f* ∨ *g*))
proof −
have *01*: ⊢ *f* ⟶ (*f* ∨ *g*)
by *auto*
have *02*: ⊢ (*wpowerstar g*) ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*metis WPowerstar-subdist WPowerstar-swap inteq-reflection*)
have *03*: ⊢ (*f*;(*wpowerstar g*)) ⟶ (*f* ∨ *g*);(*wpowerstar* (*f* ∨ *g*))
using *01 02 ChopImpChop* **by** *blast*
have *1*: ⊢ (*womega* (*f*;(*wpowerstar g*))) ⟶
(*womega* ((*f* ∨ *g*);(*wpowerstar* (*f* ∨ *g*))))
by (*simp add*: *03 WOmega-iso*)
show *?thesis* **using** *WOmega-Wagner-1*
by (*metis 1 int-eq*)
qed

lemma *Omega-Wagner-1-gen*:
⊢ (*omega* ((*f* ∧ *more*)⌢(*schopstar g*))) ⟶ (*omega* (*f* ∨ *g*))

proof –
have *1*: ⊢ ((*f* ∧ *more*) ∧ *finite* ∨ (*g* ∧ *more*) ∧ *finite*) =(((*f* ∨ *g*) ∧ *more*) ∧ *finite*)
by *fastforce*
show *?thesis*
using *Omega-WOmega*[*of LIFT* ((*f* ∧ *more*)⌢(*schopstar g*))] *Omega-WOmega*[*of LIFT* (*f* ∨ *g*)]
WOmega-Wagner-1-gen[*of LIFT* ((*f* ∧ *more*) ∧ *finite*) *LIFT* ((*g* ∧ *more*) ∧ *finite*)]
SChopstar-WPowerstar[*of LIFT* (*g* ∧ *more*)] *SChopstar-WPowerstar-more*[*of g*] *1*
by (*metis AndMoreSChopAndMoreEqvAndMoreSChop Prop10 SChopImpFinite SChopstar-finite inteq-reflection schop-d-def*)
qed

lemma *WOmega-swap*:
⊢ (*womega* (*f* ∨ *g*)) = (*womega* (*g* ∨ *f*))
proof –
have *1*: ⊢ (*f* ∨ *g*) = (*g* ∨ *f*)
by *fastforce*
show *?thesis*
by (*metis 1 WOmega-star-5 int-eq*)
qed

lemma *Omega-swap*:
⊢ (*omega* (*f* ∨ *g*)) = (*omega* (*g* ∨ *f*))
proof –
have *1*: ⊢ (*f* ∨ *g*) = (*g* ∨ *f*)
by *fastforce*
show *?thesis*
by (*metis 1 Omega-star-1 inteq-reflection*)
qed

lemma *WOmega-Wagner-1-var-gen*:
⊢ (*womega* ((*wpowerstar f*);*g*)) ⟶ (*womega* (*f* ∨ *g*))
proof –
have *1*: ⊢ (*womega* ((*wpowerstar f*);*g*)) =
(*wpowerstar f*);(*womega* (*g*;(*wpowerstar f*)))
by (*simp add*: *WOmega-Wagner-2*)
have *2*: ⊢ (*womega* (*g*;(*wpowerstar f*))) ⟶ (*womega* (*g* ∨ *f*))
using *WOmega-Wagner-1-gen*[*of g f*] **by** *blast*
have *5*: ⊢ (*wpowerstar f*);(*womega* (*g*;(*wpowerstar f*)))
⟶ (*wpowerstar f*);(*womega* (*f* ∨ *g*))
by (*metis 2 RightChopImpChop WOmega-swap int-eq*)
have *6*: ⊢ (*wpowerstar f*) ⟶ (*wpowerstar* (*f* ∨ *g*))
by (*simp add*: *WPowerstar-subdist*)
have *7*: ⊢ (*wpowerstar f*);(*womega* (*f* ∨ *g*)) ⟶ (*wpowerstar* (*f* ∨ *g*));(*womega* (*f* ∨ *g*))
by (*simp add*: *6 LeftChopImpChop*)
have *8*: ⊢ (*wpowerstar* (*f* ∨ *g*));(*womega* (*f* ∨ *g*)) ⟶ (*womega* (*f* ∨ *g*))
by (*simp add*: *WOmega-star-1 int-iffD1*)

show *?thesis*
using *1 5 7 8*
by (*metis int-eq lift-imp-trans*)
qed

lemma *Omega-Wagner-1-var-gen*:
$\vdash$ (*omega* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*))) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
proof –
have *1*: $\vdash$ (*omega* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*))) =
(*schopstar f*)$\frown$(*omega* ((*g* $\wedge$ *more*)$\frown$(*schopstar f*)))
by (*simp add*: *Omega-Wagner-2-var1*)
have *2*: $\vdash$ (*omega* ((*g* $\wedge$ *more*)$\frown$(*schopstar f*))) $\longrightarrow$ (*omega* (*g* $\vee$ *f*))
using *Omega-Wagner-1-gen*[*of g f*] **by** *blast*
have *3*: $\vdash$ (*g* $\vee$ *f*) = (*f* $\vee$ *g*)
by *fastforce*
have *4*: $\vdash$ (*omega* (*g* $\vee$ *f*)) = (*omega* (*f* $\vee$ *g*))
by (*simp add*: *Omega-swap*)
have *5*: $\vdash$ (*schopstar f*)$\frown$(*omega* ((*g* $\wedge$ *more*)$\frown$(*schopstar f*)))
$\longrightarrow$ (*schopstar f*)$\frown$(*omega* (*f* $\vee$ *g*))
by (*metis 2 3 RightSChopImpSChop inteq-reflection*)
have *6*: $\vdash$ (*schopstar f*) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))
by (*metis 3 MP Prop03 SBaGen SBaSCSImpSCS int-eq*)
have *7*: $\vdash$ (*schopstar f*)$\frown$(*omega* (*f* $\vee$ *g*)) $\longrightarrow$ (*schopstar* (*f* $\vee$ *g*))$\frown$(*omega* (*f* $\vee$ *g*))
by (*simp add*: *6 LeftSChopImpSChop*)
have *8*: $\vdash$ (*schopstar* (*f* $\vee$ *g*))$\frown$(*omega* (*f* $\vee$ *g*)) $\longrightarrow$ (*omega* (*f* $\vee$ *g*))
by (*simp add*: *Omega-star-1 int-iffD1*)
show *?thesis*
using *1 5 7 8* **by** *fastforce*
qed

lemma *WOmega-Denest*:
$\vdash$ (*womega* (*f* $\vee$ *g*)) =
((*womega* ((*wpowerstar f*);*g*)) $\vee$
(*wpowerstar* ((*wpowerstar f*);*g*));(*womega f*))
proof –
have *1*: $\vdash$ (*womega* (*f* $\vee$ *g*)) $\longrightarrow$ (*womega* (((*wpowerstar f*);*g*)))
$\vee$ (*wpowerstar* ((*wpowerstar f*);*g*));(*womega f*)
by (*simp add*: *WOmega-or-unfold-coinduct*)
have *2*: $\vdash$ (*womega* ((*wpowerstar f*);*g*)) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*simp add*: *WOmega-Wagner-1-var-gen*)
have *3*: $\vdash$ (*wpowerstar* ((*wpowerstar f*);*g*));(*womega f*) $\longrightarrow$ (*womega* (*f* $\vee$ *g*))
by (*simp add*: *WOmega-or-unfold-induct*)
show *?thesis*
by (*meson 1 2 3 Prop02 int-iffI*)
qed

lemma *Omega-Denest*:
$\vdash$ (*omega* (*f* $\vee$ *g*)) =
((*omega* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*))) $\vee$
(*schopstar* ((*schopstar f*)$\frown$(*g* $\wedge$ *more*)))$\frown$(*omega f*))

proof −
have *1*: ⊢ (*omega* (*f* ∨ *g*)) ⟶ (*omega* (((*schopstar f*)⁀(*g* ∧ *more*))))
∨ (*schopstar* ((*schopstar f*)⁀(*g* ∧ *more*)))⁀(*omega f*)
by (*simp add*: *Omega-or-unfold-coinduct*)
have *2*: ⊢ (*omega* ((*schopstar f*)⁀(*g* ∧ *more*))) ⟶ (*omega* (*f* ∨ *g*))
by (*simp add*: *Omega-Wagner-1-var-gen*)
have *3*: ⊢ (*schopstar* ((*schopstar f*)⁀(*g* ∧ *more*)))⁀(*omega f*) ⟶ (*omega* (*f* ∨ *g*))
by (*simp add*: *Omega-or-unfold-induct*)
show *?thesis*
by (*meson 1 2 3 Prop02 int-iffI*)
qed

lemma *WOmega-or-refine*:
assumes ⊢ *g*; *f* ⟶ *f* ; (*wpowerstar* (*f* ∨ *g*))
shows ⊢ (*womega* (*f* ∨ *g*)) = ((*womega f*) ∨ (*wpowerstar f*);(*womega g*))
proof −
have *1*: ⊢ (*wpowerstar g*);*f* ⟶ *f*;(*wpowerstar* (*f* ∨ *g*))
using *assms WPowerstar-Quasicomm-var* **by** *blast*
have *2*: ⊢ (*womega* (*f* ∨ *g*)) = ((*womega g*) ∨ ((*wpowerstar g*);*f*);(*womega* (*f* ∨ *g*)))
by (*metis WOmega-swap WOmega-or-unfold inteq-reflection*)
have *3*: ⊢ ((*womega g*) ∨ ((*wpowerstar g*);*f*);(*womega* (*f* ∨ *g*))) ⟶
(*womega g*) ∨ (*f*;(*wpowerstar* (*f* ∨ *g*)));(*womega* (*f* ∨ *g*))
by (*metis 1 2 AndChopB Prop08 Prop10 int-iffD2 inteq-reflection*)
have *40*: ⊢ (*womega g*) ⟶ *f*;(*womega* (*f* ∨ *g*)) ∨ (*womega g*)
by (*simp add*: *Prop05*)
have *41*: ⊢ (*f*;(*wpowerstar* (*f* ∨ *g*)));(*womega* (*f* ∨ *g*)) ⟶
f;(*womega* (*f* ∨ *g*)) ∨ (*womega g*)
by (*metis* (*mono-tags*, *lifting*) *ChopAssoc WOmega-star-1 intI inteq-reflection unl-lift2*)
have *4*: ⊢ (*womega g*) ∨ (*f*;(*wpowerstar* (*f* ∨ *g*)));(*womega* (*f* ∨ *g*)) ⟶
f;(*womega* (*f* ∨ *g*)) ∨ (*womega g*)
using *40 41 Prop02* **by** *blast*
have *5*: ⊢ (*womega* (*f* ∨ *g*)) ⟶ ((*womega f*) ∨ (*wpowerstar f*);(*womega g*))
by (*metis 2 3 WOmega-coinduct WOmega-star-1 ChopAssoc inteq-reflection*)
have *6*: ⊢ ((*womega f*) ∨ (*wpowerstar f*);(*womega g*)) ⟶
(*womega* (*f* ∨ *g*)) ∨ (*wpowerstar* (*f* ∨ *g*));(*womega* (*f* ∨ *g*))
by (*metis ChopImpChop Prop02 Prop05 WOmega-star-1 WOmega-subdist WOmega-swap WPowerstar-subdist int-eq*)
have *7*: ⊢ ((*womega f*) ∨ (*wpowerstar f*);(*womega g*)) ⟶ (*womega* (*f* ∨ *g*))
using *6 WOmega-star-1*
by (*metis Prop02 int-iffD1 inteq-reflection lift-imp-trans*)
show *?thesis*
using *5 7 int-iffI* **by** *blast*
qed

lemma *Omega-or-refine*:
assumes ⊢ (*g* ∧ *more*)⁀ ((*f* ∧ *more*) ∧ *finite*) ⟶ (*f* ∧ *more*) ⁀ (*schopstar* (*f* ∨ *g*))
shows ⊢ (*omega* (*f* ∨ *g*)) = ((*omega f*) ∨ (*schopstar f*)⁀(*omega g*))
proof −

have *1*: ⊢ (*schopstar* (*g*))⁀((*f* ∧ *more*) ∧ *finite*) ⟶ (*f* ∧ *more*)⁀(*schopstar* (*f* ∨ *g*))
using *assms*
using *SChopstar-QuasicommMore-var* **by** *blast*
have *2*: ⊢ (*omega* (*f* ∨ *g*)) = ((*omega g*) ∨ ((*schopstar g*)⁀(*f* ∧ *more*))⁀(*omega* (*f* ∨ *g*)))
by (*metis Omega-swap Omega-or-unfold integ-reflection*)
have *20*: ⊢ ((*schopstar g*)⁀(*f* ∧ *more*))⁀(*omega* (*f* ∨ *g*)) =
((*schopstar g*)⁀((*f* ∧ *more*) ∧ *finite*))⁀(*omega* (*f* ∨ *g*))
by (*metis* (*no-types*, *lifting*) *AndMoreAndFiniteEqvAndFmore Prop10 Prop12 SChopAssoc int-eq int-iffD2 itl-def*(*9*))
have *25*: ⊢ ((*schopstar g*)⁀(*f* ∧ *more*))⁀(*omega* (*f* ∨ *g*)) ⟶
((*f* ∧ *more*)⁀(*schopstar* (*f* ∨ *g*)))⁀(*omega* (*f* ∨ *g*))
using *1 20* **by** (*metis LeftSChopImpSChop integ-reflection*)
have *3*: ⊢ ((*omega g*) ∨ ((*schopstar g*)⁀(*f* ∧ *more*))⁀(*omega* (*f* ∨ *g*))) ⟶
(*omega g*) ∨ ((*f* ∧ *more*)⁀(*schopstar* (*f* ∨ *g*)))⁀(*omega* (*f* ∨ *g*))
using *25* **by** *auto*
have *4*: ⊢ (*omega g*) ∨ ((*f* ∧ *more*)⁀(*schopstar* (*f* ∨ *g*)))⁀(*omega* (*f* ∨ *g*)) ⟶
(*omega g*) ∨ (*f* ∧ *more*)⁀(*omega* (*f* ∨ *g*))
by (*metis* (*mono-tags*, *lifting*) *Omega-star-1 SChopAssoc intI int-eq intensional-rews*(*3*))
have *5*: ⊢ (*omega* (*f* ∨ *g*)) ⟶ ((*omega f*) ∨ (*schopstar f*)⁀(*omega g*))
by (*metis 2 3 Omega-coinduct Omega-star-1 SChopAssoc integ-reflection*)
have *50*: ⊢ (*omega f*) ⟶ (*omega* (*f* ∨ *g*))
by (*simp add*: *Omega-subdist*)
have *51*: ⊢ (*omega g*) ⟶ (*omega* (*f* ∨ *g*))
by (*metis Omega-swap Omega-subdist integ-reflection*)
have *52*: ⊢ (*schopstar f*) ⟶ (*schopstar* (*f* ∨ *g*))
by (*simp add*: *SChopstar-subdist*)
have *6*: ⊢ ((*omega f*) ∨ (*schopstar f*)⁀(*omega g*)) ⟶
(*omega* (*f* ∨ *g*)) ∨ (*schopstar* (*f* ∨ *g*))⁀(*omega* (*f* ∨ *g*))
by (*metis 50 51 52 Omega-star-1 Prop02 Prop05 SChopImpSChop integ-reflection*)
have *7*: ⊢ ((*omega f*) ∨ (*schopstar f*)⁀(*omega g*)) ⟶ (*omega* (*f* ∨ *g*))
using *6 Omega-star-1* **by** *fastforce*
show *?thesis*
using *5 7 int-iffI* **by** *blast*
qed

lemma *WOmega-bachmair-dershowitz*:
assumes ⊢ *g*;*f* ⟶ *f*;(*wpowerstar* (*f* ∨ *g*))
shows (⊢ (*womega* (*f* ∨ *g*)) = *#False*) = (⊢ ((*womega f*) ∨ (*womega g*)) = *#False*)
proof −
have *1*: (⊢ (*womega* (*f* ∨ *g*)) = *#False*) ⟹ (⊢ ((*womega f*) ∨ (*womega g*)) = *#False*)
using *assms*
by (*metis Prop10 WOmega-subdist-var int-eq int-simps*(*18*))
have *2*: (⊢ ((*womega f*) ∨ (*womega g*)) = *#False*) ⟹ (⊢ (*womega* (*f* ∨ *g*)) = *#False*)
using *assms WOmega-or-refine*[*of g f*]
by (*metis Prop03 Prop10 WOmega-star-1 int-simps*(*18*) *int-simps*(*25*) *int-simps*(*26*) *integ-reflection*)
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *Omega-bachmair-dershowitz*:
assumes $\vdash (g \wedge more) \frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g))$
shows $(\vdash (omega\ (f \vee g)) = \#False) = (\vdash ((omega\ f) \vee (omega\ g)) = \#False)$
proof –
have *1*: $(\vdash (omega\ (f \vee g)) = \#False) \Longrightarrow (\vdash ((omega\ f) \vee (omega\ g)) = \#False)$
using *assms*
by (*metis Prop10 Omega-subdist-var int-eq int-simps*(*18*))
have *2*: $(\vdash ((omega\ f) \vee (omega\ g)) = \#False) \Longrightarrow (\vdash (omega\ (f \vee g)) = \#False)$
using *assms Omega-or-refine*[*of g f*]
by (*metis Prop03 SChopRightFalse int-simps*(*14*) *int-simps*(*16*) *int-simps*(*21*) *int-simps*(*9*) *inteq-reflection*)
show *?thesis*
using *1 2* **by** *blast*
qed

lemma *Omega-and-more-imp-more*:
$\vdash (omega\ (f \wedge more)) \longrightarrow more$
by (*metis AndSChopB MoreSChopImpMore OmegaUnroll inteq-reflection lift-imp-trans schop-d-def*)

lemma *WOmega-and-more-imp-more*:
$\vdash (womega\ (f \wedge more)) \longrightarrow more$
by (*metis ChopImpDi DiAndB DiMoreEqvMore WOmegaUnroll int-eq lift-imp-trans*)

lemma *Omega-and-fmore-imp-more*:
$\vdash (omega\ ((f \wedge more) \wedge finite)) \longrightarrow more$
by (*metis Omega-and-more-imp-more Omega-subdist OrFiniteInf inteq-reflection lift-imp-trans*)

lemma *WOmega-and-fmore-imp-more*:
$\vdash (womega\ ((f \wedge more) \wedge finite)) \longrightarrow more$
by (*metis LeftChopImpMoreRule Prop12 WOmegaUnroll int-iffD1 inteq-reflection lift-and-com*)

lemma *womega-len*:
$\vdash womega\ skip = (len\ k);womega\ skip$
proof (*induct k*)
case *0*
then show *?case*
by (*metis EmptyChop inteq-reflection wpow-0 wpower-len*)
next
case (*Suc k*)
then show *?case*
by (*metis ChopAssoc WOmegaUnroll int-eq wpow-Suc wpower-len*)
qed

lemma *omega-len*:
$\vdash omega\ skip = (len\ k) \frown omega\ skip$
proof (*induct k*)
case *0*

then show *?case*
by (*metis EmptyChop FiniteAndEmptyEqvEmpty inteq-reflection lift-and-com schop-d-def wpow-0 wpower-len*)
next
case (*Suc k*)
then show *?case*
 proof −
  have *1*: ⊢ *len Suc k* = *len k* ⌢ *skip*
    by (*simp add*: *len-k-schop*)
  have *2*: ⊢ *len Suc k* ⌢ *omega skip* = *len k*⌢(*skip*⌢*omega skip*)
    by (*metis 1 SChopAssoc inteq-reflection*)
  have *3*: ⊢ *skip*⌢*omega skip* = *omega skip*
  by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite NextSChopdef Omega-WOmega*

Omega-Wagner-1 Prop10 SkipChopFiniteImpFinite WOmega-Wagner-2 inteq-reflection next-d-def schopstar-skip-finite)
  show *?thesis*
  by (*metis 2 3 Suc inteq-reflection*)
 qed
qed

lemma *womega-exist-len*:
⊢ (*womega skip*) = (∃ *k*. *len k*; (*womega skip*))
proof −
 have *1*: ⊢ (*womega skip*) = (*wpowerstar skip*);(*womega skip*)
   by (*simp add*: *WOmegaIntros WOmega-star-1 WPowerstar-induct-lvar int-iffD2 int-iffI*)
 have *2*: ⊢ (*wpowerstar skip*) = (∃ *k*. *len k*)
   unfolding *wpowerstar-d-def*
     by (*simp add*: *ExEqvRule wpower-len*)
 have *3*: ⊢ (∃ *k*. *len k*);(*womega skip*) = (∃ *k*. *len k*; (*womega skip*))
 by (*metis ExistChop int-eq*)
 show *?thesis* **using** *1 2 3*
 by (*metis int-eq*)
qed

lemma *omega-exist-len*:
⊢ (*omega skip*) = (∃ *k*. *len k*⌢ (*omega skip*))
proof −
 have *1*: ⊢ (*omega skip*) = (*schopstar skip*)⌢(*omega skip*)
   by (*meson Omega-star-1 Prop11*)
 have *2*: ⊢ (*schopstar skip*) = (∃ *k*. *len k*)
   by (*metis ExEqvRule int-eq schopstar-skip-finite wpower-len wpowerstar-d-def wpowerstar-skip-finite*)
 have *3*: ⊢ (∃ *k*. *len k*)⌢(*omega skip*) = (∃ *k*. *len k*⌢ (*omega skip*))
   by (*metis ExistSChop inteq-reflection*)
 show *?thesis* **using** *1 2 3*
  by (*metis int-eq*)
qed

lemma *not-len-and-womega*:
⊢ ¬ (*len k* ∧ (*womega skip*))

using *womega-len*[*of Suc k*]
using *len-len-suc-not* **by** *fastforce*

lemma *not-len-and-omega*:
⊢ ¬ (*len k* ∧ (*omega skip*))
using *omega-len*[*of Suc k*]
len-len-suc-not-schop **by** *fastforce*

lemma *not-len-and-womega-var*:
⊢ (*womega skip*) ⟶ ¬(*len k*)
using *not-len-and-womega* **by** *fastforce*

lemma *not-len-and-omega-var*:
⊢ (*omega skip*) ⟶ ¬(*len k*)
using *not-len-and-omega* **by** *fastforce*

lemma *womega-skip-inf*:
⊢ (*womega skip*) ⟶ *inf*
proof −
have *1*: ⊢ *finite* = (∃ *k*. *len k*)
by (*simp add*: *intI itl-defs len-defs nfinite-conv-nlength-enat*)
have *2*: ⊢ *inf* = (∀ *k*. ¬ *len k*)
using *1* **unfolding** *finite-d-def* **by** *fastforce*
have *3*: ⊢ (*womega skip*) ⟶ (∀ *k*. ¬ *len k*)
using *not-len-and-womega-var* **by** *fastforce*
show *?thesis* **using** *2 3*
by (*metis int-eq*)
qed

lemma *omega-skip-inf*:
⊢ (*omega skip*) ⟶ *inf*
proof −
have *1*: ⊢ *finite* = (∃ *k*. *len k*)
by (*simp add*: *intI itl-defs len-defs nfinite-conv-nlength-enat*)
have *2*: ⊢ *inf* = (∀ *k*. ¬ *len k*)
using *1* **unfolding** *finite-d-def* **by** *fastforce*
have *3*: ⊢ (*omega skip*) ⟶ (∀ *k*. ¬ *len k*)
using *not-len-and-omega-var* **by** *fastforce*
show *?thesis* **using** *2 3*
by *fastforce*
qed

lemma *womega-fmore-inf*:
⊢ (*womega fmore*) ⟶ *inf*
using *wpowerstar-skip-fmore*
by (*metis WOmega-Wagner-1 int-eq womega-skip-inf*)

lemma *omega-fmore-inf*:
⊢ (*omega fmore*) ⟶ *inf*

by (*metis AndMoreAndFiniteEqvAndFmore FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*
FmoreEqvSkipChopFinite Omega-Wagner-1-var2 Prop10 SCSAndMoreEqvAndFMoreSChop SkipChopFiniteImpFinite integ-reflection omega-skip-inf schopstar-skip-finite)

lemma *womega-and-fmore-inf*:
⊢ (*womega* (*f* ∧ *fmore*)) ⟶ *inf*
by (*meson Prop12 WOmega-iso int-iffD1 lift-and-com lift-imp-trans womega-fmore-inf*)

lemma *omega-and-fmore-inf*:
⊢ (*omega* (*f* ∧ *fmore*)) ⟶ *inf*
by (*meson Prop12 Omega-iso int-iffD1 lift-and-com lift-imp-trans omega-fmore-inf*)

lemma *womega-and-empty-init*:
⊢ (*womega* (*f* ∧ *empty*)) = *init f*
proof–
have *1*: ⊢ (*womega* (*f* ∧ *empty*)) ⟶ *init f*
by (*metis InitAndEmptyEqvAndEmpty StateChopExportA WOmegaUnroll inteq-reflection*)
have *2*: ⊢ *init f* ⟶ (*womega* (*f* ∧ *empty*))
unfolding *init-d-def* **using** *WOmega-coinduct-eq-var2*[*of LIFT* (*empty* ∧ *f*);#*True LIFT* (*f* ∧ *empty*)]
by (*metis* (*no-types*, *lifting*) *AndChopCommute AndEmptyChopAndEmptyEqvAndEmpty ChopAssoc inteq-reflection*)
show *?thesis*
by (*simp add*: *1 2 int-iffI*)
qed

lemma *omega-and-empty*:
⊢ ¬(*omega* (*f* ∧ *empty*))
proof –
have *1*: ⊢ (*omega* (*f* ∧ *empty*)) = ((*f* ∧ *empty*) ∧ *more*)⁀(*omega* (*f* ∧ *empty*))
by (*simp add*: *OmegaUnroll schop-d-def*)
have *2*: ⊢ ((*f* ∧ *empty*) ∧ *more*) = #*False*
unfolding *empty-d-def* **by** *auto*
show *?thesis*
by (*metis 1 2 NotDfFalse SChopImpDf int-simps*(*14*) *inteq-reflection lift-imp-trans*)
qed

lemma *EmptyOrMoreSplit*:
⊢ *f* = ((*f* ∧ *empty*) ∨ (*f* ∧ *more*))
unfolding *empty-d-def* **by** *auto*

lemma *womega-split-empty-more*:
⊢ (*womega f*) = ((*womega* (*f* ∧ *more*)) ∨ *wpowerstar f*;*init f*)
proof –
have *1*: ⊢ (*womega* ((*f* ∧ *empty*) ∨ (*f* ∧ *more*))) =
(*womega* (*wpowerstar* (*f* ∧ *empty*);(*f* ∧ *more*)) ∨
wpowerstar (*wpowerstar* (*f* ∧ *empty*);(*f* ∧ *more*));*womega* (*f* ∧ *empty*))
using *WOmega-Denest*[*of LIFT f* ∧ *empty LIFT f* ∧ *more*] **by** *blast*
have *2*: ⊢ (*womega* (*wpowerstar* (*f* ∧ *empty*);(*f* ∧ *more*))) = (*womega* (*f* ∧ *more*))

by (*metis EmptyChop WPowerstar-and-empty int-eq*)
have *3*: ⊢ *wpowerstar* (*wpowerstar* (*f* ∧ *empty*);(*f* ∧ *more*)) = *wpowerstar f*
by (*metis EmptyChop WPowerstar-and-empty WPowerstar-more-absorb int-eq*)
show *?thesis*
by (*metis 1 2 3 EmptyOrMoreSplit inteq-reflection womega-and-empty-init*)
qed

lemma *omega-split-empty-more*:
⊢ (*omega f*) = ((*omega* (*f* ∧ *more*)))
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb Omega-Wagner-1 SChopstar-WPowerstar int-eq*)

lemma *omega-split-empty-more-var*:
⊢ (*omega* (*f* ∧ *more*)) = (*omega* (*f* ∧ *fmore*))
by (*metis AndMoreSChopEqvAndFmoreChop ChopEmpty EmptySChop Omega-Wagner-2-var1 inteq-reflection*)

lemma *omega-imp-inf*:
⊢ (*omega f*) ⟶ *inf*
by (*metis int-eq omega-and-fmore-inf omega-split-empty-more omega-split-empty-more-var*)

lemma *inf-imp-omega-skip*:
⊢ *inf* ⟶ (*omega skip*)
proof −
have *1*: ⊢ ((*skip* ∧ *more*) ∧ *finite*) = *skip*
by (*metis DiIntro DiSkipEqvMore Prop10 WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)
have *2*: ⊢ *inf* ⟶ *skip*;*inf*
by (*metis AndInfEqvChopFalse ChopAndInf MoreAndInfEqvInf MoreEqvSkipChopTrue infinite-d-def int-eq int-iffD1*)
have *3*: ⊢ *inf* ⟶ ((*skip* ∧ *more*) ∧ *finite*);*inf*
using *1 2* **by** (*metis int-eq*)
show *?thesis*
using *3 OmegaWeakCoinduct*[*of LIFT inf LIFT skip*] **by** *blast*
qed

lemma *Omega-schopstar-max-element*:
⊢ *f* ⟶ (*omega skip*) ∨ (*schopstar skip*)
proof −
have *1*: ⊢ (*schopstar skip*) = *finite*
by (*meson Prop11 schopstar-skip-finite*)
have *2*: ⊢ (*omega skip*) = *inf*
by (*simp add*: *inf-imp-omega-skip int-iffI omega-skip-inf*)
show *?thesis* **using** *1 2* **unfolding** *finite-d-def* **by** *fastforce*
qed

lemma *Omega-fmore-absorb*:

$\vdash$ (*omega f*) = (*omega* (*f* $\wedge$ *fmore*))
by (*metis int-eq omega-split-empty-more omega-split-empty-more-var*)

lemma *Omega-top*:
$\vdash$ (*omega f*);(*omega empty*) = (*omega f*)
proof –
have *1*: $\vdash$ (*omega f*) $\longrightarrow$ *inf*
by (*simp add*: *omega-imp-inf*)
have *2*: $\vdash$ (*omega f*);(*omega empty*) $\longrightarrow$ (*omega f*)
by (*simp add*: *Omega-1*)
have *3*: $\vdash$ (*omega f*) $\longrightarrow$ (*omega f*);(*omega empty*)
by (*metis 1 AndInfChopEqvAndInf Prop10 int-iffD2 inteq-reflection*)
show *?thesis*
by (*simp add*: *2 3 int-iffI*)
qed

lemma *womega-and-inf*:
$\vdash$ (*womega* (*f* $\wedge$ *inf*)) = (*f* $\wedge$ *inf*)
by (*metis AndInfChopEqvAndInf WOmegaUnroll int-eq*)

lemma *womega-split-finite-inf*:
$\vdash$ (*womega f*) =
(*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\vee$ (*womega* (*f* $\wedge$ *finite*)))
proof –
have *0*: $\vdash$ (*womega f*) = (*womega* ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*)))
by (*metis OrFiniteInf int-eq*)
have *1*: $\vdash$ (*womega* ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*))) =
(*womega* (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\vee$
wpowerstar (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*));*womega* (*f* $\wedge$ *finite*))
using *WOmega-Denest*[*of LIFT f* $\wedge$ *finite LIFT f* $\wedge$ *inf*] **by** *blast*
have *2*: $\vdash$ (*womega* (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*))) = *wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)
by (*metis ChopAndInf int-eq womega-and-inf*)
have *3*: $\vdash$ *wpowerstar* (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) = (*empty* $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)))

by (*metis ChopAndInf WPowerstar-and-inf int-eq*)
have *4*: $\vdash$ (*empty* $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*))); *womega* (*f* $\wedge$ *finite*) =
(*womega* (*f* $\wedge$ *finite*) $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)))
by (*meson AndInfChopEqvAndInf ChopAssoc EmptyOrChopEqv Prop04 Prop06 RightChopEqvChop*)
have *5*: $\vdash$ (*womega f*) =
(*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\vee$
(*empty* $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)));*womega* (*f* $\wedge$ *finite*))
by (*metis 0 1 2 3 int-eq*)
have *6*: $\vdash$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*) $\vee$
(*empty* $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)));*womega* (*f* $\wedge$ *finite*)) =
((*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)) $\vee$
(*womega* (*f* $\wedge$ *finite*)) $\vee$ (*wpowerstar* (*f* $\wedge$ *finite*);(*f* $\wedge$ *inf*)))
using *4*
by (*metis 5 int-eq*)

have *7*: ⊢ ((*wpowerstar* (*f* ∧ *finite*);(*f* ∧ *inf*)) ∨
(*womega* (*f* ∧ *finite*)) ∨ (*wpowerstar* (*f* ∧ *finite*);(*f* ∧ *inf*))) =
((*wpowerstar* (*f* ∧ *finite*);(*f* ∧ *inf*)) ∨
(*womega* (*f* ∧ *finite*)))
by *auto*
show *?thesis* **using** *5 6 7*
by (*metis int-eq*)
qed

lemma *WPowerstar-SChopstar*:
⊢ (*wpowerstar f*) = (*schopstar f*);(*empty* ∨ (*f* ∧ *inf*))
by (*metis SChopstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf int-eq*)

lemma *WOmega-Omega*:
⊢ (*womega f*) = ((*schopstar f*);((*init f*) ∨ (*f* ∧ *inf*)) ∨ (*omega f*))
proof –
have *1*: ⊢ (*womega f*) = ((*womega* (*f* ∧ *more*)) ∨ *wpowerstar f*;*init f*)
by (*simp add*: *womega-split-empty-more*)
have *2*: ⊢ (*womega* (*f* ∧ *more*)) = (*wpowerstar* ((*f* ∧ *more*) ∧ *finite*);((*f* ∧ *more*) ∧ *inf*) ∨
(*womega* ((*f* ∧ *more*) ∧ *finite*)))
by (*simp add*: *womega-split-finite-inf*)
have *3*: ⊢ ((*womega* (*f* ∧ *more*)) ∨ *wpowerstar f*;*init f*) =
(*wpowerstar f*;*init f* ∨
(*wpowerstar* ((*f* ∧ *more*) ∧ *finite*));((*f* ∧ *more*) ∧ *inf*) ∨
(*womega* ((*f* ∧ *more*) ∧ *finite*)))
using *1 2* **by** *fastforce*
have *4*: ⊢ *wpowerstar f*;*init f* = (*schopstar f*);((*empty* ∨ (*f* ∧ *inf*));*init f*)
by (*metis ChopAssoc WPowerstar-SChopstar inteq-reflection*)
have *5*: ⊢ ((*empty* ∨ (*f* ∧ *inf*));*init f*) = ((*init f*) ∨ (*f* ∧ *inf*))
by (*meson AndInfChopEqvAndInf EmptyOrChopEqv Prop06*)
have *6*: ⊢ (*wpowerstar* ((*f* ∧ *more*) ∧ *finite*));((*f* ∧ *more*) ∧ *inf*) = (*schopstar f*);(*f* ∧ *inf*)
by (*meson AndMoreAndInfEqvAndInf ChopEqvChop Prop04 SChopstar-WPowerstar SChopstar-and-more*)
have *7*: ⊢ (*womega* ((*f* ∧ *more*) ∧ *finite*)) = (*omega f*)
by (*meson Omega-WOmega int-iffD1 int-iffD2 int-iffI*)
have *8*: ⊢ (*wpowerstar f*;*init f* ∨
(*wpowerstar* ((*f* ∧ *more*) ∧ *finite*));((*f* ∧ *more*) ∧ *inf*) ∨
(*womega* ((*f* ∧ *more*) ∧ *finite*))) =
((*schopstar f*);((*init f*) ∨ (*f* ∧ *inf*)) ∨
(*schopstar f*);(*f* ∧ *inf*) ∨
(*omega f*))
by (*metis 3 4 5 6 7 int-eq*)
have *90*: ⊢ (*schopstar f*);((*init f*) ∨ (*f* ∧ *inf*)) =
((*schopstar f*);(*init f*) ∨ (*schopstar f*);(*f* ∧ *inf*))
by (*simp add*: *ChopOrEqv*)
have *91*: ⊢ ((*schopstar f*);((*init f*) ∨ (*f* ∧ *inf*)) ∨
(*schopstar f*);(*f* ∧ *inf*)) =
((*schopstar f*);((*init f*) ∨ (*f* ∧ *inf*)))
using *90* **by** *fastforce*
have *9*: ⊢ ((*schopstar f*);((*init f*) ∨ (*f* ∧ *inf*)) ∨
(*schopstar f*);(*f* ∧ *inf*) ∨

(*omega f*)) =
((*schopstar f*);((*init f*) ∨ (*f* ∧ *inf*)) ∨ (*omega f*))
using *91* **by** *fastforce*
show *?thesis*
by (*metis 1 3 8 9 int-eq*)
qed

lemma *womega-and-more-inf*:
⊢ (*womega* (*f* ∧ *more*)) ⟶ *inf*
using *womega-split-finite-inf*[*of LIFT* (*f* ∧ *more*)]
by (*metis* (*no-types*, *lifting*) *AndMoreAndFiniteEqvAndFmore ChopAndInf Prop02 Prop12 int-eq int-iffD1 womega-and-fmore-inf*)

lemma *womega-fmore-inf-eq*:
⊢ *womega fmore* = *inf*
by (*metis* (*no-types*, *lifting*) *ChopAndA ChopAndInf MoreAndInfEqvInf MoreEqvSkipSChopTrue SChopAssoc*
TrueChopMoreEqvMore TrueEqvTrueSChopTrue WOmega-simulation fmore-d-def int-eq int-iffI schop-d-def womega-and-inf womega-fmore-inf)

lemma *womega-skip-inf-eq*:
⊢ *womega skip* = *inf*
by (*metis WOmega-Wagner-1 int-eq womega-fmore-inf-eq wpowerstar-skip-fmore*)

lemma *womega-more-inf*:
⊢ *womega more* ⟶ *inf*
by (*metis MoreAndInfEqvInf Prop02 WOmega-star-1 fmore-d-def inteq-reflection womega-fmore-inf-eq womega-skip-inf womega-skip-inf-eq womega-split-finite-inf*)

lemma *womega-more-inf-eq*:
⊢ *womega more* = *inf*
by (*metis OrFiniteInf WOmega-subdist fmore-d-def int-eq int-iffI womega-fmore-inf-eq womega-more-inf*)

lemma *womega-true*:
⊢ *womega #True*
by (*meson MP TrueEqvTrueChopTrue TrueW WOmega-coinduct-eq-var2*)

9.3 Properties of Omega

lemma *NotOmegaInf*:
⊢ ¬(*omega* (*inf*))
proof −
have *1*: ⊢ *omega* (*inf*) = ((*inf* ∧ *more*) ∧ *finite*);*omega inf*
by (*simp add*: *OmegaUnroll*)
have *2*: ⊢ ((*inf* ∧ *more*) ∧ *finite*) = *#False*
unfolding *finite-d-def* **by** *auto*
have *3*: ⊢ *#False*;*omega inf* = *#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 *InfImpOmegaLenPlusOne*:
$\vdash$ *inf* $\longrightarrow$ *omega* (*len*(*Suc n*))
proof –
have *1*: $\vdash$ *inf* $\longrightarrow$ ((*len Suc n* $\wedge$ *more*) $\wedge$ *finite*);*inf*
by (*auto simp add*: *Valid-def itl-defs len-defs nfinite-conv-nlength-enat zero-enat-def*)
show *?thesis*
using *OmegaWeakCoinduct*[*of LIFT inf LIFT len* (*Suc n*)] *1* **by** *blast*
qed

lemma *OmegaLenPlusOneEqvInf*:
$\vdash$ *omega* (*len*(*Suc n*)) = *inf*
by (*simp add*: *InfImpOmegaLenPlusOne int-iffI omega-imp-inf*)

lemma *OmegaSkipEqvInf*:
$\vdash$ *omega skip* = *inf*
proof –
have *1*: $\vdash$ *skip* = (*len 1*)
by (*simp add*: *len-d-def wpower-d-def*)
(*metis ChopEmpty inteq-reflection*)
have *2*: $\vdash$ *omega skip* = *omega* (*len 1*)
using *1* **by** (*metis OmegaUnroll inteq-reflection*)
from *2* **show** *?thesis* **using** *OmegaLenPlusOneEqvInf* **by** *fastforce*
qed

lemma *InfImpOmegaTrue*:
$\vdash$ *inf* $\longrightarrow$ *omega* #*True*
proof –
have *1*: $\vdash$ *inf* $\longrightarrow$ ((#*True* $\wedge$ *more*) $\wedge$ *finite*);*inf*
by (*auto simp add*: *Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)
show *?thesis*
using *OmegaWeakCoinduct*[*of LIFT inf LIFT* #*True*] *1* **by** *blast*
qed

lemma *OmegaTrueEqvInf*:
$\vdash$ *omega* #*True* = *inf*
by (*simp add*: *InfImpOmegaTrue int-iffI omega-imp-inf*)

lemma *InfImpOmegaMore*:
$\vdash$ *inf* $\longrightarrow$ *omega more*
using *OmegaWeakCoinduct*[*of LIFT inf LIFT more*]
proof –
have *1*: $\vdash$ *inf* $\longrightarrow$ ((*more* $\wedge$ *more*) $\wedge$ *finite*);*inf*
by (*auto simp add*: *Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)
show *?thesis* **using** *OmegaWeakCoinduct*[*of LIFT inf LIFT more*] *1* **by** *blast*

qed

lemma *OmegaMoreEqvInf*:
⊢ *omega more* = *inf*
by (*simp add*: *InfImpOmegaMore int-iffI omega-imp-inf*)

lemma *InfImpOmegaFinite*:
⊢ *inf* ⟶ *omega finite*
proof –
have *1*: ⊢ *inf* ⟶ ((*finite* ∧ *more*) ∧ *finite*);*inf*
by (*auto simp add*: *Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)
show *?thesis* **using** *OmegaWeakCoinduct*[*of LIFT inf LIFT finite*] *1* **by** *blast*
qed

lemma *OmegaFiniteEqvInf*:
⊢ *omega finite* = *inf*
by (*simp add*: *InfImpOmegaFinite int-iffI omega-imp-inf*)

lemma *BoxStateAndInfImpWOmegaBoxState*:
⊢ □(*init w*) ⟶ *womega* (□(*init w*))
using *WOmegaWeakCoinduct*[*of LIFT* (□(*init w*)) *LIFT* □(*init w*)]
by (*simp add*: *BoxStateChopBoxEqvBox int-iffD2*)

lemma *BoxStateAndInfImpOmegaBoxState*:
⊢ □(*init w*) ∧ *inf* ⟶ *omega* (□(*init w*))
proof –
have *1*: ⊢ ((□ (*init w*) ∧ *more*) ∧ *finite*);(□ (*init w*) ∧ *inf*) =
(□ (*init w*) ∧ (*more* ∧ *finite*);*inf*)
using *BoxStateAndChopEqvChop*[*of w LIFT* (*more* ∧ *finite*) *LIFT inf*]
by (*metis AndMoreAndFiniteEqvAndFmore fmore-d-def int-eq*)
have *2*: ⊢ (*more* ∧ *finite*);*inf* = *inf*
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite*
OmegaFiniteEqvInf OmegaUnroll Prop10 SkipChopFiniteImpFinite fmore-d-def int-eq)
have *3*: ⊢ □ (*init w*) ∧ *inf* ⟶ ((□ (*init w*) ∧ *more*) ∧ *finite*);(□ (*init w*) ∧ *inf*)
by (*metis 1 2 Prop11 int-eq*)
show *?thesis* **using** *OmegaWeakCoinduct*[*of LIFT* (□(*init w*) ∧ *inf*) *LIFT* □(*init w*)] *3* **by** *blast*
qed

lemma *WOmegaBoxStateAndMoreImpBoxState*:
⊢ *womega* (□(*init w*) ∧ *more*) ⟶ □(*init w*)
proof –
have *1*: ⊢ *womega* (□(*init w*) ∧ *more*) ⟶ *init w*
by (*metis AndChopA AndChopB BoxWPowerstarEqvBox StateChopExportA WOmegaUnroll WOmega-star-1*
WPowerstar-more-absorb int-eq int-iffI)
have *2*: ⊢ *womega* (□(*init w*) ∧ *more*) ⟶ (□(*init w*) ∧ *more*); *womega* (□(*init w*) ∧ *more*)

by (*simp add*: *WOmegaUnroll int-iffD1*)
have *21*: ⊢ ((□(*init w*) ∧ *more*)) ⟶ ○(□(*init w*))
by (*metis BoxImpNowAndWeakNext Prop01 Prop12 WnextEqvEmptyOrNext empty-d-def inteq-reflection*

lift-and-com)
have *22*: ⊢ *womega* (□(*init w*) ∧ *more*) ⟶ *more*
by (*simp add*: *WOmega-and-more-imp-more*)
have *23*: ⊢ ○(□(*init w*)) ⟶ ○ (*wpowerstar* (□ (*init w*)))
by (*simp add*: *NextImpNext WPowerstar-ext*)
have *24*: ⊢ *womega* (□(*init w*) ∧ *more*) ⟶ (○(*wpowerstar* (□ (*init w*) ∧ *more*))); (*womega* (□(*init w*) ∧ *more*))
by (*metis 21 23 LeftChopImpChop WOmegaUnroll WPowerstar-more-absorb int-eq lift-imp-trans*)
have *10*: ⊢ *more* ∧ *womega* (□(*init w*) ∧ *more*) ⟶ ○ (*womega* (□(*init w*) ∧ *more*))
using *WOmega-star-1*[*of LIFT* (□(*init w*) ∧ *more*)]
by (*metis 22 24 NextChop Prop10 inteq-reflection lift-and-com*)
show *?thesis* **using** *BoxIntro*[*of LIFT womega* (□(*init w*) ∧ *more*) *LIFT* (*init w*)]
using *1 10* **by** *blast*
qed

lemma *OmegaBoxStateImpBoxState*:
⊢ *omega* (□(*init w*)) ⟶ □(*init w*) ∧ *inf*
proof −
have *1*: ⊢ *omega* (□(*init w*)) ⟶ *init w*
by (*metis AndMoreAndFiniteEqvAndFmore BoxElim OmegaUnroll Omega-iso StateChopExportA inteq-reflection*
lift-imp-trans)
have *2*: ⊢ *omega* (□(*init w*)) ⟶ ((□(*init w*) ∧ *more*) ∧ *finite*);(*omega* (□(*init w*)))
by (*simp add*: *OmegaUnroll int-iffD1*)
have *21*: ⊢ ((□(*init w*) ∧ *more*) ∧ *finite*) ⟶ ○(□(*init w*))
by (*metis BoxImpNowAndWeakNext Prop01 Prop05 Prop12 WnextEqvEmptyOrNext empty-d-def*
inteq-reflection lift-and-com)
have *22*: ⊢ *finite* = (*empty* ∨ *fmore*)
by (*metis EmptyOrMoreSplit FiniteAndEmptyEqvEmpty fmore-d-def inteq-reflection lift-and-com*)
have *23*: ⊢ (□ (*init w*) ∧ *finite*) = ((□ (*init w*) ∧ *empty*) ∨ (□ (*init w*) ∧ *fmore*))
using *22* **by** *fastforce*
have *24*: ⊢ (□ (*init w*) ∧ *empty*) = (*init w* ∧ *empty*)
using *BoxEqvAndBox StateAndEmptyImpBoxState* **by** *fastforce*
have *25*: ⊢ ○(□(*init w*)) ∧ *fmore* ⟶ ○(((*init w*) ∧ *empty*) ∨ (□ (*init w*) ∧ *fmore*))
using *23 24* **by** (*metis FmoreEqvSkipChopFinite NextAndEqvNextAndNext SkipChopEqvNext int-iffD2*
inteq-reflection)
have *26*: ⋀ *g*. ⊢ (○(((*init w*) ∧ *empty*) ∨ (□ (*init w*) ∧ *fmore*)));*g* =
(○(*init w* ∧ *g*) ∨ ○((□(*init w*) ∧ *fmore*);*g*))
proof −
fix *g* :: ′*a nellist* ⇒ *bool*
have *f1*: ⊢○ (*init w* ∧ *empty* ∨ □ (*init w*) ∧ *fmore*);*g* =
○ ((*init w* ∧ *empty* ∨ □ (*init w*) ∧ *fmore*);*g*)
by (*meson NextChop int-eq*)
have ⊢ *skip*;((*init w* ∧ *empty* ∨ □ (*init w*) ∧ *fmore*);*g*) =
(*skip*;(*init w* ∧ *g*) ∨ *skip*;((□ (*init w*) ∧ *fmore*);*g*))
by (*metis ChopOrEqvRule OrChopEqv StateAndEmptyChop int-eq*)

then show $\vdash \bigcirc$ (*init w* $\wedge$ *empty* $\vee$ $\Box$ (*init w*) $\wedge$ *fmore*);*g* =
(○ (*init w* $\wedge$ *g*) $\vee$ ○ (($\Box$ (*init w*) $\wedge$ *fmore*);*g*))
by (*metis* (*no-types*, *opaque-lifting*) *ChopAssoc Prop04 next-d-def*)
qed
have *27*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *fmore*) = ($\Box$ (*init w*) $\wedge$ *skip*);($\Box$ (*init w*) $\wedge$ *finite*)
by (*metis BoxStateAndChopEqvChop FmoreEqvSkipChopFinite inteq-reflection*)
have *28*: $\vdash$(*init w* $\wedge$ *empty* $\vee$ $\Box$ (*init w*) $\wedge$ *fmore*) = ($\Box$ (*init w*) $\wedge$ *finite*)
by (*metis 23 24 int-eq*)
have *29*: $\vdash$ (*skip*;($\Box$ (*init w*) $\wedge$ *finite*));*omega* ($\Box$ (*init w*)) =
(*skip*;(*init w* $\wedge$ *omega* ($\Box$ (*init w*))) $\vee$ *skip*;(($\Box$ (*init w*) $\wedge$ *fmore*);*omega* ($\Box$ (*init w*))))
by (*metis 26 28 SkipChopEqvNext int-eq*)
have *3*: $\vdash$ ($\Box$(*init w*) $\wedge$ *fmore*);(*omega* ($\Box$(*init w*))) $\longrightarrow$
(○(*init w* $\wedge$ *omega* ($\Box$(*init w*))) $\vee$ ○(($\Box$(*init w*) $\wedge$ *fmore*);(*omega* ($\Box$(*init w*)))))
by (*metis 27 29 AndChopB LeftChopImpChop inteq-reflection next-d-def*)
have *4*: $\vdash$ (○(*init w* $\wedge$ *omega* ($\Box$(*init w*))) $\vee$ ○(($\Box$(*init w*) $\wedge$ *fmore*);(*omega* ($\Box$(*init w*))))) $\longrightarrow$
○(*omega* ($\Box$(*init w*)))
by (*metis 1 AndMoreAndFiniteEqvAndFmore ChopAndB OmegaUnroll Prop02 Prop10 int-eq lift-and-com next-d-def*)
have *5*: $\vdash$ *omega* ($\Box$(*init w*)) $\longrightarrow$ ○(*omega* ($\Box$(*init w*)))
by (*metis 3 4 AndMoreAndFiniteEqvAndFmore OmegaUnroll inteq-reflection lift-imp-trans*)
from *1 5* **show** *?thesis* **using** *BoxIntro*
by (*metis Prop01 Prop05 Prop12 omega-imp-inf*)
qed

lemma *OmegaIntro*:
assumes $\vdash$ *h* $\longrightarrow$ (*f* $\wedge$ *fmore*);*h*
shows $\vdash$ *h* $\longrightarrow$ *omega f*
using *assms*
by (*metis AndMoreAndFiniteEqvAndFmore Omega-coinduct-var2 int-eq schop-d-def*)

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

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

lemma *AndWOmegaA*:
$\vdash$ *womega* (*f* $\wedge$ *g*) $\longrightarrow$ *womega f*
by (*meson WOmega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndOmegaA*:
$\vdash$ *omega* (*f* $\wedge$ *g*) $\longrightarrow$ *omega f*
by (*meson Omega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndWOmegaB*:

$\vdash$ *womega* $(f \wedge g) \longrightarrow$ *womega g*
by (*meson WOmega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndOmegaB*:
$\vdash$ *omega* $(f \wedge g) \longrightarrow$ *omega g*
by (*meson Omega-iso Prop12 int-iffD2 lift-and-com*)

lemma *BaWOmegaImpWOmega*:
$\vdash$ *ba* $(f \longrightarrow g) \longrightarrow$ *womega f* $\longrightarrow$ *womega g*
proof –
 have *1*: $\vdash$ *ba* $(f \longrightarrow g) \wedge (f)$;*womega f* $\longrightarrow((f \longrightarrow g) \wedge (f)\ )$;$((f \longrightarrow g) \wedge$ *womega f*$)$
  using *BaAndChopImport*[*of LIFT* $(f \longrightarrow g)$ *LIFT* (f) *LIFT womega f*]
   by *blast*
 have *2*: $\vdash (f \longrightarrow g) \wedge (f) \longrightarrow (\ g)$
  by *auto*
 have *3*: $\vdash (f \longrightarrow g) \wedge$ *womega f* $\longrightarrow$ *womega f*
  by *auto*
 have *4*: $\vdash$ *ba* $(f \longrightarrow g) \wedge (f)$;*womega f* $\longrightarrow (g)$; *womega f*
  using *1 2 3*
  by (*metis* (*no-types*, *lifting*) *AndChopB ChopAndB Prop10 int-eq lift-imp-trans*)
 have *5*: $\vdash$ *ba* $(f \longrightarrow g) \wedge (f)$;*womega f* $\longrightarrow ((f \longrightarrow g) \wedge (f))$;(*ba*$(f \longrightarrow g) \wedge$ *womega f*$)$
  using *BaAndChopImportB* **by** *blast*
 have *6*: $\vdash ((f \longrightarrow g) \wedge (f))$;(*ba*$(f \longrightarrow g) \wedge$ *womega f*$) \longrightarrow$
      (g);(*ba*$(f \longrightarrow g) \wedge$ *womega f*$)$
  using *2 LeftChopImpChop* **by** *blast*
 have *61*: $\vdash$ *womega f* $= (f)$;*womega f*
   by (*simp add*: *WOmegaUnroll*)
 have *62*: $\vdash$ (*ba* $(f \longrightarrow g) \wedge$ *womega f*$) \longrightarrow$ (*ba* $(f \longrightarrow g) \wedge (f)$;*womega f*$)$
   by (*metis 61 ChopEmpty Prop11 int-eq*)
 have *7*: $\vdash$ (*ba* $(f \longrightarrow g) \wedge$ *womega f*$) \longrightarrow(g)$;(*ba* $(f \longrightarrow g) \wedge$ *womega f*$)$
  using *62 5 6*
  by (*meson lift-imp-trans*)
 have *8*: $\vdash$ (*ba* $(f \longrightarrow g) \wedge$ *womega f*$)$ $\longrightarrow$ *womega g*
  using *7*
  using *WOmegaWeakCoinduct* **by** *blast*
 from *8* **show** *?thesis* **by** *fastforce*
qed

lemma *BaOmegaImpOmega*:
$\vdash$ *ba* $(f \longrightarrow g) \longrightarrow$ *omega f* $\longrightarrow$ *omega g*
proof –
 have *1*: $\vdash$ *ba* $(f \longrightarrow g) \wedge (f \wedge$ *fmore*$)$;*omega f* $\longrightarrow((f \longrightarrow g) \wedge (f \wedge$ *fmore*$)\ )$;$((f \longrightarrow g) \wedge$ *omega f*$)$
  using *BaAndChopImport*[*of LIFT* $(f \longrightarrow g)$ *LIFT* $(f \wedge$ *fmore*$)$ *LIFT omega f*]
   by *blast*
 have *2*: $\vdash (f \longrightarrow g) \wedge (f \wedge$ *fmore*$) \longrightarrow (\ g \wedge$ *fmore*$)$
  by *auto*
 have *3*: $\vdash (f \longrightarrow g) \wedge$ *omega f* $\longrightarrow$ *omega f*
  by *auto*

have *4*: $\vdash$ *ba* ($f \longrightarrow g$) $\wedge$ ($f \wedge$ *fmore*);*omega f* $\longrightarrow$ ($g \wedge$ *fmore*); *omega f*
  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*);*omega f* $\longrightarrow$ (($f \longrightarrow g$) $\wedge$ ($f \wedge$ *fmore*));(*ba*($f \longrightarrow g$) $\wedge$ *omega f*)
  using *BaAndChopImportB* **by** *blast*
have *6*: $\vdash$ (($f \longrightarrow g$) $\wedge$ ($f \wedge$ *fmore*));(*ba*($f \longrightarrow g$) $\wedge$ *omega f*) $\longrightarrow$
    ($g \wedge$ *fmore*);(*ba*($f \longrightarrow g$) $\wedge$ *omega f*)
  using *2 LeftChopImpChop* **by** *blast*
have *61*: $\vdash$ *omega f* = ($f \wedge$ *fmore*);*omega f*
  by (*metis AndMoreAndFiniteEqvAndFmore OmegaUnroll inteq-reflection*)
have *62*: $\vdash$ (*ba* ($f \longrightarrow g$) $\wedge$ *omega f*) $\longrightarrow$ (*ba* ($f \longrightarrow g$) $\wedge$ ($f \wedge$ *fmore*);*omega f*)
   by (*metis 61 ChopEmpty Prop11 int-eq*)
have *7*: $\vdash$ (*ba* ($f \longrightarrow g$) $\wedge$ *omega f*) $\longrightarrow$($g \wedge$ *fmore*);(*ba* ($f \longrightarrow g$) $\wedge$ *omega f*)
  using *62 5 6*
  by (*meson lift-imp-trans*)
have *8*: $\vdash$ (*ba* ($f \longrightarrow g$) $\wedge$ *omega f*) $\longrightarrow$ *omega g*
  using *7 OmegaIntro* **by** *blast*
from *8* **show** *?thesis* **by** *fastforce*
qed

lemma *BaWOmegaEqvWOmega*:
$\vdash$ *ba* ($f = g$) $\longrightarrow$ (*womega f* = *womega g*)
proof –
have *0*: $\vdash$ ($f = g$) = (($f \longrightarrow g$) $\wedge$ ($g \longrightarrow f$)) **by** *fastforce*
have *1*: $\vdash$ *ba* ($f = g$) = (*ba* ($f \longrightarrow g$) $\wedge$ *ba* ($g \longrightarrow f$)) **by** (*metis 0 BaAndEqv int-eq*)
have *2*: $\vdash$ *ba* ($f \longrightarrow g$) $\longrightarrow$ (*womega f* $\longrightarrow$ *womega g*) **using** *BaWOmegaImpWOmega* **by** *blast*
have *3*: $\vdash$ *ba* ($g \longrightarrow f$) $\longrightarrow$ (*womega g* $\longrightarrow$ *womega f*) **using** *BaWOmegaImpWOmega* **by** *blast*
have *4*: $\vdash$ *ba* ($f = g$) $\longrightarrow$ (*womega f* $\longrightarrow$ *womega g*) $\wedge$ (*womega g* $\longrightarrow$ *womega f*) **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash$ ((*womega f* $\longrightarrow$ *womega g*) $\wedge$ (*womega g* $\longrightarrow$ *womega f*)) = (*womega f* = *womega g*) **by** *auto*
from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *BaOmegaEqvOmega*:
$\vdash$ *ba* ($f = g$) $\longrightarrow$ (*omega f* = *omega g*)
proof –
have *0*: $\vdash$ ($f = g$) = (($f \longrightarrow g$) $\wedge$ ($g \longrightarrow f$)) **by** *fastforce*
have *1*: $\vdash$ *ba* ($f = g$) = (*ba* ($f \longrightarrow g$) $\wedge$ *ba* ($g \longrightarrow f$)) **by** (*metis 0 BaAndEqv int-eq*)
have *2*: $\vdash$ *ba* ($f \longrightarrow g$) $\longrightarrow$ (*omega f* $\longrightarrow$ *omega g*) **using** *BaOmegaImpOmega* **by** *blast*
have *3*: $\vdash$ *ba* ($g \longrightarrow f$) $\longrightarrow$ (*omega g* $\longrightarrow$ *omega f*) **using** *BaOmegaImpOmega* **by** *blast*
have *4*: $\vdash$ *ba* ($f = g$) $\longrightarrow$ (*omega f* $\longrightarrow$ *omega g*) $\wedge$ (*omega g* $\longrightarrow$ *omega f*) **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash$ ((*omega f* $\longrightarrow$ *omega g*) $\wedge$ (*omega g* $\longrightarrow$ *omega f*)) = (*omega f* = *omega g*) **by** *auto*
from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *BaAndWOmegaImport*:
$\vdash$ *ba f* $\wedge$ *womega g* $\longrightarrow$ *womega* ($f \wedge g$)
proof –
have *1*: $\vdash$ $f \longrightarrow$ ($g \longrightarrow$ ($f \wedge g$)) **by** *auto*
hence *2*: $\vdash$ *ba f* $\longrightarrow$ *ba* ($g \longrightarrow f \wedge g$) **by** (*rule BaImpBa*)

have *3*: $\vdash$ *ba* $(g \longrightarrow f \wedge g) \longrightarrow$ *womega* $g \longrightarrow$ *womega* $(f \wedge g)$ **by** (*rule BaWOmegaImpWOmega*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndOmegaImport*:
$\vdash$ *ba* $f \wedge$ *omega* $g \longrightarrow$ *omega* $(f \wedge g)$
proof –
have *1*: $\vdash f \longrightarrow (g \longrightarrow (f \wedge g))$ **by** *auto*
hence *2*: $\vdash$ *ba* $f \longrightarrow$ *ba* $(g \longrightarrow f \wedge g)$ **by** (*rule BaImpBa*)
have *3*: $\vdash$ *ba* $(g \longrightarrow f \wedge g) \longrightarrow$ *omega* $g \longrightarrow$ *omega* $(f \wedge g)$ **by** (*rule BaOmegaImpOmega*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *InfAndChop*:
$\vdash$ (*inf* $\wedge$ ($f \wedge$ *fmore*);g) = ($f \wedge$ *fmore*);($g \wedge$ *inf*)
by (*meson ChopAndInf Prop04 lift-and-com*)

lemma *InfImportBox*:
$\vdash$ (*inf* $\wedge \Box f$) = $\Box$ (*inf* $\wedge f$)
by (*metis BoxAndBoxEqvBoxRule FiniteChopEqvDiamond FiniteChopFiniteEqvFinite InfEqvNotFinite NotDiamondNotEqvBox OrFiniteInf finite-d-def int-eq*)

lemma *InfImportBoxImp*:
$\vdash$ (*inf* $\wedge \Box (f \longrightarrow g)$) = $\Box$ ((*inf* $\longrightarrow f$) $\longrightarrow$ (*inf* $\wedge g$))
proof –
have *1*: $\vdash$ (*inf* $\wedge (f \longrightarrow g)$) = ((*inf* $\longrightarrow f$) $\longrightarrow$ (*inf* $\wedge g$))
by *auto*
show *?thesis*
by (*metis 1 InfImportBox inteq-reflection*)
qed

lemma *OmegaImpAOmega*:
$\vdash$ *omega* $f \longrightarrow$ *aomega* f
using *AOmegaWeakCoInduct*[*of LIFT omega f f*]
by (*metis AndMoreAndFiniteEqvAndFmore InfAndChop OmegaIntros OmegaUnroll Omega-fmore-absorb Prop10*
int-eq omega-and-fmore-inf)

lemma *AOmegaImpOmega*:
$\vdash$ *aomega* $f \longrightarrow$ *omega* f
using *OmegaWeakCoinduct*[*of LIFT aomega f f*]
by (*simp add*: *AOmegaUnroll OmegaIntro int-iffD1*)

lemma *OmegaEqvAOmega*:
$\vdash$ *omega* f = *aomega* f

using *OmegaImpAOmega AOmegaImpOmega*
by (*metis int-iffI*)

end

10 Projection operator

theory *Projection*
imports *Omega*
begin

This theory introduces the projection operator fproj [5]. The projection operator is defined and we prove the soundness of the rules and axiom system. We also provide the infinite projection operator oproj which was defined in [7].

10.1 Definitions

primcorec *pfilt* :: *'a intervals* ⇒ *nat nellist* ⇒ *'a intervals*
where
pfilt σ *ls* = (*case ls of* (*NNil n*) ⇒ (*NNil* (*nnth* σ *n*)) |
(*NCons n ls1*) ⇒ (*NCons* (*nnth* σ *n*) (*pfilt* σ *ls1*)))

simps-of-case *pfilt-code* [*code*, *simp*, *nitpick-simp*]: *pfilt.code*

primcorec *lsum* :: *'a intervals intervals* ⇒ *nat* ⇒ *nat nellist*
where
lsum nells a = (*case nells of* (*NNil nell*) ⇒ (*NNil* (*a*+(*the-enat* (*nlength nell*)))) |
(*NCons nell nells1*) ⇒ (*NCons* (*a*+(*the-enat* (*nlength nell*)))
(*lsum nells1* (*a*+(*the-enat* (*nlength nell*))))))

simps-of-case *lsum-code* [*code*, *simp*, *nitpick-simp*]: *lsum.code*

definition *addzero* :: *nat nellist* ⇒ *nat nellist*
where *addzero ls* = (*if nlength ls* = *0 then*
(*if nfirst ls* = *0 then ls else* (*NCons 0 ls*)) *else* (*NCons 0 ls*))

definition *powerinterval* :: (*'a::world*) *formula* ⇒ *'a intervals* ⇒ *nat nellist* ⇒ *bool*
where *powerinterval F* σ *l* =
($\forall$ (*i::nat*). (*enat i*)<*nlength l* ⟶ ((*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) ⊨ *F*))

definition *cppl* :: (*'a::world*) *formula* ⇒ *'a formula* ⇒ *'a intervals* ⇒ *nat nellist*
where *cppl f g* σ = (ϵ *ls*. *nidx ls* ∧ (*nnth ls 0*) = *0* ∧ *powerinterval f* σ *ls* ∧
((*nfinite ls* ∧ *nfinite* σ ∧ (*nlast ls*) = *the-enat*(*nlength* σ)) ∨
(¬ *nfinite ls* ∧ ¬*nfinite* σ)) ∧
((*pfilt* σ *ls*) ⊨ *g*))

primcorec *lcppl* :: (*'a ::world*) *formula* ⇒ *'a formula* ⇒ *'a intervals* ⇒ *nat nellist* ⇒ *nat intervals intervals*
where *lcppl f g* σ *ls* =
(*case ls of* (*NNil x*) ⇒ (*NNil* (*nmap* (λ*l*. *l*+*x*) (*cppl f g* (*nsubn* σ *x x*)))) |

$(NCons\ x\ ls1) \Rightarrow$
$(case\ ls1\ of\ (NNil\ y) \Rightarrow (NNil\ (nmap\ (\lambda l.\ l+x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ y)))) \mid$
$(NCons\ y\ ls2) \Rightarrow (NCons\ (nmap\ (\lambda l.\ l+x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ y)))\ (lcppl\ f\ g\ \sigma\ ls1))))$

simps-of-case *lcppl-code* [*code*, *simp*, *nitpick-simp*]: *lcppl.code*

definition *fprojection-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where $fprojection\text{-}d\ F\ G \equiv \lambda\sigma.\ (\exists\ ls.\ \ nidx\ ls\ \wedge (nnth\ ls\ 0) = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma\ \wedge$
$(nlast\ ls) = the\text{-}enat(nlength\ \sigma) \wedge$
$powerinterval\ F\ \sigma\ ls \wedge ((pfilt\ \sigma\ ls) \models G)$
$)$

definition *oprojection-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where $oprojection\text{-}d\ F\ G \equiv \lambda\sigma.\ (\exists\ ls.\ \ nidx\ ls\ \wedge (nnth\ ls\ 0) = 0 \wedge \neg nfinite\ ls \wedge \neg\ nfinite\ \sigma\ \wedge$
$powerinterval\ F\ \sigma\ ls \wedge ((pfilt\ \sigma\ ls) \models G)$
$)$

syntax
-fprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *fproj* -) [*84*,*84*] *83*)
-oprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *oproj* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-fprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *fproj* -) [*84*,*84*] *83*)
-oprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *oproj* -) [*84*,*84*] *83*)

translations
-fprojection-d ⇌ *CONST fprojection-d*
-oprojection-d ⇌ *CONST oprojection-d*

definition *ufprojection-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where $ufprojection\text{-}d\ F\ G \equiv LIFT(\neg(F\ fproj\ (\neg\ G)))$

definition *uoprojection-d* :: ('*a*:: *world*) *formula* ⇒ '*a formula* ⇒ '*a formula*
where $uoprojection\text{-}d\ F\ G \equiv LIFT(\neg(F\ oproj\ (\neg\ G)))$

syntax
-ufprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *ufproj* -) [*84*,*84*] *83*)
syntax
-uoprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *uoproj* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-ufprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *ufproj* -) [*84*,*84*] *83*)
-uoprojection-d :: [*lift*,*lift*] ⇒ *lift* ((- *uoproj* -) [*84*,*84*] *83*)

translations
-ufprojection-d ⇌ *CONST ufprojection-d*

-uoprojection-d ⇌ *CONST uoprojection-d*

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

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

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

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

syntax
-fdp-d :: *lift* ⇒ *lift* ((*fdp* -) [*88*] *87*)
-fbp-d :: *lift* ⇒ *lift* ((*fbp* -) [*88*] *87*)
-odp-d :: *lift* ⇒ *lift* ((*odp* -) [*88*] *87*)
-obp-d :: *lift* ⇒ *lift* ((*obp* -) [*88*] *87*)

syntax (*ASCII*)
-fdp-d :: *lift* ⇒ *lift* ((*fdp* -) [*88*] *87*)
-fbp-d :: *lift* ⇒ *lift* ((*fbp* -) [*88*] *87*)
-odp-d :: *lift* ⇒ *lift* ((*odp* -) [*88*] *87*)
-obp-d :: *lift* ⇒ *lift* ((*obp* -) [*88*] *87*)

translations
-fdp-d ⇌ *CONST fdp-d*
-fbp-d ⇌ *CONST fbp-d*
-odp-d ⇌ *CONST odp-d*
-obp-d ⇌ *CONST obp-d*

abbreviation *all-nfinite nells* ≡ (∀ *nell* ∈ *nset nells. nfinite nell*)

abbreviation *all-gr-zero nells* ≡ (∀ *nell* ∈ *nset nells. 0*<*nlength nell*)

10.2 Lemmas

10.2.1 Misc

lemma *all-nfinite-nnth-a*:
assumes ∀ *i* ≤*nlength nells. nfinite* (*nnth nells i*)
shows *all-nfinite nells*
using *assms*
by (*metis in-nset-conv-nnth*)

lemma *all-nfinite-nnth-b*:
assumes *all-nfinite nells*
shows ∀ *i* ≤*nlength nells. nfinite* (*nnth nells i*)
using *assms*

by (*meson in-nset-conv-nnth*)

lemma *all-gr-zero-nnth-a*:
assumes $\forall\, i \leq$*nlength nells*. *0* $<$ *nlength* (*nnth nells i*)
shows *all-gr-zero nells*
using *assms*
by (*metis in-nset-conv-nnth*)

lemma *all-gr-zero-nnth-b*:
assumes *all-gr-zero nells*
shows $\forall\, i \leq$*nlength nells*. *0* $<$ *nlength* (*nnth nells i*)
using *assms*
by (*meson in-nset-conv-nnth*)

lemma *all-nfinite-nnth-ntaken*:
assumes *all-nfinite nells*
$n \leq$ *nlength nells*
shows *all-nfinite* (*ntaken n nells*)
using *assms*
by (*meson nset-ntaken subsetD*)

lemma *all-nfinite-nnth-ndropn*:
assumes *all-nfinite nells*
$n \leq$ *nlength nells*
shows *all-nfinite* (*ndropn n nells*)
using *assms*
by (*metis nset-ndropn subsetD*)

lemma *all-nfinite-nnth-nsubn*:
assumes *all-nfinite nells*
$n \leq$ *nlength nells*
$k \leq n$
shows *all-nfinite* (*nsubn nells k n*)
using *assms*
by (*metis in-mono nset-ndropn nset-ntaken nsubn-def1*)

lemma *all-gr-zero-nnth-ntaken*:
assumes *all-gr-zero nells*
$n \leq$ *nlength nells*
shows *all-gr-zero* (*ntaken n nells*)
using *assms*
by (*meson nset-ntaken subsetD*)

lemma *all-gr-zero-nnth-ndropn*:
assumes *all-gr-zero nells*
$n \leq$ *nlength nells*
shows *all-gr-zero* (*ndropn n nells*)
using *assms*
by (*metis nset-ndropn subsetD*)

lemma *all-gr-zero-nnth-nsubn*:
assumes *all-gr-zero nells*
$n \leq$ *nlength nells*
$k \leq n$
shows *all-gr-zero* (*nsubn nells k n*)
using *assms*
by (*metis in-mono nset-ndropn nset-ntaken nsubn-def1*)

10.2.2 pfilt Lemmas

lemma *pfilt-nmap*:
pfilt nell ls = *nmap* ($\lambda x.$ (*nnth nell x*)) *ls*
proof (*coinduction arbitrary*: *nell ls*)
case (*Eq-nellist nell1*)
then show *?case* **by** *simp* (*metis nellist.case-eq-if*)
qed

lemma *pfilt-nlength*:
nlength(*pfilt nell ls*) = *nlength ls*
by (*simp add*: *pfilt-nmap*)

lemma *pfilt-nnth*:
assumes $i \leq$ *nlength* (*pfilt nell ls*)
shows (*nnth* (*pfilt nell ls*) *i*) = (*nnth nell* (*nnth ls i*))
using *assms*
by (*simp add*: *pfilt-nmap*)

lemma *pfilt-expand*:
(*nell1* = (*pfilt nell ls*)) =
(*nlength nell1* = *nlength ls* $\wedge$
($\forall$ (*i*::*nat*). $i \leq$ *nlength nell1* $\longrightarrow$ (*nnth nell1 i*) = (*nnth nell* (*nnth ls i*))))
by (*metis nellist-eq-nnth-eq pfilt-nlength pfilt-nnth*)

lemma *pfilt-nfuse*:
pfilt nell (*nfuse ls1 ls2*) = (*nfuse* (*pfilt nell ls1*) (*pfilt nell ls2*))
using *pfilt-nmap*[*of nell* (*nfuse ls1 ls2*)] *pfilt-nmap*[*of nell ls1*] *pfilt-nmap*[*of nell ls2*]
by (*simp add*: *nmap-nfuse*)

lemma *nfuse-pfilt-nlength*:
shows *nlength* (*pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls*) =
nlength (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* ($\lambda x.$ $x-$ (*nnth ls n*)) (*ndropn n ls*))))
by (*metis add.right-neutral linorder-le-cases ndropn-all nfuse-nlength nfuse-ntaken-ndropn-nlength nlength-NNil nlength-nmap ntaken-all pfilt-nlength*)

lemma *nfuse-pfilt-nnth-a*:
assumes *nidx ls*

(*nnth ls 0*) = *0*
(*nlast ls*) = *the-enat*(*nlength nell*)
nfinite nell
nfinite ls
i ≤ *nlength ls*
n ≤ *nlength ls*
shows *nnth* (*pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls*) *i* =
nnth nell (*nnth ls i*)
using *assms*
by (*metis linorder-le-cases ndropn-all nfinite-conv-nlength-enat nfuse-ntaken-ndropn ntaken-all pfilt-nlength pfilt-nnth*)

lemma *nfuse-pfilt-nnth-a-alt*:
assumes *nidx ls*
(*nnth ls 0*) = *0*
¬*nfinite nell*
¬*nfinite ls*
i ≤ *nlength ls*
n ≤ *nlength ls*
shows *nnth* (*pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls*) *i* =
nnth nell (*nnth ls i*)
using *assms*
by (*metis linorder-le-cases nfinite-ntaken nfuse-ntaken-ndropn ntaken-all pfilt-nlength pfilt-nnth*)

lemma *nfuse-pfilt-nnth-b*:
assumes *nidx ls*
nnth ls 0 = *0*
(*nlast ls*) = *the-enat*(*nlength nell*)
nfinite nell
nfinite ls
i ≤ *nlength ls*
n ≤ *nlength ls*
shows *nnth* (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x* − (*nnth ls n*)) (*ndropn n ls*)))) *i* =
nnth nell (*nnth ls i*)
proof −
have *1*: *i* ≤ *nlength*(*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x* − (*nnth ls n*)) (*ndropn n ls*))))
using *assms*
by (*metis nfuse-pfilt-nlength pfilt-nlength*)
have *2*: *nlast*(*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) = *nnth nell* (*nnth ls n*)
using *assms pfilt-nnth*[*of* - (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)]
by (*simp add*: *nlength-eq-enat-nfiniteD nnth-nlast ntaken-nnth pfilt-nlength*)
have *3*: *nfirst*(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) =
nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) *0*
by (*metis ndropn-0 ndropn-nfirst*)
have *4*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) *0* =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) *0*)

using *pfilt-nnth*
by (*metis i0-lb zero-enat-def*)
have *5*: *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)) *0*) =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth ls* (*n*+*0*) − (*nnth ls n*))
using *zero-enat-def* **by** *force*
have *6*: *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth ls* (*n*+*0*) − (*nnth ls n*)) =
nnth (*ndropn* (*nnth ls n*) *nell*) *0*
by *simp*
have *7*: *nnth* (*ndropn* (*nnth ls n*) *nell*) *0* = *nnth nell* (*nnth ls n*)
using *assms* **by** *simp*
have *8*: *nlast*(*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) =
nfirst(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)))
using *2 4 5 7 3 6* **by** *presburger*
have *10*: *nnth* (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i* =
(*if i* ≤ *nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
then nnth (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) *i*
else nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − (*the-enat* (*nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))))))
using *1 8* **by** (*meson nfuse-nnth not-le-imp-less*)
have *11*: *nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) = *n*
using *assms* **by** (*simp add*: *pfilt-nlength*)
have *12*: *i*≤*n* ⟶ *nnth* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) *i* =
nnth (*ntaken* (*nnth ls n*) *nell*) (*nnth* (*ntaken n ls*) *i*)
by (*simp add*: *11 pfilt-nnth*)
have *13*: *i*≤*n* ⟶ *nnth* (*ntaken* (*nnth ls n*) *nell*) (*nnth* (*ntaken n ls*) *i*) =
nnth (*ntaken* (*nnth ls n*) *nell*) (*nnth ls i*)
by (*simp add*: *ntaken-nnth*)
have *15*: *i*≤*n* ⟶ *nnth* (*ntaken* (*nnth ls n*) *nell*) (*nnth ls i*) = *nnth nell* (*nnth ls i*)
using *assms* **by** (*simp add*: *nidx-less-eq ntaken-nnth*)
have *16*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − (*the-enat* (*nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))))) =
nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − *n*)
by (*simp add*: *11*)
have *17*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*))) (*i* − *n*) =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*))
using *assms pfilt-nnth*[*of i*−*n* (*ndropn* (*nnth ls n*) *nell*)
(*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*))]
by (*metis enat-minus-mono1 idiff-enat-enat ndropn-nlength nlength-nmap pfilt-nlength*)
have *18*: *i*> *n* ⟶ *nnth* (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*) =
nnth ls (*n*+(*i*−*n*)) − (*nnth ls n*)
using *assms*
by (*metis linorder-le-cases ndropn-nnth nfinite-ntaken nlast-nmap nlength-nmap nnth-nmap ntaken-all ntaken-nlast*)
have *19*: *i*>*n* ⟶ *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* ($\lambda x.$ *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*))
= *nnth* (*ndropn* (*nnth ls n*) *nell*) ((*nnth ls i*) − (*nnth ls n*))
by (*simp add*: *18*)
have *20* : *i*>*n* ⟶ *nnth* (*ndropn* (*nnth ls n*) *nell*) ((*nnth ls i*) − (*nnth ls n*)) =

nnth nell ((*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*)))
using *assms ndropn-nnth* **by** *blast*
have *22*: *i*>*n* ⟶ (*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*)) = (*nnth ls i*)
using *assms* **by** (*simp add*: *nidx-less-eq*)
have *23*: *i*>*n* ⟶ *nnth nell* ((*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*))) = *nnth nell* (*nnth ls i*)
by (*simp add*: *22*)
from *10* **show** *?thesis*
by (*metis 11 12 13 15 16 17 19 20 23 enat-ord-simps*(*1*) *not-le-imp-less*)
qed

lemma *nfuse-pfilt-nnth-b-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite nell*
¬*nfinite ls*
i ≤ *nlength ls*
n ≤ *nlength ls*
shows *nnth* (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x* − (*nnth ls n*)) (*ndropn n ls*)))) *i* =
nnth nell (*nnth ls i*)
proof −
have *1*: *i* ≤ *nlength*(*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x* − (*nnth ls n*)) (*ndropn n ls*))))
using *assms*
by (*metis nfuse-pfilt-nlength pfilt-nlength*)
have *2*: *nlast*(*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) = *nnth nell* (*nnth ls n*)
using *assms pfilt-nnth*[*of* - (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)]
by (*simp add*: *nlength-eq-enat-nfiniteD nnth-nlast ntaken-nnth pfilt-nlength*)
have *3*: *nfirst*(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) =
nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) *0*
by (*metis ndropn-0 ndropn-nfirst*)
have *4*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) *0* =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) *0*)
using *pfilt-nnth*
by (*metis i0-lb zero-enat-def*)
have *5*: *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) *0*) =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth ls* (*n*+*0*) − (*nnth ls n*))
using *zero-enat-def* **by** *force*
have *6*: *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth ls* (*n*+*0*) − (*nnth ls n*)) =
nnth (*ndropn* (*nnth ls n*) *nell*) *0*
by *simp*
have *7*: *nnth* (*ndropn* (*nnth ls n*) *nell*) *0* = *nnth nell* (*nnth ls n*)
using *assms* **by** *simp*
have *8*: *nlast*(*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) =
nfirst(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
using *2 4 5 7 3 6* **by** *presburger*
have *10*: *nnth* (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i* =
(*if i* ≤ *nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))

then nnth (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) *i*
else nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − (*the-enat* (*nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))))))
using *1 8* **by** (*meson nfuse-nnth not-le-imp-less*)
have *11*: *nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) = *n*
using *assms* **by** (*simp add*: *pfilt-nlength*)
have *12*: *i*≤*n* ⟶ *nnth* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*)) *i* =
nnth (*ntaken* (*nnth ls n*) *nell*) (*nnth* (*ntaken n ls*) *i*)
by (*simp add*: *11 pfilt-nnth*)
have *13*: *i*≤*n* ⟶ *nnth* (*ntaken* (*nnth ls n*) *nell*) (*nnth* (*ntaken n ls*) *i*) =
nnth (*ntaken* (*nnth ls n*) *nell*) (*nnth ls i*)
by (*simp add*: *ntaken-nnth*)
have *15*: *i*≤*n* ⟶ *nnth* (*ntaken* (*nnth ls n*) *nell*) (*nnth ls i*) = *nnth nell* (*nnth ls i*)
using *assms* **by** (*simp add*: *nidx-less-eq ntaken-nnth*)
have *16*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − (*the-enat* (*nlength* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))))) =
nnth (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
(*i* − *n*)
by (*simp add*: *11*)
have *17*: *nnth* (*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) (*i* − *n*) =
nnth (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*))
using *assms pfilt-nnth*[*of i*−*n* (*ndropn* (*nnth ls n*) *nell*)
(*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))]
by (*metis enat-minus-mono1 idiff-enat-enat ndropn-nlength nlength-nmap pfilt-nlength*)
have *18*: *i*> *n* ⟶ *nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*) =
nnth ls (*n*+(*i*−*n*)) − (*nnth ls n*)
using *assms*
by (*metis linorder-le-cases ndropn-nnth nfinite-ntaken nlast-nmap nlength-nmap nnth-nmap ntaken-all ntaken-nlast*)
have *19*: *i*>*n* ⟶ *nnth* (*ndropn* (*nnth ls n*) *nell*) (*nnth* (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)) (*i* −*n*))
= *nnth* (*ndropn* (*nnth ls n*) *nell*) ((*nnth ls i*) − (*nnth ls n*))
by (*simp add*: *18*)
have *20* : *i*>*n* ⟶ *nnth* (*ndropn* (*nnth ls n*) *nell*) ((*nnth ls i*) − (*nnth ls n*)) =
nnth nell ((*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*)))
using *assms ndropn-nnth* **by** *blast*
have *22*: *i*>*n* ⟶ (*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*)) = (*nnth ls i*)
using *assms* **by** (*simp add*: *nidx-less-eq*)
have *23*: *i*>*n* ⟶ *nnth nell* ((*nnth ls n*) + ((*nnth ls i*) − (*nnth ls n*))) = *nnth nell* (*nnth ls i*)
by (*simp add*: *22*)
from *10* **show** *?thesis*
by (*metis 11 12 13 15 16 17 19 20 23 enat-ord-simps*(*1*) *not-le-imp-less*)
qed

lemma *nfuse-pfilt-nnth*:
assumes *nidx ls*
nnth ls 0 = *0*
(*nlast ls*) = *the-enat*(*nlength nell*)
nfinite nell

nfinite ls
i ≤ *nlength ls*
n ≤ *nlength ls*
shows *nnth* (*pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls*) *i* =
nnth (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*
using *assms nfuse-pfilt-nnth-a*[*of ls nell i n*]
nfuse-pfilt-nnth-b[*of ls nell i n*]
by *simp*

lemma *nfuse-pfilt-nnth-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite nell*
¬*nfinite ls*
i ≤ *nlength ls*
n ≤ *nlength ls*
shows *nnth* (*pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls*) *i* =
nnth (*nfuse* (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*
using *assms nfuse-pfilt-nnth-a-alt*[*of ls nell i n*]
nfuse-pfilt-nnth-b-alt[*of ls nell i n*]
by *simp*

lemma *nfuse-pfilt*:
assumes *nidx ls*
nnth ls 0 = *0*
(*nlast ls*) = *the-enat*(*nlength nell*)
nfinite nell
nfinite ls
n ≤ *nlength ls*
shows *pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls* =
nfuse (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
using *assms nfuse-pfilt-nlength nfuse-pfilt-nnth*
nellist-eq-nnth-eq **by** (*metis pfilt-nlength*)

lemma *nfuse-pfilt-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite nell*
¬*nfinite ls*
n ≤ *nlength ls*
shows *pfilt* (*nfuse* (*ntaken* (*nnth ls n*) *nell*) (*ndropn* (*nnth ls n*) *nell*)) *ls* =
nfuse (*pfilt* (*ntaken* (*nnth ls n*) *nell*) (*ntaken n ls*))
(*pfilt* (*ndropn* (*nnth ls n*) *nell*) (*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
using *assms nfuse-pfilt-nlength nfuse-pfilt-nnth-alt*
nellist-eq-nnth-eq **by** (*metis pfilt-nlength*)

lemma *nfuse-pfilt-a*:
assumes *nidx ls1*
nfinite ls1
nnth ls1 0 = 0
nidx ls2
nfinite ls2
nnth ls2 0 = nlast ls1
(*nlast ls2*) = *the-enat*(*nlength nell*)
nfinite nell
shows *pfilt* (*nfuse* (*ntaken* (*nlast ls1*) *nell*) (*ndropn* (*nfirst ls2*) *nell*)) (*nfuse ls1 ls2*) =
nfuse (*pfilt* (*ntaken* (*nlast ls1*) *nell*) *ls1*)
(*pfilt* (*ndropn* (*nfirst ls2*) *nell*) (*nmap* (λ*x*. *x*− (*nfirst ls2*)) *ls2*))
proof −
have *0*: ∃ *ll*. (*enat ll*) = *nlength ls1*
using *assms*(*2*) *nfinite-nlength-enat* **by** *fastforce*
obtain *ll* **where** *01*: (*enat ll*) = *nlength ls1*
using *0* **by** *auto*
have *1*: *nlast ls1* = *nfirst ls2*
using *assms* **by** (*metis nlast-NNil ntaken-0 ntaken-nlast*)
have *2*: *nidx* (*nfuse ls1 ls2*)
using *assms nidx-nfuse* **by** *blast*
have *20*: *nnth* (*nfuse ls1 ls2*) *0* = *0*
by (*metis 1 assms*(*3*) *nfuse-nnth zero-enat-def zero-le*)
have *3*: *nlast*(*nfuse ls1 ls2*) = *the-enat*(*nlength nell*)
using *assms*
by (*simp add*: *1 nlast-nfuse*)
have *4*: *ll* ≤ *nlength* (*nfuse ls1 ls2*)
by (*simp add*: *01 nfuse-nlength*)
have *5*: *pfilt* (*nfuse* (*ntaken* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*)
(*ndropn* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*))
(*nfuse ls1 ls2*)
=
nfuse (*pfilt* (*ntaken* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*) (*ntaken ll* (*nfuse ls1 ls2*)))
(*pfilt* (*ndropn* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*)
(*nmap* (λ *x*. *x*− (*nnth* (*nfuse ls1 ls2*) *ll*)) (*ndropn ll* (*nfuse ls1 ls2*))))
using *2 3 4 nfuse-pfilt*[*of* (*nfuse ls1 ls2*) *nell ll*]
using *20 assms nfuse-nfinite* **by** *blast*
have *6*: *pfilt* (*nfuse* (*ntaken* (*nlast ls1*) *nell*) (*ndropn* (*nfirst ls2*) *nell*)) (*nfuse ls1 ls2*) =
pfilt (*nfuse* (*ntaken* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*)
(*ndropn* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*))
(*nfuse ls1 ls2*)
using *1 4*
by (*metis 01 assms*(*2*) *ntaken-nfuse ntaken-nlast the-enat.simps*)
have *7*: (*pfilt* (*ntaken* (*nlast ls1*) *nell*) *ls1*) =
(*pfilt* (*ntaken* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*) (*ntaken ll* (*nfuse ls1 ls2*)))
using *1 4*
by (*metis 01 assms*(*2*) *ntaken-nfuse ntaken-nlast the-enat.simps*)
have *8*: (*pfilt* (*ndropn* (*nfirst ls2*) *nell*) (*nmap* (λ*x*. *x*− (*nfirst ls2*)) *ls2*)) =
(*pfilt* (*ndropn* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*)
(*nmap* (λ*x*. *x*− (*nnth* (*nfuse ls1 ls2*) *ll*)) (*ndropn ll* (*nfuse ls1 ls2*))))

using *1 4*
by (*metis 01 ndropn-nfirst ndropn-nfuse nfinite-conv-nlength-enat the-enat.simps*)
show *?thesis* **using** *5 6 7 8* **by** *auto*
qed

lemma *nfuse-pfilt-a-alt*:
assumes *nidx ls1*
nfinite ls1
nnth ls1 0 = 0
nidx ls2
¬nfinite ls2
nnth ls2 0 = nlast ls1
¬nfinite nell
shows *pfilt* (*nfuse* (*ntaken* (*nlast ls1*) *nell*) (*ndropn* (*nfirst ls2*) *nell*)) (*nfuse ls1 ls2*) =
nfuse (*pfilt* (*ntaken* (*nlast ls1*) *nell*) *ls1*)
(*pfilt* (*ndropn* (*nfirst ls2*) *nell*) (*nmap* (*λx. x−* (*nfirst ls2*)) *ls2*))
proof −
have *0*: *∃ ll.* (*enat ll*) = *nlength ls1*
using *assms*(*2*) *nfinite-nlength-enat* **by** *fastforce*
obtain *ll* **where** *01*: (*enat ll*) = *nlength ls1*
using *0* **by** *auto*
have *1*: *nlast ls1 = nfirst ls2*
using *assms* **by** (*metis nlast-NNil ntaken-0 ntaken-nlast*)
have *2*: *nidx* (*nfuse ls1 ls2*)
using *assms nidx-nfuse* **by** *blast*
have *20*: *nnth* (*nfuse ls1 ls2*) *0 = 0*
by (*metis 1 assms*(*3*) *nfuse-nnth zero-enat-def zero-le*)
have *4*: *ll ≤ nlength* (*nfuse ls1 ls2*)
by (*simp add*: *01 nfuse-nlength*)
have *5*: *pfilt* (*nfuse* (*ntaken* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*)
(*ndropn* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*))
(*nfuse ls1 ls2*)
=
nfuse (*pfilt* (*ntaken* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*) (*ntaken ll* (*nfuse ls1 ls2*)))
(*pfilt* (*ndropn* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*)
(*nmap* (*λ x. x−* (*nnth* (*nfuse ls1 ls2*) *ll*)) (*ndropn ll* (*nfuse ls1 ls2*))))
using *2 4 nfuse-pfilt-alt*[*of* (*nfuse ls1 ls2*) *nell ll*] *20 assms nfuse-nfinite* **by** *blast*
have *6*: *pfilt* (*nfuse* (*ntaken* (*nlast ls1*) *nell*) (*ndropn* (*nfirst ls2*) *nell*)) (*nfuse ls1 ls2*) =
pfilt (*nfuse* (*ntaken* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*)
(*ndropn* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*))
(*nfuse ls1 ls2*)
using *1 4*
by (*metis 01 assms*(*2*) *ntaken-nfuse ntaken-nlast the-enat.simps*)
have *7*: (*pfilt* (*ntaken* (*nlast ls1*) *nell*) *ls1*) =
(*pfilt* (*ntaken* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*) (*ntaken ll* (*nfuse ls1 ls2*)))
using *1 4*
by (*metis 01 assms*(*2*) *ntaken-nfuse ntaken-nlast the-enat.simps*)
have *8*: (*pfilt* (*ndropn* (*nfirst ls2*) *nell*) (*nmap* (*λx. x−* (*nfirst ls2*)) *ls2*)) =
(*pfilt* (*ndropn* (*nnth* (*nfuse ls1 ls2*) *ll*) *nell*)

$(nmap\ (\lambda x.\ x- (nnth\ (nfuse\ ls1\ ls2)\ ll))\ (ndropn\ ll\ (nfuse\ ls1\ ls2))\ ))$
using *1 4*
by (*metis 01 ndropn-nfirst ndropn-nfuse nfinite-conv-nlength-enat the-enat.simps*)
show *?thesis* **using** *5 6 7 8* **by** *auto*
qed

lemma *pfilt-ntaken*:
assumes $n \le nlength\ ls$
shows $ntaken\ n\ (pfilt\ \sigma\ ls) = pfilt\ \sigma\ (ntaken\ n\ ls)$
proof −
have *1*: $nlength\ (ntaken\ n\ (pfilt\ \sigma\ ls)) = nlength\ (\ pfilt\ \sigma\ (ntaken\ n\ ls))$
by (*simp add*: *assms pfilt-nlength*)
have *2*: $\forall\ (i::nat).\ i \le nlength(ntaken\ n\ (pfilt\ \sigma\ ls)) \longrightarrow$
$(nnth\ (ntaken\ n\ (pfilt\ \sigma\ ls))\ i) = (nnth\ (\ pfilt\ \sigma\ (ntaken\ n\ ls))\ i)$
by (*simp add*: *pfilt-nmap*)
show *?thesis* **using** *1 2 nellist-eq-nnth-eq* **by** *blast*
qed

lemma *pfilt-ntaken-nidx*:
assumes $n \le nlength\ nell$
nidx ls
$nnth\ ls\ 0 = 0$
nfinite ls
$nlast\ ls = n$
shows $pfilt\ nell\ ls = pfilt\ (ntaken\ n\ nell)\ ls$
proof −
have *1*: $nlength(pfilt\ nell\ ls) = nlength\ (pfilt\ (ntaken\ n\ nell)\ ls)$
by (*simp add*: *pfilt-nlength*)
have *2*: $\forall (i::nat).\ i \le nlength(pfilt\ nell\ ls) \longrightarrow$
$(nnth\ (pfilt\ nell\ ls)\ i) = (nnth\ (pfilt\ (ntaken\ n\ nell)\ ls)\ i)$
using *assms*
by (*auto simp add*: *nidx-all-le-nlast ntaken-nnth pfilt-nlength pfilt-nnth*)
show *?thesis*
by (*simp add*: *1 2 nellist-eq-nnth-eq*)
qed

lemma *pfilt-ndropn*:
assumes $n \le nlength\ ls$
shows $ndropn\ n\ (pfilt\ nell\ ls) = pfilt\ nell\ (ndropn\ n\ ls)$
proof −
have *1*: $nlength\ (ndropn\ n\ (pfilt\ nell\ ls)) = nlength\ (pfilt\ nell\ (ndropn\ n\ ls))$
by (*simp add*: *assms pfilt-nlength*)
have *2*: $\forall\ (i::nat).\ i \le nlength\ (ndropn\ n\ (pfilt\ nell\ ls)) \longrightarrow$
$(nnth\ (ndropn\ n\ (pfilt\ nell\ ls))\ i) = (nnth\ (pfilt\ nell\ (ndropn\ n\ ls))\ i)$
by (*simp add*: *ndropn-nmap pfilt-nmap*)
show *?thesis* **using** *1 2 nellist-eq-nnth-eq* **by** *blast*
qed

lemma *pfilt-ndropn-nidx-nlength*:
assumes (*enat n*) ≤ *nlength nell*
nidx ls
nnth ls 0 = *0*
shows *nlength* (*pfilt nell* (*nmap* (λ*x*. *x*+ *n*) *ls*)) = *nlength* (*pfilt* (*ndropn n nell*) *ls*)
using *assms* **by** (*simp add*: *pfilt-nlength*)

lemma *pfilt-ndropn-nidx-nnth*:
assumes (*enat n*) ≤ *nlength nell*
nidx ls
nnth ls 0 = *0*
i≤*nlength ls*
shows *nnth* (*pfilt nell* (*nmap* (λ*x*. *x*+ *n*) *ls*)) *i* = *nnth* (*pfilt* (*ndropn n nell*) *ls*) *i*
proof −
have *1*: *nnth* (*pfilt nell* (*nmap* (λ*x*. *x*+ *n*) *ls*)) *i* = *nnth nell* (*nnth* (*nmap* (λ*x*. *x*+ *n*) *ls*) *i*)
by (*simp add*: *assms pfilt-nnth pfilt-nlength*)
have *2*: *nnth nell* (*nnth* (*nmap* (λ*x*. *x*+ *n*) *ls*) *i*) = (*nnth nell* ((*nnth ls i*) + *n*))
by (*simp add*: *assms*)
have *3*: *nnth* (*pfilt* (*ndropn n nell*) *ls*) *i* = *nnth* (*ndropn n nell*) (*nnth ls i*)
by (*simp add*: *assms pfilt-nnth pfilt-nlength*)
have *4*: *nnth* (*ndropn n nell*) (*nnth ls i*) = (*nnth nell* ((*nnth ls i*) + *n*))
by (*metis ntaken-ndropn-nlast ntaken-nlast*)
show *?thesis*
using *1 2 3 4* **by** *presburger*
qed

lemma *pfilt-ndropn-nidx*:
assumes (*enat n*) ≤ *nlength nell*
nidx ls
nnth ls 0 = *0*
shows *pfilt nell* (*nmap* (λ*x*. *x*+ *n*) *ls*) = *pfilt* (*ndropn n nell*) *ls*
using *assms pfilt-ndropn-nidx-nlength pfilt-ndropn-nidx-nnth nellist-eq-nnth-eq*
by (*simp add*: *pfilt-ndropn-nidx-nnth nellist-eq-nnth-eq pfilt-nlength*)

lemma *pfilt-pfilt*:
(*nnth* (*pfilt* (*pfilt nell ls1*) *ls2*) *k*) = (*nnth nell* (*nnth ls1* (*nnth ls2 k*)))
by (*metis enat-le-plus-same*(*1*) *gen-nlength-def ndropn-nmap nlength-code nnth-nmap nnth-zero-ndropn pfilt-nmap*)

lemma *pfilt-pfilt-nmap*:
(*pfilt* (*pfilt nell ls1*) *ls2*) = (*pfilt nell* (*nmap* (λ *x*. (*nnth ls1 x*)) *ls2*))
by (*simp add*: *pfilt-expand pfilt-nlength pfilt-pfilt*)

lemma *pfilt-nmap-pfilt*:
(*pfilt* (*pfilt nell ls1*) *ls2*) = *pfilt nell* (*pfilt ls1 ls2*)
by (*metis pfilt-nmap pfilt-pfilt-nmap*)

lemma *pfilt-nsubn*:
assumes $k \leq n$
$n \leq$ *nlength ls*
shows (*nsubn* (*pfilt nell ls*)) *k n* = (*pfilt nell* (*nsubn ls k n*))
using *assms*
by (*simp add*: *ndropn-nmap nsubn-def1 pfilt-nmap*)

lemma *pfilt-nfusecat-nmap*:
(*pfilt nell* (*nfusecat lls*)) = (*nfusecat* (*nmap* (λ *ls* . (*pfilt nell ls*)) *lls*))
proof –
have *1*: *pfilt nell* (*nfusecat lls*) = *nmap* (*nnth nell*) (*nfusecat lls*)
using *pfilt-nmap*[*of nell* (*nfusecat lls*)] **by** *blast*
have *2*: (*nfusecat* (*nmap* (λ *ls* . (*pfilt nell ls*)) *lls*)) =
(*nfusecat* (*nmap* (*nmap* (*nnth nell*)) *lls*))
using *pfilt-nmap* **by** *metis*
have *3*: *nmap* (*nnth nell*) (*nfusecat lls*) = (*nfusecat* (*nmap* (*nmap* (*nnth nell*)) *lls*))
by (*simp add*: *nmap-nfusecat*)
show *?thesis*
by (*simp add*: *1 2 3*)
qed

10.2.3 powerinterval lemmas

lemma *powerinterval-splita0*:
assumes *nidx ls*
nnth ls 0 = *0*
$n \leq$ *nlength ls*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength* σ)
$i <$ *nlength*(*ntaken n ls*)
shows (*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) =
(*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
proof –
have *01*: (*nnth ls n*) $\leq$ *nlength* σ
using *assms*
by (*metis enat-ord-simps*(*1*) *nfinite-nlength-enat nidx-less-eq nnth-nlast the-enat.simps verit-comp-simplify1*(*2*))
have *02*: (*nnth* (*ntaken n ls*) *i*) $\leq$ (*nnth* (*ntaken n ls*) (*Suc i*))
using *assms*
by (*metis dual-order.trans eSuc-enat enat-ord-simps*(*1*) *ileI1 le-add2 min-def nidx-less-eq ntaken-nlength ntaken-nnth plus-1-eq-Suc*)
have *03*: (*nnth* (*ntaken n ls*) (*Suc i*)) $\leq$ (*nnth ls n*)
using *assms*
by (*metis eSuc-enat enat-ord-simps*(*1*) *ileI1 min-def nidx-less-eq ntaken-nlength ntaken-nnth*)
show *?thesis* **unfolding** *nsubn-def1* **using** *01 02 03 assms*
by (*simp add*: *ntaken-nnth ntaken-ndropn-swap order-subst2*)
qed

lemma *powerinterval-splita0-alt*:

assumes *nidx ls*
nnth ls 0 = 0
$n \leq$ *nlength ls*
$\neg$*nfinite ls*
$\neg$*nfinite* σ
$i <$ *nlength*(*ntaken n ls*)
shows (*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) =
(*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
proof –
have *01*: (*nnth ls n*) $\leq$ *nlength* σ
by (*meson assms*(*5*) *enat-less-imp-le less-enatE nfinite-conv-nlength-enat*)
have *02*: (*nnth* (*ntaken n ls*) *i*) $\leq$ (*nnth* (*ntaken n ls*) (*Suc i*))
using *assms*
by (*metis dual-order.trans eSuc-enat enat-ord-simps*(*1*) *ileI1 le-add2 min-def nidx-less-eq ntaken-nlength ntaken-nnth plus-1-eq-Suc*)
have *03*: (*nnth* (*ntaken n ls*) (*Suc i*)) $\leq$ (*nnth ls n*)
using *assms*
by (*metis eSuc-enat enat-ord-simps*(*1*) *ileI1 min-def nidx-less-eq ntaken-nlength ntaken-nnth*)
show *?thesis* **unfolding** *nsubn-def1* **using** *01 02 03 assms*
by (*simp add*: *ntaken-nnth ntaken-ndropn-swap order-subst2*)
qed

lemma *powerinterval-splita*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
$n \leq$ *nlength ls*
nfinite σ
nlast ls = *the-enat*(*nlength* σ)
powerinterval f σ *ls*
shows *powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*)
proof –
have *0*: ($\forall$ *i*. *i*<*nlength ls* $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f*))
using *assms* **by** (*simp add*: *powerinterval-def*)
have *1*: *powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*) =
($\forall$ *i*. $i <$ *nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*))
using *powerinterval-def* **by** *blast*
have *2*: ($\forall$ *i*. $i <$ *nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*))
proof
fix *i*
show $i <$ *nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*)
proof –
have *21*: *i*<*n* $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f*)
using *0 assms less-le-trans* **by** (*metis enat-ord-simps*(*2*))
have *22*: *i*<*nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* σ (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*)
by (*simp add*: *21 assms*(*4*) *ntaken-nnth*)

have *23*: i<*nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*)
using *22 assms powerinterval-splita0*
using *21* **by** *fastforce*
show *?thesis* **using** *23* **by** *blast*
qed
qed
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *powerinterval-splita-alt*:
assumes *nidx ls*
nnth ls 0 = 0
$\neg$*nfinite ls*
n $\leq$ *nlength ls*
$\neg$*nfinite* σ
powerinterval f σ *ls*
shows *powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*)
proof –
have *0*: ($\forall$ *i*. *i*<*nlength ls* $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f*))
using *assms* **by** (*simp add*: *powerinterval-def*)
have *1*: *powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*) =
($\forall$ *i*. *i* < *nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*))
using *powerinterval-def* **by** *blast*
have *2*: ($\forall$ *i*. *i* < *nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*))
proof
fix *i*
show *i* < *nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*)
proof –
have *21*: *i*<*n* $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f*)
using *0 assms less-le-trans* **by** (*metis enat-ord-simps*(*2*))
have *22*: *i*<*nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* σ (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*)
by (*simp add*: *21 assms*(*4*) *ntaken-nnth*)
have *23*: *i*<*nlength*(*ntaken n ls*) $\longrightarrow$
((*nsubn* (*ntaken* (*nnth ls n*) σ) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) $\models$ *f*)
using *22 assms powerinterval-splita0-alt*
using *21* **by** *fastforce*
show *?thesis* **using** *23* **by** *blast*
qed
qed
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *powerinterval-splitb0*:
assumes *nidx ls*
nnth ls 0 = 0

nfinite ls
n ≤ *nlength ls*
nfinite σ
nlast ls = *the-enat*(*nlength* σ)
i < (*nlength ls* −*n*)
shows (*nsubn* (*ndropn* (*nnth ls n*) σ)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) =
(*nsubn* σ (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*)))
proof −
have *1*: *nlength* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) = (*nlength ls*) − *n*
by (*simp add*: *assms*)
have *2*: *i*<(*nlength ls* −*n*) ⟶
(*nnth* (((*nmap* (λ*x* .*x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*) =
(*nnth ls* (*n* + *i*)) − (*nnth ls n*)
by *simp*
have *3*: *i*<(*nlength ls* −*n*) ⟶
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*)) =
(*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*)
using *assms* **by** (*metis eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap*)
have *4*: *i*<(*nlength ls* −*n*) ⟶
(*nsubn* (*ndropn* (*nnth ls n*) σ)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) =
(*nsubn* (*ndropn* (*nnth ls n*) σ)
((*nnth ls* (*n* + *i*)) − (*nnth ls n*))
((*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*))
)
by (*simp add*: *2 3*)
have *5*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls* (*n* + *i*)) < (*nnth ls* (*n* + (*Suc i*)))
using *assms*
by (*metis Suc-ile-eq add.commute add-Suc-right enat-min nidx-expand plus-enat-simps*(*1*))
have *6*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls n*) ≤ (*nnth ls* (*n* + *i*))
using *assms*
by (*metis add.commute enat-min le-add1 nidx-less-eq order-less-imp-le plus-enat-simps*(*1*))
have *7*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls n*) ≤ (*nnth ls* (*n* + (*Suc i*)))
using *5 6* **by** *linarith*
have *8*: *i*<(*nlength ls* −*n*) ⟶ (*nnth ls* (*n* + *i*)) − (*nnth ls n*) < (*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*)
using *5 6 diff-less-mono* **by** *blast*
have *10*: *i*<(*nlength ls* −*n*) ⟶
(*nsubn* (*ndropn* (*nnth ls n*) σ)
((*nnth ls* (*n* + *i*)) − (*nnth ls n*))
((*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*))
) =
(*nsubn* σ (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*)))
by (*metis 6 7 8 add.commute diff-add nsubn-ndropn*)
show *?thesis*
using *2 3 10 assms* **by** *auto*

qed

lemma *powerinterval-splitb0-alt*:
assumes *nidx ls*
nnth ls 0 = 0
$\neg$*nfinite ls*
$n \leq$ *nlength ls*
$\neg$*nfinite* σ
$i <$ (*nlength ls* $-n$)
shows (*nsubn* (*ndropn* (*nnth ls n*) σ)
(*nnth* (((*nmap* (λx. $x-$ (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (λx. $x-$ (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) =
(*nsubn* σ (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*)))
proof –
have *1*: *nlength* (((*nmap* (λx. $x-$ (*nnth ls n*)) (*ndropn n ls*)))) = (*nlength ls*) $-$ *n*
by (*simp add*: *assms*)
have *2*: *i*<(*nlength ls* $-n$) $\longrightarrow$
(*nnth* (((*nmap* (λx .$x-$ (*nnth ls n*)) (*ndropn n ls*)))) *i*) =
(*nnth ls* (*n* + *i*)) $-$ (*nnth ls n*)
by *simp*
have *3*: *i*<(*nlength ls* $-n$) $\longrightarrow$
(*nnth* (((*nmap* (λx. $x-$ (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*)) =
(*nnth ls* (*n* + (*Suc i*))) $-$ (*nnth ls n*)
using *assms* **by** (*metis eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap*)
have *4*: *i*<(*nlength ls* $-n$) $\longrightarrow$
(*nsubn* (*ndropn* (*nnth ls n*) σ)
(*nnth* (((*nmap* (λx. $x-$ (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (λx. $x-$ (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) =
(*nsubn* (*ndropn* (*nnth ls n*) σ)
((*nnth ls* (*n* + *i*)) $-$ (*nnth ls n*))
((*nnth ls* (*n* + (*Suc i*))) $-$ (*nnth ls n*))
)
by (*simp add*: *2 3*)
have *5*: *i*<(*nlength ls* $-n$) $\longrightarrow$ (*nnth ls* (*n* + *i*)) < (*nnth ls* (*n* + (*Suc i*)))
using *assms*
by (*metis Suc-ile-eq add.commute add-Suc-right enat-min nidx-expand plus-enat-simps*(*1*))
have *6*: *i*<(*nlength ls* $-n$) $\longrightarrow$ (*nnth ls n*) $\leq$ (*nnth ls* (*n* + *i*))
using *assms*
by (*metis add.commute enat-min le-add1 nidx-less-eq order-less-imp-le plus-enat-simps*(*1*))
have *7*: *i*<(*nlength ls* $-n$) $\longrightarrow$ (*nnth ls n*) $\leq$ (*nnth ls* (*n* + (*Suc i*)))
using *5 6* **by** *linarith*
have *8*: *i*<(*nlength ls* $-n$) $\longrightarrow$ (*nnth ls* (*n* + *i*)) $-$ (*nnth ls n*) < (*nnth ls* (*n* + (*Suc i*))) $-$ (*nnth ls n*)
using *5 6 diff-less-mono* **by** *blast*
have *9*: *i*<(*nlength ls* $-n$) $\longrightarrow$ (*nnth ls* (*n* + (*Suc i*))) $-$ (*nnth ls n*) $\leq$ *nlength* σ $-$ (*nnth ls n*)
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat*)
have *10*: *i*<(*nlength ls* $-n$) $\longrightarrow$
(*nsubn* (*ndropn* (*nnth ls n*) σ)

((*nnth ls* (*n* + *i*)) − (*nnth ls n*))
((*nnth ls* (*n* + (*Suc i*))) − (*nnth ls n*))
) =
(*nsubn* σ (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*)))
by (*metis 6 7 8 add.commute diff-add nsubn-ndropn*)
show *?thesis*
using *2 3 10 assms* **by** *auto*
qed

lemma *powerinterval-splitb*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
n ≤ *nlength ls*
nfinite σ
nlast ls = *the-enat*(*nlength* σ)
powerinterval f σ *ls*
shows *powerinterval f* (*ndropn* (*nnth ls n*) σ) ((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))
proof −
have *0*: (∀ *i*. *i*<*nlength ls* ⟶ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f*))
using *assms* **by** (*simp add*: *powerinterval-def*)
have *1*: *powerinterval f* (*ndropn* (*nnth ls n*) σ) ((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*))) =
(∀ *i*. *i*<*nlength* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) ⟶
((*nsubn* (*ndropn* (*nnth ls n*) σ)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) ⊨ *f*))
by (*simp add*: *powerinterval-def*)
have *2*: (∀ *i*. *i*<*nlength*(((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) ⟶
((*nsubn* (*ndropn* (*nnth ls n*) σ)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) ⊨ *f*))
proof
fix *i*
show *i*<*nlength*(((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) ⟶
((*nsubn* (*ndropn* (*nnth ls n*) σ)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) ⊨ *f*)
proof −
have *21*: *i*< (*nlength ls*) −*n* ⟶
((*nsubn* σ (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*))) ⊨ *f*)
using *0 assms*(*4*) *enat-min* **by** *auto*
show *?thesis*
using *21 assms powerinterval-splitb0* **by** *fastforce*
qed
qed
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *powerinterval-splitb-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
n ≤ nlength ls
¬nfinite σ
powerinterval f σ ls
shows *powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x− (nnth ls n)) (ndropn n ls)))*
proof −
have *0*: (∀ *i*. *i*<*nlength ls* ⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f*))
using *assms* **by** (*simp add*: *powerinterval-def*)
have *1*: *powerinterval f* (*ndropn* (*nnth ls n*) *σ*) ((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*))) =
(∀ *i*. *i*<*nlength* (((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) ⟶
((*nsubn* (*ndropn* (*nnth ls n*) *σ*)
(*nnth* (((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) ⊨ *f*))
by (*simp add*: *powerinterval-def*)
have *2*: (∀ *i*. *i*<*nlength*(((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) ⟶
((*nsubn* (*ndropn* (*nnth ls n*) *σ*)
(*nnth* (((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) ⊨ *f*))
proof
fix *i*
show *i*<*nlength*(((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) ⟶
((*nsubn* (*ndropn* (*nnth ls n*) *σ*)
(*nnth* (((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) *i*)
(*nnth* (((*nmap* (*λx*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) (*Suc i*))
) ⊨ *f*)
proof −
have *21*: *i*< (*nlength ls*) −*n* ⟶
((*nsubn σ* (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*))) ⊨ *f*)
using *0 assms*(*4*) *enat-min* **by** *auto*
show *?thesis*
using *21 assms powerinterval-splitb0-alt* **by** *fastforce*
qed
qed
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *powerinterval-split*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
n ≤ nlength ls
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
shows *powerinterval f σ ls* =

(*powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*) $\wedge$
powerinterval f (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x.$ $x-$ (*nnth ls n*)) (*ndropn n ls*))))
proof –
have *1*: *powerinterval f* σ *ls* $\Longrightarrow$
powerinterval f (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*)
by (*simp add*: *assms powerinterval-splita*)
have *2*: *powerinterval f* σ *ls* $\Longrightarrow$
powerinterval f (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x.$ $x-$ (*nnth ls n*)) (*ndropn n ls*)))
by (*simp add*: *assms powerinterval-splitb*)
have *3*: *powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*) =
($\forall$ *i*. *i*<*n* $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f*))
using *assms* **by** (*simp add*: *powerinterval-def powerinterval-splita0*)
have *4*: *powerinterval f* (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x.$ $x-$ (*nnth ls n*)) (*ndropn n ls*))) =
($\forall$ *i*. *i*<(*nlength ls*) −*n* $\longrightarrow$ ((*nsubn* σ (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*))) $\models$ *f*))
unfolding *powerinterval-def* **using** *assms powerinterval-splitb0*[*of ls n* σ] **by** (*simp*)
have *5*: ($\forall$ *i*. *i*<(*nlength ls*) −*n* $\longrightarrow$ ((*nsubn* σ (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*))) $\models$ *f*)) =
($\forall$ *i*. *n* $\leq$ *i* $\wedge$ *i*<(*nlength ls*) $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f*))
using *assms enat-min* **by** *auto*
(*metis diff-add diff-less-mono enat-ord-simps*(*2*) *idiff-enat-enat nfinite-nlength-enat*)
have *6*: (*powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*) $\wedge$
powerinterval f (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x.$ $x-$ (*nnth ls n*)) (*ndropn n ls*)))) $\Longrightarrow$
powerinterval f σ *ls*
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 6* **by** *blast*
qed

lemma *powerinterval-split-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite ls*
n $\leq$ *nlength ls*
¬*nfinite* σ
shows *powerinterval f* σ *ls* =
(*powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*) $\wedge$
powerinterval f (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x.$ $x-$ (*nnth ls n*)) (*ndropn n ls*))))
proof –
have *1*: *powerinterval f* σ *ls* $\Longrightarrow$
powerinterval f (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*)
by (*simp add*: *assms powerinterval-splita-alt*)
have *2*: *powerinterval f* σ *ls* $\Longrightarrow$
powerinterval f (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x.$ $x-$ (*nnth ls n*)) (*ndropn n ls*)))
by (*simp add*: *assms powerinterval-splitb-alt*)
have *3*: *powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*) =
($\forall$ *i*. *i*<*n* $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f*))
using *assms* **by** (*simp add*: *powerinterval-def powerinterval-splita0-alt*)
have *4*: *powerinterval f* (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x.$ $x-$ (*nnth ls n*)) (*ndropn n ls*))) =
($\forall$ *i*. *i*<(*nlength ls*) −*n* $\longrightarrow$ ((*nsubn* σ (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*))) $\models$ *f*))
unfolding *powerinterval-def* **using** *powerinterval-splitb0-alt*[*of ls n* σ] *assms*

by *simp*
have *5*: (∀ *i*. *i*<(*nlength ls*) −*n* ⟶ ((*nsubn σ* (*nnth ls* (*i*+*n*)) (*nnth ls* ((*Suc i*)+*n*))) ⊨ *f*)) =
(∀ *i*. *n* ≤ *i* ∧ *i*<(*nlength ls*) ⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f*))
using *assms enat-min*
by *auto*
(*metis diff-add ndropn-nlength nfinite-ndropn-b nfinite-ntaken not-le-imp-less ntaken-all*)
have *6*: (*powerinterval f* (*ntaken* (*nnth ls n*) *σ*) (*ntaken n ls*) ∧
powerinterval f (*ndropn* (*nnth ls n*) *σ*) ((*nmap* (λ*x*. *x*− (*nnth ls n*)) (*ndropn n ls*)))) ⟹
powerinterval f σ ls
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 6* **by** *blast*
qed

lemma *powerinterval-nfuse*:
assumes *nidx ls1*
nnth ls1 0 = *0*
nidx ls2
nnth ls2 0 = *0*
nfinite ls1
nlast ls1 = *cp*
cp ≤ *nlength σ*
nfinite ls2
nfinite σ
nlast ls2 = *the-enat*(*nlength σ*) − *cp*
shows *powerinterval f σ* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)) =
(*powerinterval f* (*ntaken cp σ*) *ls1* ∧
powerinterval f (*ndropn cp σ*) *ls2*)
proof −
have *1*: *nidx* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
using *assms nidx-nfuse*[*of ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)]
using *Suc-ile-eq nidx-expand zero-enat-def* **by** *force*
have *2*: *cp* = (*nnth ls1* (*the-enat* (*nlength ls1*)))
using *assms* **using** *nnth-nlast* **by** *blast*
have *3*: *nlength ls1* ≤ *nlength* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
by (*simp add*: *nfuse-nlength*)
have *30*: *nfinite* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
using *assms*
by (*metis nfinite-conv-nlength-enat nfuse-nlength nlength-nmap plus-enat-simps*(*1*))
have *31*: *nfirst* (*nmap* (λ*x*. *x*+ *cp*) *ls2*) = *nnth* (*nmap* (λ*x*. *x*+ *cp*) *ls2*) *0*
by (*metis ndropn-0 ndropn-nfirst*)
have *32*: *nnth* (*nmap* (λ*x*. *x*+ *cp*) *ls2*) *0* = *cp*
using *assms nnth-nmap*[*of 0 ls2* (λ*x*. *x*+ *cp*)]
using *zero-enat-def* **by** *auto*
have *33*: *nlast ls1* = *nfirst* (*nmap* (λ*x*. *x*+ *cp*) *ls2*)
by (*simp add*: *31 32 assms*(*6*))
have *34*: *nlast* (*nmap* (λ*x*. *x* + *cp*) *ls2*) = *the-enat*(*nlength σ*)
using *assms*
by (*metis diff-add enat.simps*(*3*) *enat-ord-simps*(*1*) *enat-the-enat nfinite-conv-nlength-enat nlast-nmap*)

have *4*: *nlast* (*nfuse ls1* (*nmap* (λx. *x*+ *cp*) *ls2*)) = *the-enat*(*nlength* σ)
using *assms* **using** *30 assms nlast-nfuse*[*of ls1* (*nmap* (λx. *x*+ *cp*) *ls2*)] *33 34* **by** *presburger*
have *5*: *cp* = *nnth* (*nfuse ls1* (*nmap* (λx. *x*+ *cp*) *ls2*)) (*the-enat* (*nlength ls1*))
using *assms 2*
by (*metis enat.simps*(*3*) *enat-the-enat nfinite-conv-nlength-enat nfuse-def1 nnth-nappend not-le-imp-less order-less-imp-le*)
have *50*: *nlength* (*nmap* (λx. *x*− *cp*) (*nmap* (λx. *x*+ *cp*) *ls2*)) = *nlength ls2*
by *simp*
have *51*: $\bigwedge$*j*. *j* ≤ *nlength ls2* ⟶
nnth (*nmap* (λx. *x*− *cp*) (*nmap* (λx. *x*+ *cp*) *ls2*)) *j* = *nnth ls2 j*
by *simp*
have *6*: (*nmap* (λx. *x*− *cp*) (*nmap* (λx. *x*+ *cp*) *ls2*)) = *ls2*
using *50 51* **by** (*simp add*: *nellist-eq-nnth-eq*)
have *70*: *ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λx::*nat*. *x* + *cp*) *ls2*)) = *ls1*
by (*simp add*: *33 assms*(*5*) *ntaken-nfuse*)
have *71*: *ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λx::*nat*. *x* + *cp*) *ls2*)) =
nmap (λx::*nat*. *x* + *cp*) *ls2*
by (*simp add*: *33 assms*(*5*) *ndropn-nfuse*)
have *72*: *nidx* (*nfuse* (*ls1*::*nat nellist*) (*nmap* (λx::*nat*. *x* + (*cp*::*nat*)) (*ls2*::*nat nellist*)))
by (*simp add*: *1*)
have *73*: *nnth* (*nfuse ls1* (*nmap* (λx::*nat*. *x* + *cp*) *ls2*)) (*0*::*nat*) = (*0*::*nat*)
by (*metis 33 assms*(*2*) *nfuse-nnth zero-enat-def zero-le*)
have *74*: *enat* (*the-enat* (*nlength ls1*)) ≤ *nlength* (*nfuse ls1* (*nmap* (λx::*nat*. *x* + *cp*) *ls2*))
by (*metis 70 assms*(*5*) *enat.simps*(*3*) *enat-the-enat min-def nfinite-nlength-enat ntaken-nlength order-eq-refl*)
have *75*: *enat* (*nlast* (*nfuse ls1* (*nmap* (λx::*nat*. *x* + *cp*) *ls2*))) = *nlength* σ
using *4 assms enat-the-enat nfinite-conv-nlength-enat* **by** *auto*
have *76*: *nfinite* σ
by (*simp add*: *assms*(*9*))
have *80*: *powerinterval f* σ (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)) =
(*powerinterval f* (*ntaken* (*nnth* (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*))) ∧
powerinterval f (*ndropn* (*nnth* (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*nmap* (λx. *x* − *nnth* (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*)))
(*ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)))))
using *72 73 30 74 76 75 4* **using** *powerinterval-split*[*of* (*nfuse ls1* (*nmap* (λx. *x*+ *cp*) *ls2*))
(*the-enat* (*nlength ls1*)) σ *f*]
by *fastforce*
have *81*: *powerinterval f* (*ntaken* (*nnth* (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*))) =
powerinterval f (*ntaken cp* σ) *ls1*
using *5 70* **by** *auto*
have *82*: *powerinterval f* (*ndropn* (*nnth* (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*nmap* (λx. *x* − *nnth* (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*)))
(*ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λx. *x* + *cp*) *ls2*)))) =
powerinterval f (*ndropn cp* σ) *ls2*
using *5 6 71* **by** *presburger*
show *?thesis*
using *80 81 82* **by** *blast*
qed

lemma *powerinterval-nfuse-alt*:
assumes *nidx ls1*
nnth ls1 0 = 0
nidx ls2
nnth ls2 0 = 0
nfinite ls1
nlast ls1 = cp
cp $\leq$ *nlength* σ
$\neg$*nfinite ls2*
$\neg$*nfinite* σ
shows *powerinterval f* σ (*nfuse ls1* (*nmap* (λ*x. x*+ *cp*) *ls2*)) =
(*powerinterval f* (*ntaken cp* σ) *ls1* $\wedge$
powerinterval f (*ndropn cp* σ) *ls2*)
proof −
have *1*: *nidx* (*nfuse ls1* (*nmap* (λ*x. x*+ *cp*) *ls2*))
using *assms nidx-nfuse*[*of ls1* (*nmap* (λ*x. x*+ *cp*) *ls2*)]
using *Suc-ile-eq nidx-expand zero-enat-def* **by** *force*
have *2*: *cp* = (*nnth ls1* (*the-enat* (*nlength ls1*)))
using *assms* **using** *nnth-nlast* **by** *blast*
have *3*: *nlength ls1* $\leq$ *nlength* (*nfuse ls1* (*nmap* (λ*x. x*+ *cp*) *ls2*))
by (*simp add*: *nfuse-nlength*)
have *30*: $\neg$*nfinite* (*nfuse ls1* (*nmap* (λ*x. x*+ *cp*) *ls2*))
by (*metis assms*(*8*) *enat-ile enat-le-plus-same*(*2*) *nfinite-conv-nlength-enat nfuse-nlength nlength-nmap*)

have *5*: *cp* = *nnth* (*nfuse ls1* (*nmap* (λ*x. x*+ *cp*) *ls2*)) (*the-enat* (*nlength ls1*))
by (*metis 2 assms*(*5*) *enat.simps*(*3*) *enat-the-enat nfinite-conv-nlength-enat nfuse-def1 nnth-nappend not-le-imp-less order-less-imp-le*)
have *50*: *nlength* (*nmap* (λ*x. x*− *cp*) (*nmap* (λ*x. x*+ *cp*) *ls2*)) = *nlength ls2*
by *simp*
have *51*: $\bigwedge$*j. j* $\leq$ *nlength ls2* $\longrightarrow$
nnth (*nmap* (λ*x. x*− *cp*) (*nmap* (λ*x. x*+ *cp*) *ls2*)) *j* = *nnth ls2 j*
by *simp*
have *6*: (*nmap* (λ*x. x*− *cp*) (*nmap* (λ*x. x*+ *cp*) *ls2*)) = *ls2*
using *50 51* **by** (*simp add*: *nellist-eq-nnth-eq*)
have *7*: *nlast ls1* = *nfirst* (*nmap* (λ*x. x*+ *cp*) *ls2*)
by (*metis add-0 assms*(*4*) *assms*(*6*) *nfinite-conv-nlength-enat nfinite-ntaken nlast-NNil nlength-NNil nnth-nmap ntaken-0 ntaken-nlast the-enat.simps the-enat-0 zero-le*)
have *70*: *ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*::*nat. x* + *cp*) *ls2*)) = *ls1*
by (*simp add*: *7 assms*(*5*) *ntaken-nfuse*)
have *71*: *ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*::*nat. x* + *cp*) *ls2*)) =
nmap (λ*x*::*nat. x* + *cp*) *ls2*
by (*simp add*: *7 assms*(*5*) *ndropn-nfuse*)
have *72*: *nidx* (*nfuse* (*ls1*::*nat nellist*) (*nmap* (λ*x*::*nat. x* + (*cp*::*nat*)) (*ls2*::*nat nellist*)))
by (*simp add*: *1*)
have *73*: *nnth* (*nfuse ls1* (*nmap* (λ*x*::*nat. x* + *cp*) *ls2*)) (*0*::*nat*) = (*0*::*nat*)
by (*metis 7 assms*(*2*) *nfuse-nnth zero-enat-def zero-le*)
have *74*: *enat* (*the-enat* (*nlength ls1*)) $\leq$ *nlength* (*nfuse ls1* (*nmap* (λ*x. x* + *cp*) *ls2*))
by (*metis 70 assms*(*5*) *enat.simps*(*3*) *enat-the-enat min-def nfinite-nlength-enat ntaken-nlength order-eq-refl*)

have *76*: ¬*nfinite* σ
by (*simp add*: *assms*(*9*))
have *80*: *powerinterval f* σ (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) =
(*powerinterval f* (*ntaken* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*))) ∧
powerinterval f (*ndropn* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*nmap* (λ*x*::*nat*. *x* − *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*)))
(*ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)))))
using *72 73 30 74 76*
using *powerinterval-split-alt*[*of* (*nfuse ls1* (*nmap* (λ*x*. *x*+ *cp*) *ls2*))
(*the-enat* (*nlength ls1*)) σ *f*]
by *blast*
have *81*: *powerinterval f* (*ntaken* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*ntaken* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*))) =
powerinterval f (*ntaken cp* σ) *ls1*
using *5 70* **by** *auto*
have *82*: *powerinterval f* (*ndropn* (*nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*))) σ)
(*nmap* (λ*x*. *x* − *nnth* (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)) (*the-enat* (*nlength ls1*)))
(*ndropn* (*the-enat* (*nlength ls1*)) (*nfuse ls1* (*nmap* (λ*x*. *x* + *cp*) *ls2*)))) =
powerinterval f (*ndropn cp* σ) *ls2*
using *5 6 71* **by** *presburger*
show *?thesis*
using *80 81 82* **by** *blast*
qed

10.2.4 cppl lemmas

lemma *cppl-expand*:
assumes (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ *powerinterval f* σ *ls* ∧
((*nfinite ls* ∧ *nfinite* σ ∧ *nlast ls* = *the-enat*(*nlength* σ)) ∨
(¬*nfinite ls* ∧ ¬ *nfinite* σ)) ∧
((*pfilt* σ *ls*) ⊨ *g*))
shows *nidx* (*cppl f g* σ) ∧ *nnth* (*cppl f g* σ) *0* = *0* ∧ *powerinterval f* σ (*cppl f g* σ) ∧
((*nfinite* (*cppl f g* σ) ∧ *nfinite* σ ∧ *nlast* (*cppl f g* σ) = *the-enat*(*nlength* σ)) ∨
(¬ *nfinite* (*cppl f g* σ) ∧ ¬*nfinite* σ)) ∧ ((*pfilt* σ (*cppl f g* σ)) ⊨ *g*)
proof −
have *0*: *cppl f g* σ = (ϵ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ *powerinterval f* σ *ls* ∧
((*nfinite ls* ∧ *nfinite* σ ∧ *nlast ls* = *the-enat*(*nlength* σ)) ∨
(¬*nfinite ls* ∧ ¬*nfinite* σ)) ∧ ((*pfilt* σ *ls*) ⊨ *g*))
using *cppl-def* **by** *blast*
have *1*: (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ *powerinterval f* σ *ls* ∧
((*nfinite ls* ∧ *nfinite* σ ∧ *nlast ls* = *the-enat*(*nlength* σ)) ∨
(¬*nfinite ls* ∧ ¬*nfinite* σ)) ∧ ((*pfilt* σ *ls*) ⊨ *g*))
using *assms* **by** *auto*
have *2*: *nidx* (*cppl f g* σ) ∧ *nnth* (*cppl f g* σ) *0* = *0* ∧ *powerinterval f* σ (*cppl f g* σ) ∧
((*nfinite* (*cppl f g* σ) ∧ *nfinite* σ ∧ *nlast* (*cppl f g* σ) = *the-enat*(*nlength* σ)) ∨
(¬*nfinite* (*cppl f g* σ) ∧ ¬*nfinite* σ)) ∧ ((*pfilt* σ (*cppl f g* σ)) ⊨ *g*)
using *0 1*
someI-ex[*of* λ*ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ *powerinterval f* σ *ls* ∧
((*nfinite ls* ∧ *nfinite* σ ∧ *nlast ls* = *the-enat*(*nlength* σ)) ∨

$(\neg nfinite\ ls \wedge \neg\ nfinite\ \sigma)) \wedge$
$((pfilt\ \sigma\ ls) \models g)]$ **by** *simp*
show *?thesis*
using *2* **by** *blast*
qed

lemma *cppl-fprojection*:
$(\sigma \models f\ \ fproj\ \ g) =$
$(\ nidx\ \ (cppl\ f\ g\ \sigma) \wedge nnth\ (cppl\ f\ g\ \sigma)\ 0 = 0\ \ \wedge powerinterval\ f\ \sigma\ (cppl\ f\ g\ \sigma) \wedge$
$nfinite\ (cppl\ f\ g\ \sigma) \wedge nfinite\ \sigma \wedge$
$nlast\ (cppl\ f\ g\ \sigma) = the\text{-}enat(nlength\ \sigma) \wedge g\ (pfilt\ \sigma\ (cppl\ f\ g\ \sigma))\ )$
using *cppl-expand*[*of f σ g*]
by (*simp add*: *fprojection-d-def*)
(*metis nfinite-conv-nlength-enat the-enat.simps*)

lemma *cppl-oprojection*:
$(\sigma \models f\ oproj\ g) =$
$(\ nidx\ \ (cppl\ f\ g\ \sigma) \wedge nnth\ (cppl\ f\ g\ \sigma)\ 0 = 0\ \ \wedge powerinterval\ f\ \sigma\ (cppl\ f\ g\ \sigma) \wedge$
$\neg nfinite\ (cppl\ f\ g\ \sigma) \wedge \neg nfinite\ \sigma\ \ \wedge g\ (pfilt\ \sigma\ (cppl\ f\ g\ \sigma))\ )$
using *cppl-expand* **by** (*simp add*: *oprojection-d-def*, *blast*)

lemma *cppl-empty*:
assumes $nlength\ \sigma = 0$
$(\sigma \models f\ \ fproj\ \ g)$
shows $(cppl\ f\ g\ \sigma) = (NNil\ 0)$
using *assms cppl-fprojection*[*of f g σ*]
by (*metis gr-zeroI less-numeral-extra*(*3*) *ndropn-0 ndropn-nlast nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast the-enat.simps the-enat-0*)

lemma *cppl-empty-a*:
assumes $nlength\ \sigma = 0$
$(cppl\ f\ g\ \sigma) = NNil\ 0$
$g(pfilt\ \sigma\ (NNil\ 0))$
shows $(\sigma \models f\ \ fproj\ \ g)$
proof −
have *1*: *nidx* (*NNil 0*)
by (*simp add*: *nidx-def*)
have *10*: $nnth\ (NNil\ 0)\ 0 = 0$
by (*simp add*: *nnth-NNil*)
have *11*: *nfinite* (*NNil 0*)
by *simp*
have *12*: *nfinite σ*
by (*simp add*: *assms*(*1*) *nlength-eq-enat-nfiniteD zero-enat-def*)
have *2*: *powerinterval f σ* (*NNil 0*)
by (*simp add*: *powerinterval-def*)
have *3*: $nlast\ (NNil\ 0) =\ \ nlength\ \sigma$
by (*simp add*: *assms*)
have *4*: $g(pfilt\ \sigma\ (NNil\ 0))$

using *assms* **by** *blast*
from *1 10 11 12 2 3 4* **show** *?thesis* **using** *assms*
by (*simp add*: *cppl-fprojection zero-enat-def*)
qed

lemma *cppl-more*:
assumes *nlength* σ *>0*
(σ $\models$ *f fproj g*)
shows *nlength*(*cppl f g* σ)*>0*
using *assms*
by (*metis cppl-fprojection enat-add-sub-same gen-nlength-def gr-zeroI i0-ne-infinity ndropn-nlast ndropn-nlength nlength-NNil nlength-code nnth-nlast the-enat-0 zero-enat-def*)

lemma *cppl-more-infinite*:
assumes *nlength* σ *>0*
(σ $\models$ *f oproj g*)
shows *nlength*(*cppl f g* σ)*>0*
using *assms*
by (*simp add*: *cppl-oprojection nfinite-conv-nlength-enat*)

lemma *cppl-more-than-first*:
assumes *nlength* σ *>0*
(σ $\models$ *f fproj g*)
shows (*nnth* (*cppl f g* σ) *0*) = *0*
using *assms*
using *cppl-fprojection* **by** *blast*

lemma *cppl-more-than-first-alt*:
assumes *nlength* σ *>0*
(σ $\models$ *f oproj g*)
shows (*nnth* (*cppl f g* σ) *0*) = *0*
using *assms*
using *cppl-oprojection* **by** *blast*

lemma *cppl-more-than-last*:
assumes *nlength* σ *>0*
(σ $\models$ *f fproj g*)
shows *nlast* (*cppl f g* σ) = *the-enat*(*nlength* σ)
using *assms cppl-fprojection* **by** *blast*

lemma *cppl-sub-more*:
assumes $n < k$
$k \leq$ *nlength* σ
((*nsubn* σ *n k*) $\models$ *f fproj g*)
shows *nlength*(*cppl f g* (*nsubn* σ *n k*)) > *0*
using *assms cppl-fprojection*[*of f g* (*nsubn* σ *n k*)]
using *cppl-more nsubn-nlength-gr-one* **by** *blast*

lemma *cppl-bounds*:
assumes $n < k$
$k \leq nlength\ \sigma$
$((nsubn\ \sigma\ n\ k\ ) \models f\ \ fproj\ \ g)$
$i < nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k\ ))$
shows $0 \leq (nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k))\ i) \wedge (nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k\ ))\ i) \leq k\ -n$
using *assms*
using *cppl-fprojection*[*of f g* (*nsubn σ n k*)] **using** *nsubn-nlength*[*of σ n k*]
by *simp*
(*metis enat-minus-mono1 idiff-enat-enat min-def nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast order-less-imp-le the-enat.simps*)

lemma *cppl-nmap-bounds*:
assumes $n < k$
$k \leq nlength\ \sigma$
$((nsubn\ \sigma\ n\ k\ ) \models f\ \ fproj\ \ g)$
$i < nlength\ (nmap\ (\lambda x.\ x+\ \ n)\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k)))$
shows $n \leq (nnth\ (nmap\ (\lambda x.\ x+\ n)\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k)))\ i) \wedge$
$(nnth\ (nmap\ (\lambda x.\ \ x+n)\ (cppl\ f\ g\ (nsubn\ \sigma\ n\ k\ )))\ i) \leq k$
using *assms*
using *cppl-bounds* **by** *fastforce*

lemma *cppl-nfirst*:
assumes $(nsubn\ \sigma\ x1a\ (nfirst\ ls)\ ) \models f\ \ fproj\ \ g$
shows $nfirst((nmap\ (\lambda x.\ x+x1a)\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls)\ ))))\ = x1a$
proof −
have *1*: ($nidx\ \ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls))) \wedge$
$nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls)))\ 0 = 0 \wedge$
$powerinterval\ f\ (nsubn\ \sigma\ x1a\ (nfirst\ ls))\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls))) \wedge$
$nfinite\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls))) \wedge nfinite\ (nsubn\ \sigma\ x1a\ (nfirst\ ls)) \wedge$
$nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls)))\ =$
$the\text{-}enat(nlength\ (nsubn\ \sigma\ x1a\ (nfirst\ ls))) \wedge$
$g\ (pfilt\ (nsubn\ \sigma\ x1a\ (nfirst\ ls))\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls)\ )))$)
using *cppl-fprojection assms* **by** *auto*
have *3*: $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ (nfirst\ ls)\ ))\ 0) = 0$
by (*simp add*: *1*)
show *?thesis*
by (*metis 3 add-cancel-right-left nfirst-eq-nnth-zero nnth-nmap zero-enat-def zero-le*)
qed

lemma *cppl-nfirst-same*:
assumes $(nsubn\ \sigma\ x1a\ x1a) \models f\ \ fproj\ \ g$
shows $nfirst((nmap\ (\lambda x.\ x+\ x1a)\ (cppl\ f\ g\ (nsubn\ \sigma\ x1a\ x1a))))\ = x1a$
proof −
have *1*: $nfirst\ (NNil\ x1a\ ) = x1a$
by (*metis NNil-eq-ntake-iff nellist.inject*(*1*))
from *1 cppl-nfirst* **show** *?thesis*
by (*metis assms*)
qed

lemma *cppl-nlast*:
assumes $((nsubn\ \sigma\ x\ (nfirst\ ls)\ ) \models f\ \ fproj\ \ g)$
$x < nfirst\ ls$
$nfirst\ ls \leq nlength\ \sigma$
shows $nlast((nmap\ (\lambda y.\ y+\ x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls)))))\ = nfirst\ ls$
proof −
have *01*: ($nidx\ \ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls)))\ \wedge$
$nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls)))\ 0 = 0\ \wedge$
$powerinterval\ f\ (nsubn\ \sigma\ x\ (nfirst\ ls))\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls)\ ))\ \wedge$
$nfinite\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls)))\ \wedge\ nfinite\ (nsubn\ \sigma\ x\ (nfirst\ ls))\ \wedge$
$nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls))) = the\text{-}enat(nlength\ (nsubn\ \sigma\ x\ (nfirst\ ls)))\ \wedge$
$g\ (pfilt\ (nsubn\ \sigma\ x\ (nfirst\ ls))\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls))))$)
using *cppl-fprojection assms* **by** (*simp add*: *cppl-fprojection*)
have *02*: $nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls))) =$
$the\text{-}enat(nlength\ (nsubn\ \sigma\ x\ (nfirst\ ls)\ ))$
using *01* **by** *auto*
have *04*: $x < nfirst\ ls$
using *assms* **by** *blast*
have *05*: $nlength\ (nsubn\ \sigma\ x\ (nfirst\ ls)) = (nfirst\ ls)\ -x$
using *assms* **by** (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength*)
have *06*: $nlast((nmap\ (\lambda y.\ y+\ x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ (nfirst\ ls))))) =$
$((nfirst\ ls)\ -x)+x$
using *01 05* **by** *auto*
show *?thesis* **using** *04 06* **by** *auto*
qed

lemma *cppl-nlast-i*:
assumes $((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ \ fproj\ \ g)$
$(nnth\ ls\ i) < (nnth\ ls\ (Suc\ i))$
$(nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma$
shows $nlast((nmap\ (\lambda x.\ x+\ (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))\ =$
$(nnth\ ls\ (Suc\ i))$
proof −
have *01*: ($nidx\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ \wedge$
$nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ 0 = 0\ \wedge$
$powerinterval\ f\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
$(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ \wedge$
$nfinite\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ \wedge$
$nfinite\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ \wedge$
$nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$
$the\text{-}enat(nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ \wedge$
$g\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
$(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))$)
using *assms* **by** (*simp add*: *cppl-fprojection*)
have *02*: $nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$
$the\text{-}enat(nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
using *01* **by** *auto*
have *04*: $(nnth\ ls\ i) < (nnth\ ls\ (Suc\ i))$
by (*simp add*: *assms*)
have *05*: $nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) = (nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i)$

by (*metis assms*(*3*) *enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength*)
have *06*: *nlast*((*nmap* (λx. x+ (*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))))) =
((*nnth ls* (*Suc i*)) −(*nnth ls i*))+(*nnth ls i*)
using *01 05* **by** *auto*
show *?thesis* **using** *04 06* **by** *auto*
qed

10.2.5 lcppl lemmas

lemma *lcppl-nnth*:
assumes *nidx ls*
$i <$ *nlength ls*
shows (*nnth* (*lcppl f g* σ *ls*) *i*) =
(*nmap* (λx .x+ (*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))))
using *assms*
proof (*induct i arbitrary*: *ls*)
case *0*
then show *?case*
proof (*cases ls*)
case (*NNil x1*)
then show *?thesis*
using *0.prems*(*2*) **by** *auto*
next
case (*NCons x21 x22*)
then show *?thesis* **using** *NCons*
proof (*cases is-NNil x22*)
case *True*
then show *?thesis* **using** *NCons* **by** *simp*
(*metis is-NNil-def lcppl-code*(*2*) *nnth-NNil*)
next
case *False*
then show *?thesis* **using** *NCons* **by** *simp*
(*metis lcppl-code*(*3*) *nellist.collapse*(*2*) *nnth-0*)
qed
qed
next
case (*Suc i*)
then show *?case*
proof (*cases ls*)
case (*NNil x1*)
then show *?thesis*
using *Suc.prems*(*2*) **by** *auto*
next
case (*NCons x21 x22*)
then show *?thesis*
proof (*cases is-NNil x22*)
case *True*
then show *?thesis*
by (*metis NCons Suc.prems*(*2*) *enat-0-iff*(*1*) *ile0-eq iless-Suc-eq nat.simps*(*3*) *nellist.collapse*(*1*)
nlength-NCons nlength-NNil)

next
case *False*
then show *?thesis* **using** *NCons* **by** *simp*
(*metis* (*no-types*, *lifting*) *Suc*(*1*) *Suc.prems*(*1*) *Suc.prems*(*2*) *Suc-ile-eq iless-Suc-eq*
lcppl-code(*3*) *nellist.collapse*(*2*) *nellist.sel*(*5*) *nidx-expand nlength-NCons nnth-ntl*)
qed
qed
qed

lemma *lcppl-nlength*:
assumes *nfinite ls*
nidx ls
nlength ls $>$ *0*
shows *nlength*(*lcppl f g* σ *ls*) = (*epred* (*nlength ls*))
using *assms*
proof (*induct ls rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** *simp*
next
case (*NCons x ls*)
then show *?case*
proof (*cases is-NNil ls*)
case *True*
then show *?thesis* **using** *NCons*
by (*metis co.enat.sel*(*2*) *lcppl-code*(*2*) *nellist.collapse*(*1*) *nlength-NCons nlength-NNil*)
next
case *False*
then show *?thesis* **using** *NCons* **unfolding** *nidx-expand* **by** *simp*
(*metis NCons.prems*(*1*) *co.enat.exhaust-sel lcppl-code*(*3*) *ndropn-0 ndropn-nlast nellist.collapse*(*2*)
nellist.disc(*1*) *nidx-LCons-1 nidx-expand nlength-NCons the-enat-0*)
qed
qed

lemma *lcppl-nfinite*:
assumes *nidx ls*
shows *nfinite ls* $\longleftrightarrow$ *nfinite*(*lcppl f g* σ *ls*) (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *a*: *?lhs*
show *?rhs*
using *a assms*
proof (*induct ls rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** *simp*
next
case (*NCons x ls*)
then show *?case*
proof (*cases is-NNil ls*)
case *True*
then show *?thesis* **using** *NCons*

by *simp*
next
case *False*
then show *?thesis* **using** *NCons* **by** *simp*
(*metis lcppl-code*(*3*) *nellist.collapse*(*2*) *nfinite-NConsI*)
qed
qed
next
assume *b*: *?rhs*
show *?lhs*
using *b assms*
proof (*induct zs*≡(*lcppl f g σ ls*) *arbitrary*: *ls rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis is-NNil-imp-nfinite lcppl.disc*(*2*) *nellist.disc*(*1*) *nfinite-ntl*)
next
case (*NCons x ls1*)
then show *?case*
proof (*cases is-NNil ls1*)
case *True*
then show *?thesis* **using** *NCons*
by (*metis is-NNil-imp-nfinite lcppl-code*(*3*) *nellist.collapse*(*2*) *nellist.disc*(*2*) *nellist.sel*(*5*) *nfinite-ntl*)
next
case *False*
then show *?thesis* **using** *NCons* **unfolding** *nidx-expand*
by (*metis NCons.hyps*(*2*) *NCons.prems is-NNil-imp-nfinite lcppl-code*(*3*) *nellist.collapse*(*2*) *nellist.inject*(*2*) *nfinite-ntl nidx-LCons-1*)
qed
qed
qed

lemma *lcppl-nlength-alt*:
assumes ¬*nfinite ls*
nidx ls
shows *nlength*(*lcppl f g σ ls*) = (*epred* (*nlength ls*))
proof −
have *1*: ¬ *nfinite* (*lcppl f g σ ls*)
using *assms*(*1*) *assms*(*2*) *lcppl-nfinite* **by** *blast*
have *2*: *epred* (*nlength ls*) = ∞
using *assms*
by (*metis enat2-cases epred-Infty nlength-eq-enat-nfiniteD*)
have *3*: *nlength* (*lcppl f g σ ls*) = ∞
by (*meson 1 enat2-cases nlength-eq-enat-nfiniteD*)
show *?thesis*
by (*simp add*: *2 3*)
qed

lemma *lcppl-nlength-zero*:
assumes *nidx ls*
nlength ls = *0*
shows *nlength*(*lcppl f g σ ls*) = *0*
using *assms*
by (*metis* (*no-types*, *lifting*) *is-NNil-def lcppl.ctr*(*1*) *le-numeral-extra*(*3*) *nlength-NNil ntaken-0 ntaken-all zero-enat-def*)

lemma *lcppl-nlast*:
assumes *nidx ls*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
nlength ls >*0*
shows *nlast* (*lcppl f g σ ls*) =
(*nmap* (*λx. x*+ (*nnth ls* (*the-enat* (*epred*(*nlength ls*)))))
(*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat* (*epred* (*nlength ls*))))
(*nnth ls* (*the-enat* (*nlength ls*))))))
proof −
have *1*: *nlast* (*lcppl f g σ ls*) = (*nnth* (*lcppl f g σ ls*) ((*the-enat* (*nlength* (*lcppl f g σ ls*)))))
using *assms lcppl-nfinite nnth-nlast* **by** *blast*
have *2*: *nlast* (*lcppl f g σ ls*) =
(*nmap* (*λx. x*+ (*nnth ls* ((*the-enat* (*nlength* (*lcppl f g σ ls*))))))
(*cppl f g* (*nsubn σ* (*nnth ls* ((*the-enat* (*nlength* (*lcppl f g σ ls*)))))
(*nnth ls* (*Suc* (*the-enat* (*nlength* (*lcppl f g σ ls*)))))
)))
using *assms lcppl-nnth*[*of ls* ((*the-enat* (*nlength* (*lcppl f g σ ls*)))) *f g σ*]
lcppl-nlength[*of ls f g σ*]
by (*metis 1 co.enat.exhaust-sel eSuc-enat enat-ord-simps*(*2*) *lcppl-nfinite less-add-same-cancel2 nfinite-nlength-enat plus-1-eq-Suc the-enat.simps zero-less-Suc zero-less-iff-neq-zero*)
have *3*: (*nmap* (*λx. x*+ (*nnth ls* (*the-enat* (*nlength* (*lcppl f g σ ls*)))))
(*cppl f g* (*nsubn σ* (*nnth ls* ((*the-enat* (*nlength* (*lcppl f g σ ls*)))))
(*nnth ls* (*Suc* (*the-enat* (*nlength* (*lcppl f g σ ls*)))))
))) =
(*nmap* (*λx. x*+ (*nnth ls* ((*the-enat* (*epred*(*nlength ls*))))))
(*cppl f g* (*nsubn σ* (*nnth ls* ((*the-enat* (*epred* (*nlength ls*)))))
(*nnth ls* (*Suc* (*the-enat* (*epred*(*nlength ls*))))))))
using *assms lcppl-nlength*
by (*metis* (*no-types*, *lifting*) *nellist.map-cong0*)
have *4*: (*Suc* (*the-enat* (*epred*(*nlength ls*)))) = (*the-enat* (*nlength ls*))
by (*metis assms*(*2*) *assms*(*5*) *co.enat.exhaust-sel enat.distinct*(*2*) *epred-Infty infinity-ne-i0 nfinite-conv-nlength-enat not-gr-zero the-enat-eSuc*)
show *?thesis*
using *1 2 3 4* **by** *presburger*
qed

lemma *lcppl-nlast-nlast*:
assumes *nidx ls*
nfinite ls
nfinite σ

nlast ls = the-enat(nlength σ)
((nsubn σ (nnth ls (the-enat (epred(nlength ls)))) (nnth ls (the-enat(nlength ls)))) ⊨ f fproj g)
nlength ls > 0
shows *nlast (nlast (lcppl f g σ ls)) = the-enat(nlength σ)*
proof −
have *2*: *(nlast (lcppl f g σ ls)) =*
(nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls)))))
(cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))
(nnth ls (the-enat (nlength ls))))))
using *assms lcppl-nlast[of ls σ f g]* **by** *blast*
have *3*: *nlast (nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls)))))*
(cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))
(nnth ls (the-enat (nlength ls)))))) =
(λx. x+ (nnth ls (the-enat (epred(nlength ls)))))
(nlast (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))
(nnth ls (the-enat (nlength ls))))))
using *nnth-nmap assms* **by** *(simp add: cppl-fprojection)*
have *4*: *nnth ls (the-enat (epred (nlength ls))) < nnth ls (Suc (the-enat (epred (nlength ls))))*
using *assms* **unfolding** *nidx-expand*
by *(metis Suc-ile-eq co.enat.exhaust-sel eSuc-enat enat.simps(3) enat-ord-simps(2) enat-the-enat i0-less i0-ne-infinity lessI nfinite-conv-nlength-enat)*
have *5*: *enat (nnth ls (Suc (the-enat (epred (nlength ls))))) ≤ nlength σ*
using *assms*
by *(metis Orderings.order-eq-iff co.enat.exhaust-sel enat.simps(3) enat-the-enat epred-simps(5) i0-less i0-ne-infinity nfinite-conv-nlength-enat nnth-nlast the-enat-eSuc)*
have *6*: *nlast (nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls)))))*
(cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))
(nnth ls (the-enat (nlength ls)))))) =
nnth ls (Suc (the-enat (epred (nlength ls))))
using *cppl-nlast-i[of f g σ ls (the-enat (epred (nlength ls)))] assms 4 5*
by *(metis co.enat.exhaust-sel enat.simps(3) epred-simps(5) i0-less i0-ne-infinity nfinite-conv-nlength-enat the-enat-eSuc)*
have *7*: *nnth ls (Suc (the-enat (epred (nlength ls)))) = the-enat(nlength σ)*
using *assms*
by *(metis co.enat.collapse enat-eSuc-iff nfinite-nlength-enat nnth-nlast the-enat.simps zero-less-iff-neq-zero)*
show *?thesis*
by *(simp add: 2 6 7)*
qed

lemma *lcppl-zero-zero*:
assumes *nidx ls*
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
nlength ls = 0
shows *(nnth (lcppl f g σ ls) 0) =*
(nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls 0))))
proof *(cases ls)*
case *(NNil x1)*

then show *?thesis*
by (*metis lcppl-code*(*1*) *nnth-NNil*)
next
case (*NCons x21 ls*)
then show *?thesis*
using *assms*(*5*) **by** *auto*
qed

lemma *lcppl-nfirst*:
assumes *nidx ls*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength* σ)
($\forall$ *i*. *i*<*nlength ls* $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f fproj g*))
nlength ls > *0*
shows *nfirst*(*nfirst*((*lcppl f g* σ *ls*))) = *nfirst ls*
proof −
have *01*: (*nfirst*((*lcppl f g* σ *ls*))) =
(*nmap* (λ*x*. *x*+ (*nnth ls 0*)) (*cppl f g* (*nsubn* σ (*nnth ls 0*) (*nnth ls* (*Suc 0*)))))
using *assms*
by (*metis lcppl-nnth ndropn-0 ndropn-nfirst zero-enat-def*)
have *02*: *nlength ls* > *0* $\longrightarrow$
nfirst((*nmap* (λ*x*. *x*+ (*nnth ls 0*)) (*cppl f g* (*nsubn* σ (*nnth ls 0*) (*nnth ls* (*Suc 0*)))))) =
(*nnth ls 0*)
using *assms 01 cppl-fprojection*[*of f g*]
by (*metis cppl-nfirst enat-0-iff*(*1*) *ndropn-nfirst*)
show *?thesis*
using *01 02 assms*
by (*metis ndropn-0 ndropn-nfirst*)
qed

lemma *lcppl-nfirst-alt*:
assumes *nidx ls*
¬*nfinite ls*
¬*nfinite* σ
($\forall$ *i*. *i*<*nlength ls* $\longrightarrow$ ((*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f fproj g*))
shows *nfirst*(*nfirst*((*lcppl f g* σ *ls*))) = *nfirst ls*
proof −
have *01*: (*nfirst*((*lcppl f g* σ *ls*))) =
(*nmap* (λ*x*. *x*+ (*nnth ls 0*)) (*cppl f g* (*nsubn* σ (*nnth ls 0*) (*nnth ls* (*Suc 0*)))))
using *assms*
by (*metis i0-less lcppl-nnth nfinite-ntaken nlength-eq-enat-nfiniteD nnth-NNil nnth-nlast*
ntaken-0 ntaken-nlast zero-enat-def)
have *02*: *nfirst*((*nmap* (λ*x*. *x*+ (*nnth ls 0*)) (*cppl f g* (*nsubn* σ (*nnth ls 0*) (*nnth ls* (*Suc 0*)))))) =
(*nnth ls 0*)
using *assms 01 cppl-fprojection*[*of f g*]
by (*metis* (*no-types*, *lifting*) *add-0 gr-zeroI ndropn-0 ndropn-nfirst nlength-eq-enat-nfiniteD*
nnth-nmap zero-enat-def zero-le)
show *?thesis*
using *01 02 assms*

by (*metis ndropn-0 ndropn-nfirst*)
qed

lemma *lcppl-nfusecat-nlastnfirst*:
assumes *nfinite ls*
nidx ls
nfinite σ
nlast ls = the-enat(nlength σ)
$((nlength\ ls = 0 \wedge ((nsubn\ \sigma\ (nnth\ ls\ 0)\ (nnth\ ls\ 0)\ ) \models f\ \ fproj\ \ g)) \vee$
$(nlength\ ls > 0 \wedge (\forall i.\ \ i{<}nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models f\ \ fproj\ \ g))))$
shows *nlastnfirst* (*lcppl f g σ ls*)
proof (*cases is-NNil ls*)
case *True*
then show *?thesis*
by (*metis is-NNil-def lcppl-code*(*1*) *nlastnfirst-NNil*)
next
case *False*
then show *?thesis*
proof (*auto simp add*: *nlastnfirst-def1*)
fix *i*
assume *a*: $enat\ (Suc\ i) \leq nlength\ (lcppl\ f\ g\ \sigma\ \ ls)$
assume *b*: $\neg$ *is-NNil ls*
show $nlast\ (nnth\ (lcppl\ f\ g\ \sigma\ \ ls)\ i) = nfirst\ (nnth\ (lcppl\ f\ g\ \sigma\ \ ls)\ (Suc\ i))$
proof −
have *1*: $nlength\ ls > 0$
by (*metis a assms*(*2*) *assms*(*5*) *enat.inject ile0-eq lcppl-nlength-zero old.nat.distinct*(*1*) *zero-enat-def*)
have *2*: $(\forall i.\ \ i{<}nlength\ ls{\longrightarrow}\ ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models f\ \ fproj\ \ g))$
using *assms*(*5*) **by** *auto*
have *3*: *nidx ls*
by (*simp add*: *assms*(*2*))
have *4*: $i{<}\ nlength\ ls$
by (*metis 1 3 Suc-ile-eq a assms*(*1*) *co.enat.exhaust-sel dual-order.order-iff-strict iless-Suc-eq lcppl-nlength less-numeral-extra*(*3*))
have *5*: $(nnth\ (lcppl\ f\ g\ \sigma\ \ ls)\ i) =$
$nmap\ (\lambda x{::}nat.\ x + nnth\ ls\ i)\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
using *lcppl-nnth*[*of ls i f g σ*]
using *3 4* **by** *blast*
have *6*: $(Suc\ i) < nlength\ ls$
by (*metis 1 3 a assms*(*1*) *co.enat.exhaust-sel iless-Suc-eq lcppl-nlength less-numeral-extra*(*3*))
have *7*: $(nnth\ (lcppl\ f\ g\ \sigma\ \ ls)\ (Suc\ i)) =$
$nmap\ (\lambda x{::}nat.\ x + nnth\ ls\ (Suc\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ (Suc\ i))\ (nnth\ ls\ (Suc\ (Suc\ i)))))$
using *lcppl-nnth*[*of ls Suc i f g σ*]
using *3 6* **by** *blast*
have *71*: $nnth\ ls\ i < nnth\ ls\ (Suc\ i)$
using *assms nidx-expand*[*of ls*] *6 order-less-imp-le* **by** *blast*
have *72*: $enat\ (nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma$
using *assms*
by (*metis 3 6 dual-order.order-iff-strict enat-ord-simps*(*1*) *nfinite-conv-nlength-enat*

nidx-less-last-1 nnth-nlast the-enat.simps)
  have *8*: *nlast* (*nmap* (*λx*::*nat*. *x* + *nnth ls i*) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) =
      (*nnth ls* (*Suc i*))
    using *assms 2 3 4 5 6 cppl-nlast-i*[*of f g σ ls i*]
    using *71 72* **by** *blast*
  have *9*: *nfirst* (*nmap* (*λx*::*nat*. *x* + *nnth ls* (*Suc i*))
        (*cppl f g* (*nsubn σ* (*nnth ls* (*Suc i*)) (*nnth ls* (*Suc* (*Suc i*)))))) =
      (*nnth ls* (*Suc i*))
    using *2 6 cppl-fprojection*[*of f g*]
    by (*metis add-0 nlast-NNil nnth-nmap ntaken-0 ntaken-nlast zero-enat-def zero-order*(*1*))
  show *?thesis* **using** *8 9 5 7* **by** *presburger*
 qed
 qed
qed

lemma *lcppl-nfusecat-nlastnfirst-alt*:
 assumes ¬*nfinite ls*
  nidx ls
  ¬*nfinite σ*
  (∀ *i*. *i*<*nlength ls*⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*))
 shows *nlastnfirst* (*lcppl f g σ ls*)
proof (*auto simp add*: *nlastnfirst-def1*)
fix *i*
assume *a*: *enat* (*Suc i*) ≤ *nlength* (*lcppl f g σ ls*)
show *nlast* (*nnth* (*lcppl f g σ ls*) *i*) = *nfirst* (*nnth* (*lcppl f g σ ls*) (*Suc i*))
proof −
  have *2*: (∀ *i*. *i*<*nlength ls*⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*))
   using *assms* **by** *auto*
  have *3*: *nidx ls*
   by (*simp add*: *assms*)
  have *4*: *i*< *nlength ls*
   by (*meson assms*(*1*) *enat-iless linorder-less-linear nfinite-conv-nlength-enat*)
  have *5*: (*nnth* (*lcppl f g σ ls*) *i*) =
     nmap (*λx*::*nat*. *x* + *nnth ls i*) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))
    using *lcppl-nnth*[*of ls i f g σ*]
    using *3 4* **by** *blast*
  have *6*: (*Suc i*) < *nlength ls*
   by (*metis 4 assms*(*1*) *eSuc-enat ileI1 nlength-eq-enat-nfiniteD order-neq-le-trans*)
  have *7*: (*nnth* (*lcppl f g σ ls*) (*Suc i*)) =
     nmap (*λx*::*nat*. *x* + *nnth ls* (*Suc i*))
      (*cppl f g* (*nsubn σ* (*nnth ls* (*Suc i*)) (*nnth ls* (*Suc* (*Suc i*)))))
    using *lcppl-nnth*[*of ls Suc i f g σ*]
     using *3 6* **by** *blast*
  have *8*: *nlast* (*nmap* (*λx*::*nat*. *x* + *nnth ls i*) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) =
      (*nnth ls* (*Suc i*))
    using *assms 2 3 4 5 6*
    by (*simp add*: *cppl-nlast-i nfinite-conv-nlength-enat nidx-expand*)
  have *9*: *nfirst* (*nmap* (*λx*::*nat*. *x* + *nnth ls* (*Suc i*))
        (*cppl f g* (*nsubn σ* (*nnth ls* (*Suc i*)) (*nnth ls* (*Suc* (*Suc i*)))))) =
      (*nnth ls* (*Suc i*))

using *2 6 cppl-fprojection*[*of f g*]
by (*metis add-0 nlast-NNil nnth-nmap ntaken-0 ntaken-nlast zero-enat-def zero-order*(*1*))
show *?thesis* **using** *8 9 5 7* **by** *presburger*
qed
qed

lemma *lcppl-nfusecat-nlength*:
assumes *nidx ls*
nfinite ls
nfinite σ
nlast ls = the-enat(*nlength σ*)
((*nlength ls =0* ∧ ((*nsubn σ* (*nnth ls 0*) (*nnth ls 0*) ⊨ *f fproj g*))) ∨
(*nlength ls >0* ∧ (∀ *i. i<nlength ls* ⟶ ((*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*))))
shows (*nlength ls =0* ⟶ *nlength*(*nfusecat* (*lcppl f g σ ls*)) = *0*) ∧
(*nlength ls >0* ⟶
nlength(*nfusecat* (*lcppl f g σ ls*)) =
((∑ *k=0..*(*the-enat* (*epred*(*nlength ls*))). ((*nlength* (*nnth* (*lcppl f g σ ls*) *k*))))))
proof −
have *1*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms lcppl-nfusecat-nlastnfirst* **by** *blast*
have *20*: *nlength ls = 0* ⟶
(*lcppl f g σ ls*) =
(*NNil* ((*nmap* (λ*x. x+* (*nnth ls 0*)) (*cppl f g* (*nsubn σ* (*nnth ls 0*) (*nnth ls 0*))))))
by (*metis ile0-eq lcppl-code*(*1*) *nellist-eq-nnth-eq nlength-NNil nnth-NNil the-enat.simps the-enat-0*)
have *2*: *nlength ls = 0* ⟶
nfusecat (*lcppl f g σ ls*) =
((*nmap* (λ*x. x+* (*nnth ls 0*)) (*cppl f g* (*nsubn σ* (*nnth ls 0*) (*nnth ls 0*)))))
using *20 nfusecat-NNil* **by** *auto*
have *3*: *nlength ls = 0* ⟶
nlength ((*nmap* (λ*x. x+* (*nnth ls 0*)) (*cppl f g* (*nsubn σ* (*nnth ls 0*) (*nnth ls 0*))))) = *0*
using *assms*
by (*metis cppl-empty diff-self-eq-0 idiff-enat-enat less-numeral-extra*(*3*) *min.orderE ndropn-nlast ndropn-nlength nlength-NNil nlength-nmap nnth-nlast not-le-imp-less nsubn-nlength the-enat-0 zero-enat-def*)
have *4*: *nlength ls = 0* ⟶ *nlength*(*nfusecat* (*lcppl f g σ ls*)) = *0*
using *2 3* **by** *auto*
have *5*: *nfinite* (*lcppl f g σ ls*)
using *assms*(*1*) *assms*(*2*) *lcppl-nfinite* **by** *blast*
have *6*: *all-nfinite* (*lcppl f g σ ls*)
by (*metis* (*no-types, lifting*) *20 all-nfinite-nnth-a assms*(*1*) *assms*(*2*) *assms*(*5*) *co.enat.exhaust-sel cppl-fprojection iless-Suc-eq lcppl-nlength lcppl-nnth nfinite-nmap nnth-NNil not-less-iff-gr-or-eq*)
have *7*: *nlength* (*lcppl f g σ ls*) = (*epred*(*nlength ls*))
by (*metis assms*(*1*) *assms*(*2*) *assms*(*5*) *epred-0 lcppl-nlength lcppl-nlength-zero*)
have *8*: *nlength ls > 0* ⟶
nlength(*nfusecat* (*lcppl f g σ ls*)) =
(∑ *k=0..*(*the-enat* (*epred*(*nlength ls*))). ((*nlength* (*nnth* (*lcppl f g σ ls*) *k*))))
using *nfusecat-nlength-nfinite*[*of* (*lcppl f g σ ls*)] *5 6 7 1* **by** *presburger*
show *?thesis*

using *4 8* **by** *blast*
qed

lemma *lcppl-nfusecat-nidx*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
nlength ls > 0
shows *nidx (nfusecat ((lcppl f g σ ls)))*
proof –
have *0*: *nlength σ >0 ⟶ nlength ls >0*
using *assms* **by** *auto*
have *2*: *nlength σ >0 ⟶ nlastnfirst (lcppl f g σ ls)*
using *assms lcppl-nfusecat-nlastnfirst* **by** *blast*
have *3*: *(∀ i<nlength ls. nidx (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))))*
using *assms cppl-fprojection* **by** *auto*
have *4*: *(∀ i<nlength ls.*
nidx (nmap (λx. x+(nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))))
using *3* **by** *(simp add: Suc-ile-eq nidx-expand)*
have *5*: *nlength σ >0 ⟶*
(∀ i<nlength ls. nidx (nnth (lcppl f g σ ls) i))
using *assms* **by** *(simp add: 4 lcppl-nnth)*
have *6*: *nlength σ >0 ⟶ nlength (lcppl f g σ ls) = epred(nlength ls)*
using *assms lcppl-nlength* **by** *blast*
have *7*: *nlength σ >0 ⟶*
(∀ i≤nlength ls.
nidx (nnth (lcppl f g σ ls) i))
using *5 assms*
by *(metis 6 Extended-Nat.eSuc-mono antisym-conv2 co.enat.exhaust-sel enat-the-enat epred-simps(5) i0-ne-infinity iless-Suc-eq lcppl-nfinite nnth-beyond nnth-nlast)*
have *68*: *⋀i. i<nlength ls ⟹ 0 < nlength (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⟹*
(nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ (f fproj g) ⟹
0 < nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
using *cppl-more* **by** *blast*
have *69*: *⋀i. i<nlength ls ⟹ 0 < nlength (nsubn σ (nnth ls i) (nnth ls (Suc i)))*
using *assms* **unfolding** *nidx-expand* **by** *simp*
(metis assms(1) dual-order.order-iff-strict eSuc-enat enat-ord-simps(2) ileI1 less-numeral-extra(3) nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast nsubn-nlength-gr-one the-enat.simps)
have *70*: *nlength σ >0 ⟶ (∀ i<nlength ls. (0::enat) < nlength (nnth (lcppl f g σ ls) i))*
using *lcppl-nnth[of ls - f g σ] 68 69 assms(1) assms(6)* **by** *auto*
have *700*: *nlength σ >0 ⟶ (∀ lx∈nset (lcppl f g σ ls). (0::enat) < nlength lx)*
using *70 assms*
by *(metis co.enat.exhaust-sel iless-Suc-eq in-nset-conv-nnth lcppl-nlength less-numeral-extra(3))*
have *71*: *nlength σ >0 ⟶ (∀ lx ∈ nset (lcppl f g σ ls). nfinite lx)*
using *assms* **unfolding** *cppl-fprojection*
by *(metis (no-types, lifting) co.enat.exhaust-sel i0-less iless-Suc-eq in-nset-conv-nnth lcppl-nlength lcppl-nnth nfinite-nmap)*

have *8*: *nlength* σ *>0* $\longrightarrow$
nidx (*nfusecat* ((*lcppl f g* σ *ls*)))
using *assms nidx-nfusecat*[*of* ((*lcppl f g* σ *ls*))] *700 71*
by (*metis 2 5 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength zero-less-iff-neq-zero*)
from *8* **show** *?thesis* **using** *assms*
by (*metis gr-zeroI less-numeral-extra*(*3*) *nfinite-nlength-enat nidx-less-last-1 nnth-nlast the-enat.simps zero-enat-def*)
qed

lemma *lcppl-nfusecat-nidx-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite ls*
¬*nfinite* σ
(∀ *i*<*nlength ls*. (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows *nidx* (*nfusecat* ((*lcppl f g* σ *ls*)))
proof −
have *2*: *nlastnfirst* (*lcppl f g* σ *ls*)
using *assms lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *3*: (∀ *i*<*nlength ls*. *nidx* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))))
using *assms cppl-fprojection* **by** *auto*
have *4*: (∀ *i*<*nlength ls*.
nidx (*nmap* (λ*x*. *x*+(*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *3* **by** (*simp add*: *Suc-ile-eq nidx-expand*)
have *5*: (∀ *i*<*nlength ls*. *nidx* (*nnth* (*lcppl f g* σ *ls*) *i*))
using *assms* **by** (*simp add*: *4 lcppl-nnth*)
have *7*: (∀ *i*<*nlength ls*.
nidx (*nnth* (*lcppl f g* σ *ls*) *i*))
using *5* **by** *simp*
have *60*: ⋀*j*. (*nsubn* σ (*nnth ls j*) (*nnth ls* (*Suc j*))) ⊨ (*f fproj g*)
by (*metis assms*(*3*) *assms*(*5*) *leI nfinite-ntaken ntaken-all*)
have *61*: ⋀*j*. *nnth ls j* < *nnth ls* (*Suc j*)
by (*metis Suc-ile-eq assms*(*1*) *assms*(*3*) *nfinite-ntaken nidx-expand not-le-imp-less ntaken-all*)
have *62*: ⋀*j*. *enat* (*nnth ls* (*Suc j*)) ≤ *nlength* σ
by (*meson assms*(*4*) *enat-iless enat-less-imp-le nfinite-conv-nlength-enat*)
have *63*: ⋀*j*. *nlast* (*nmap* (λ*x*::*nat*. *x* + *nnth ls j*) (*cppl f g* (*nsubn* σ (*nnth ls j*) (*nnth ls* (*Suc j*))))) =
nnth ls (*Suc j*)
using *60 61 62 cppl-nlast-i* **by** *blast*
have *68*: ⋀*i*. *i*<*nlength ls* ⟹ *0* < *nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) ⟹
(*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ (*f fproj g*) ⟹
0 < *nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
using *cppl-more* **by** *blast*
have *69*: ⋀*i*. *i*<*nlength ls* ⟹ *0* < *nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
using *assms* **unfolding** *nidx-expand* **using** *61 62 nsubn-nlength-gr-one* **by** *blast*
have *700*: *nlength* σ *>0* $\longrightarrow$ (∀ *i*<*nlength ls*. (*0*::*enat*) < *nlength* (*nnth* (*lcppl f g* σ *ls*) *i*))
using *lcppl-nnth*[*of ls - f g* σ] *68 69 assms* **by** (*simp*)
have *701*: *nlength* σ *>0* $\longrightarrow$ (∀ *lx*∈*nset* (*lcppl f g* σ *ls*). (*0*::*enat*) < *nlength lx*)
using *700 assms*
by (*metis co.enat.exhaust-sel iless-Suc-eq in-nset-conv-nnth lcppl-nlength-alt nlength-eq-enat-nfiniteD zero-enat-def*)

have *70*: $(\forall\, lx{\in}nset\ (lcppl\ f\ g\ \sigma\ ls).\ (0{::}enat) < nlength\ lx)$
using *701 assms nfinite-conv-nlength-enat zero-enat-def* **by** *fastforce*
have *71*: $(\forall\, lx \in nset\ (lcppl\ f\ g\ \sigma\ ls).\ nfinite\ lx)$
using *assms*
by (*metis* (*no-types, lifting*) *co.enat.exhaust-sel cppl-fprojection iless-Suc-eq in-nset-conv-nnth lcppl-nlength-alt lcppl-nnth nfinite-nmap nlength-eq-enat-nfiniteD zero-enat-def*)
have *8*: $nidx\ \ (nfusecat\ (\ (lcppl\ f\ g\ \sigma\ ls)))$
using *assms nidx-nfusecat*[*of* ((*lcppl f g σ ls*))] *70 71*
by (*metis 2 7 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength-alt nlength-eq-enat-nfiniteD zero-enat-def*)
from *8* **show** *?thesis* **using** *assms* **by** *blast*
qed

lemma *lcppl-nlength-all-nfinite*:
assumes $nidx\ \ ls$
$nnth\ ls\ 0 = 0$
$nfinite\ ls$
$nfinite\ \sigma$
$nlast\ ls = the\text{-}enat(nlength\ \sigma)$
$(\forall\ \ i{<}nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ \ fproj\ \ g)$
$nlength\ \sigma > 0$
shows $(\forall\ \ j \le nlength\ (lcppl\ f\ g\ \sigma\ ls).\ nfinite(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j)\ )$
proof
fix *j*
show $j \le nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow nfinite\ (\ nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j)$
proof −
have *1*: $nlength\ \sigma{>}0 \longrightarrow nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat i0-less lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *2*: $nlength\ \sigma{>}0 \longrightarrow nlength\ ls > 0$
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *3*: $nlength\ \sigma{>}0 \longrightarrow j \le nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$
$(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) =$
$(nmap\ (\lambda x.\ x{+}\ (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j)))))$
using *assms lcppl-nnth*[*of ls j f g σ*]
by (*metis 2 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero*)
have *4*: $nlength\ \sigma{>}0 \longrightarrow j \le nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$
$nfinite\ (nmap\ (\lambda x.\ x{+}\ (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j)))))$
using *assms 2*
by (*metis co.enat.exhaust-sel cppl-fprojection iless-Suc-eq lcppl-nlength nfinite-nmap not-gr-zero*)
show *?thesis*
using *3 4 assms*(*7*) **by** *presburger*
qed
qed

lemma *lcppl-nlength-all-gr-zero*:
assumes $nidx\ \ ls$
$nnth\ ls\ 0 = 0$

nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ *i<nlength ls.* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ >0
shows (∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)
proof
fix *j*
show *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶ *0* < *nlength* (*nnth* (*lcppl f g σ ls*) *j*)
proof −
have *1*: *nlength σ>0* ⟶ *nlastnfirst* (*lcppl f g σ ls*)
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat i0-less lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *2*: *nlength σ>0* ⟶ *nlength ls >0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *3*: *nlength σ>0* ⟶ *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶
(*nnth* (*lcppl f g σ ls*) *j*) =
(*nmap* (*λx. x+* (*nnth ls j*)) (*cppl f g* (*nsubn σ* (*nnth ls j*) (*nnth ls* (*Suc j*)))))
using *assms lcppl-nnth*[*of ls j f g σ*]
by (*metis 2 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero*)
have *4*: *nlength σ>0* ⟶ *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶
nlength (*nmap* (*λx. x+* (*nnth ls j*)) (*cppl f g* (*nsubn σ* (*nnth ls j*) (*nnth ls* (*Suc j*))))) > *0*
using *assms 2 cppl-more*[*of* (*nsubn σ* (*nnth ls j*) (*nnth ls* (*Suc j*))) *f g*]
by *simp*
(*metis co.enat.exhaust-sel eSuc-enat enat-le-plus-same*(*2*) *enat-ord-simps*(*1*) *gen-nlength-def i0-less ileI1 iless-Suc-eq lcppl-nlength nfinite-nlength-enat nidx-expand nidx-less-eq nlength-code nnth-nlast nsubn-nlength-gr-one the-enat.simps*)
show *?thesis*
using *3 4 assms*(*7*) **by** *presburger*
qed
qed

lemma *lcppl-nlength-all-nfinite-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬*nfinite ls*
¬*nfinite σ*
(∀ *i<nlength ls.* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows (∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nfinite*(*nnth* (*lcppl f g σ ls*) *j*))
proof
fix *j*
show *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶ *nfinite* (*nnth* (*lcppl f g σ ls*) *j*)
proof −
have *1*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *3*: *j* ≤ *nlength* (*lcppl f g σ ls*) ⟶
(*nnth* (*lcppl f g σ ls*) *j*) =

($nmap$ ($\lambda x.$ $x+$ ($nnth$ ls j)) ($cppl$ f g ($nsubn$ σ ($nnth$ ls j) ($nnth$ ls (Suc j)))))
using *assms lcppl-nnth*[*of ls j f g σ*] **by** (*metis leI nfinite-ntaken ntaken-all*)
have *4*: $j \leq nlength$ ($lcppl$ f g σ ls) $\longrightarrow$
nfinite ($nmap$ ($\lambda x.$ $x+$ ($nnth$ ls j)) ($cppl$ f g ($nsubn$ σ ($nnth$ ls j) ($nnth$ ls (Suc j)))))
using *assms*
by (*metis Suc-ile-eq cppl-fprojection linorder-le-cases nfinite-nmap nfinite-ntaken ntaken-all*)
show *?thesis*
using *3 4 assms* **by** *presburger*
qed
qed

lemma *lcppl-nlength-all-gr-zero-alt*:
assumes *nidx ls*
$nnth$ ls $0 = 0$
$\neg nfinite$ ls
$\neg nfinite$ σ
($\forall$ i<$nlength$ ls. ($nsubn$ σ ($nnth$ ls i) ($nnth$ ls (Suc i))) $\models$ f $fproj$ g)
shows ($\forall$ $j \leq nlength$ ($lcppl$ f g σ ls). $nlength$($nnth$ ($lcppl$ f g σ ls) j) $>$ 0)
proof
fix *j*
show $j \leq nlength$ ($lcppl$ f g σ ls) $\longrightarrow$ $0 <$ $nlength$ ($nnth$ ($lcppl$ f g σ ls) j)
proof −
have *1*: *nlastnfirst* ($lcppl$ f g σ ls)
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *3*: $j \leq nlength$ ($lcppl$ f g σ ls) $\longrightarrow$
($nnth$ ($lcppl$ f g σ ls) j) $=$
($nmap$ ($\lambda x.$ $x+$ ($nnth$ ls j)) ($cppl$ f g ($nsubn$ σ ($nnth$ ls j) ($nnth$ ls (Suc j)))))
using *assms lcppl-nnth*[*of ls j f g σ*] **by** (*metis leI nfinite-ntaken ntaken-all*)
have *4*: $j \leq nlength$ ($lcppl$ f g σ ls) $\longrightarrow$
$nlength$ ($nmap$ ($\lambda x.$ $x+$ ($nnth$ ls j)) ($cppl$ f g ($nsubn$ σ ($nnth$ ls j) ($nnth$ ls (Suc j))))) $>$ 0
using *assms*
by (*metis Suc-ile-eq cppl-more enat-ile linorder-le-cases nfinite-conv-nlength-enat nidx-expand nlength-nmap nsubn-nlength-gr-one*)
show *?thesis*
using *3 4 assms* **by** *presburger*
qed
qed

lemma *lcppl-nlast-nnth*:
assumes *nidx ls*
$nnth$ ls $0 = 0$
nfinite ls
nfinite σ
$nlast$ ls $=$ $the\text{-}enat$($nlength$ σ)
($\forall$ i<$nlength$ ls. ($nsubn$ σ ($nnth$ ls i) ($nnth$ ls (Suc i))) $\models$ f $fproj$ g)
$nlength$ σ $>$ 0
$j \leq nlength$ ($lcppl$ f g σ ls)
shows $nlast$($nnth$ ($lcppl$ f g σ ls) j) $=$ ($nnth$ ls (Suc j))
proof −

have *0*: *nlength* σ *>0* $\longrightarrow$ *nlength ls >0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *1*: *nlength* σ*>0* $\longrightarrow$
$j \leq$ *nlength* (*lcppl f g* σ *ls*) $\longrightarrow$
nnth (*lcppl f g* σ *ls*) *j* =
(*nmap* (λ*x*. *x+* (*nnth ls j*)) (*cppl f g* (*nsubn* σ (*nnth ls j*) (*nnth ls* (*Suc j*)))))
using *assms lcppl-nnth*[*of ls j f g* σ]
by (*metis 0 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero*)
have *2*: *nlength* σ*>0* $\longrightarrow$
$j \leq$ *nlength* (*lcppl f g* σ *ls*) $\longrightarrow$ (*nnth ls* (*Suc j*)) $\leq$ *nlength* σ
using *assms*
by (*metis dual-order.eq-iff enat.simps*(*3*) *enat-ord-simps*(*1*) *enat-the-enat nfinite-conv-nlength-enat nidx-all-le-nlast nnth-beyond not-le-imp-less*)
have *3*: *nlength* σ*>0* $\longrightarrow$
$j \leq$ *nlength* (*lcppl f g* σ *ls*) $\longrightarrow$
nlast (*nmap* (λ*x*. *x+* (*nnth ls j*)) (*cppl f g* (*nsubn* σ (*nnth ls j*) (*nnth ls* (*Suc j*))))) =
(*nnth ls* (*Suc j*))
using *assms cppl-nlast-i*[*of f g* σ *ls j*]
by (*metis 0 2 Suc-ile-eq co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength nidx-expand*)
show *?thesis* **using** *1 3 assms*
by *presburger*
qed

lemma *lcppl-nlast-nnth-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
$\neg$*nfinite ls*
$\neg$*nfinite* σ
($\forall$ *i<nlength ls*. (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f fproj g*)
$j \leq$ *nlength* (*lcppl f g* σ *ls*)
shows *nlast*(*nnth* (*lcppl f g* σ *ls*) *j*) = (*nnth ls* (*Suc j*))
proof −
have *1*:
$j \leq$ *nlength* (*lcppl f g* σ *ls*) $\longrightarrow$
nnth (*lcppl f g* σ *ls*) *j* =
(*nmap* (λ*x*. *x+* (*nnth ls j*)) (*cppl f g* (*nsubn* σ (*nnth ls j*) (*nnth ls* (*Suc j*)))))

using *assms lcppl-nnth*[*of ls j f g* σ] **by** (*metis leI nfinite-ntaken ntaken-all*)
have *2*:
$j \leq$ *nlength* (*lcppl f g* σ *ls*) $\longrightarrow$ (*nnth ls* (*Suc j*)) $\leq$ *nlength* σ
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat*)
have *3*: $j \leq$ *nlength* (*lcppl f g* σ *ls*) $\longrightarrow$
nlast (*nmap* (λ*x*. *x+* (*nnth ls j*)) (*cppl f g* (*nsubn* σ (*nnth ls j*) (*nnth ls* (*Suc j*))))) =
(*nnth ls* (*Suc j*))
using *assms cppl-nlast-i*[*of f g* σ *ls j*]
by (*metis 2 co.enat.exhaust-sel eSuc-enat ileI1 iless-Suc-eq lcppl-nlength-alt nidx-expand nlength-eq-enat-nfiniteD zero-enat-def*)

show *?thesis* **using** *1 3 assms*
by *presburger*
qed

lemma *lcppl-nfusecat-pfilt-fpower-help*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ *i<nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ > 0
shows (∀ *i<nlength ls*. *g* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)))
proof −
have *1*: (∀ *i<nlength ls*. *g* (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *assms cppl-fprojection* **by** *blast*
have *2*: *nlength σ >0 ⟶*
(∀ *i<nlength ls*.
(*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
(*pfilt σ* (*nmap* (λ*x*. *x+* (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))))

using *assms* **by** (*simp add*: *lcppl-nnth*)
have *3*: *nlength σ >0 ⟶*
(∀ *i<nlength ls*.
nlength (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
nlength (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))
)

by (*simp add*: *2 pfilt-nlength*)
have *4*: *nlength σ >0 ⟶*
(∀ *i<nlength ls*.
(∀ *j≤ nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)).
(*nnth* (*pfilt σ* (*nmap* (λ*x*. *x+* (*nnth ls i*))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))) *j*)
=
(*nnth σ* ((*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*) +(*nnth ls i*)))
)
)
using *nnth-nmap pfilt-nmap* **by** (*metis 2 nlength-nmap*)
have *5*: *nlength σ >0 ⟶*
(∀ *i<nlength ls*.
(∀ *j≤ nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)).
(*nnth* (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) *j*) =
(*nnth* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*))
))
by (*simp add*: *3 pfilt-nnth*)

have *6*: $nlength\ \sigma > 0 \longrightarrow$
$(\forall\ i{<}nlength\ ls.$
$(nnth\ ls\ (Suc\ i)) \le nlength\ \sigma$
$)$
using *assms*
by (*metis eSuc-enat enat-le-plus-same*(*2*) *enat-ord-simps*(*1*) *gen-nlength-def ileI1 nfinite-conv-nlength-enat nidx-less-eq nlength-code nnth-nlast the-enat.simps*)
have *7*: $nlength\ \sigma > 0 \longrightarrow$
$(\forall\ i{<}nlength\ ls.$
$(nnth\ ls\ i) \le (nnth\ ls\ (Suc\ i))$
$)$
using *assms*
by (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *70*: $\bigwedge i\ j\ .\ i{<}nlength\ ls \Longrightarrow (nnth\ ls\ i) < (nnth\ ls\ (Suc\ i)) \Longrightarrow$
$enat\ (nnth\ ls\ (Suc\ i)) \le nlength\ \sigma \Longrightarrow$
$(nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models (f\ \ fproj\ \ g) \Longrightarrow$
$enat\ j < nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \Longrightarrow$
$0 \le nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j\ \wedge$
$nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j \le (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$
using *cppl-bounds* **by** *blast*
have *71*: $\bigwedge i.\ \ i{<}nlength\ ls \Longrightarrow (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models (f\ \ fproj\ \ g) \Longrightarrow$
$nnth\ ls\ i < nnth\ ls\ (Suc\ i) \Longrightarrow$
$enat\ (nnth\ ls\ (Suc\ i)) \le nlength\ \sigma \Longrightarrow$
$nlast\ (nmap\ (\lambda x.\ x + nnth\ ls\ i)\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))) =$
$nnth\ ls\ (Suc\ i)$
using *cppl-nlast-i* **by** *blast*
have *8*: $nlength\ \sigma > 0 \longrightarrow$
$(\forall\ i{<}nlength\ ls.$
$(\forall\ j{\le}\ nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
$(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ j) \le$
$(nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$
$))$
using *6 assms 70 71* **unfolding** *cppl-fprojection nidx-expand* **by** *simp*
(*metis add-diff-cancel-right′ dual-order.eq-iff eSuc-enat ileI1 not-le-imp-less ntaken-all ntaken-nlast*)
have *9*: $nlength\ \sigma > 0 \longrightarrow$
$(\forall\ i{<}nlength\ ls.$
$(\forall\ j{\le}\ nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
$(nnth\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )$
$(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ j)) =$
$(nnth\ \sigma\ ((nnth\ \ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ j) + (nnth\ ls\ i)))$
$))$
using *6 7 8*
by (*simp add*: *add.commute nsubn-def1 ntaken-nnth*)
have *10*: $nlength\ \sigma > 0 \longrightarrow$
$(\forall\ i{<}nlength\ ls.$
$(\forall\ j{\le}\ nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
$(pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )$
$(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ ) =$
$(pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i))$

))
using *2 3 4 9* **by** (*simp add*: *pfilt-expand pfilt-nlength pfilt-nnth*)
show *?thesis* **using** *1 10 assms* **by** *fastforce*
qed

lemma *lcppl-nfusecat-pfilt-fpower-help-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬*nfinite ls*
¬*nfinite σ*
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows (∀ *i*<*nlength ls*. *g* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)))
proof −
have *1*: (∀ *i*<*nlength ls*. *g* (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *assms cppl-fprojection* **by** *blast*
have *2*: (∀ *i*<*nlength ls*.
(*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
(*pfilt σ* (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))))
using *assms* **by** (*simp add*: *lcppl-nnth*)
have *3*: (∀ *i*<*nlength ls*.
nlength (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
nlength (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))
)
by (*simp add*: *2 pfilt-nlength*)
have *4*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)).
(*nnth* (*pfilt σ* (*nmap* (λ*x*. *x*+ (*nnth ls i*))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))) *j*)
=
(*nnth σ* ((*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*) +(*nnth ls i*)))
)
)
using *nnth-nmap pfilt-nmap* **by** (*metis 2 nlength-nmap*)
have *5*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)).
(*nnth* (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) *j*) =
(*nnth* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *j*))
))
by (*simp add*: *3 pfilt-nnth*)
have *6*: (∀ *i*<*nlength ls*.
(*nnth ls* (*Suc i*)) ≤ *nlength σ*
)
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat*)
have *7*: (∀ *i*<*nlength ls*.
(*nnth ls i*) ≤ (*nnth ls* (*Suc i*))

)
using *assms*
by (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *70*: $\bigwedge i\ j\ .\ i<nlength\ ls \Longrightarrow (nnth\ ls\ i) < (nnth\ ls\ (Suc\ i)) \Longrightarrow$
$enat\ (nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma \Longrightarrow$
$(nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models (f\ \ fproj\ \ g) \Longrightarrow$
$enat\ j < nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \Longrightarrow$
$0 \leq nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j\ \wedge$
$nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j \leq (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$
using *cppl-bounds* **by** *blast*
have *71*: $\bigwedge i.\ \ i<nlength\ ls \Longrightarrow (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models (f\ \ fproj\ \ g) \Longrightarrow$
$nnth\ ls\ i < nnth\ ls\ (Suc\ i) \Longrightarrow$
$enat\ (nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma \Longrightarrow$
$nlast\ (nmap\ (\lambda x.\ x + nnth\ ls\ i)\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))) =$
$nnth\ ls\ (Suc\ i)$
using *cppl-nlast-i* **by** *blast*
have *8*: $(\forall\ \ i<nlength\ ls.$
$(\forall\ \ j\leq\ nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
$(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ j) \leq$
$(nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$
$))$
using *assms* **using** *6 assms 70 71* **unfolding** *cppl-fprojection nidx-expand* **by** *simp*
(*metis add-diff-cancel-right′ dual-order.eq-iff eSuc-enat ileI1 not-le-imp-less ntaken-all ntaken-nlast*)
have *9*: $(\forall\ \ i<nlength\ ls.$
$(\forall\ \ j\leq\ nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
$(nnth\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )$
$(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ j)) =$
$(nnth\ \sigma\ ((nnth\ \ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ j) + (nnth\ ls\ i)))$
$))$
using *6 7 8* **by** (*simp add*: *add.commute nsubn-def1 ntaken-nnth*)
have *10*: $(\forall\ \ i<nlength\ ls.$
$(\forall\ \ j\leq\ nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
$(pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )$
$(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ ) =$
$(pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i))$
$))$
using *2 3 4 9* **by** (*simp add*: *pfilt-expand pfilt-nlength pfilt-nnth*)
show *?thesis* **using** *1 10 assms* **by** *fastforce*
qed

10.2.6 lsum lemmas

lemma *lsum-NNil*:
$lsum\ (NNil\ nell)\ \ a = NNil\ (a+(the\text{-}enat\ (nlength\ nell)))$
by *simp*

lemma *lsum-addzero-NNil*:
assumes *nfinite nell*

shows *addzero* (*lsum* (*NNil nell*) *0*) =
(*if nlength nell* = *0 then* (*NNil 0*) *else* (*NCons 0* (*NNil* (*the-enat* (*nlength nell*)))))
using *assms* **unfolding** *addzero-def*
by *simp*
(*metis less-nat-zero-code ndropn-0 ndropn-nfirst ndropn-nlast nlength-NNil nnth-NNil*)

lemma *lsum-eq-NNil-conv*:
(*lsum nells a*) = (*NNil b*) ⟷ *is-NNil nells* ∧ *a*+ (*the-enat* (*nlength*(*nfirst nells*))) = *b*
by (*metis NNil-eq-ntake-iff lsum.disc-iff*(*1*) *lsum-code*(*1*) *nellist.collapse*(*1*) *nellist.disc*(*1*)
nellist.inject(*1*))

lemma *lsum-eq-NCons-conv*:
lsum nells a = (*NCons b nells1*) ⟷
(∃ *nell nells*′. *nells* = (*NCons nell nells*′) ∧ *b* = *a*+(*the-enat* (*nlength nell*)) ∧
nells1 = *lsum nells*′ (*a*+(*the-enat* (*nlength nell*))))
by (*cases nells*) (*simp-all*, *blast*)

lemma *lsum-addzero-NCons*:
addzero (*lsum* (*NCons nell nells*) *0*) = (*NCons 0* (*lsum* (*NCons nell nells*) *0*))
by (*simp add*: *addzero-def*)

lemma *lsum-nfirst*:
nfirst (*lsum nells a*) = *a*+(*the-enat* (*nlength*(*nfirst nells*)))
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **by** *simp* (*metis NNil-eq-ntake-iff nellist.inject*(*1*))
next
case (*NCons x21 nells1*)
then show *?thesis* **by** *simp* (*metis NNil-eq-ntake-iff nnth-0 nnth-NNil ntaken-0 ntaken-nlast*)
qed

lemma *nfinite-lsum-conv-a*:
assumes *nfinite nells*
shows *nfinite* (*lsum nells a*)
using *assms*
proof (*induction nells arbitrary*: *a rule*:*nfinite-induct*)
case (*NNil y*)
then show *?case* **by** *simp*
next
case (*NCons x nells*)
then show *?case* **by** *simp*
qed

lemma *nfinite-lsum-conv-b*:
assumes *nfinite* (*lsum nells a*)

shows *nfinite nells*
using *assms*
proof (*induct zs≡lsum nells a arbitrary*: *a nells rule*:*nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis is-NNil-imp-nfinite lsum-eq-NNil-conv*)
next
case (*NCons x nell*)
then show *?case*
by (*metis is-NNil-imp-nfinite lsum-code*(*2*) *nellist.collapse*(*2*) *nellist.sel*(*5*) *nfinite-NConsI*)
qed

lemma *nfinite-lsum-conv*:
nfinite (*lsum nells a*) ⟷ *nfinite nells*
using *nfinite-lsum-conv-a nfinite-lsum-conv-b* **by** *blast*

lemma *lsum-nlength*:
nlength (*lsum nells a*) = *nlength nells*
proof (*cases nfinite nells*)
case *True*
then show *?thesis*
  proof (*induction nells arbitrary*: *a rule*: *nfinite-induct*)
  case (*NNil y*)
  then show *?case* **by** *simp*
  next
  case (*NCons x nell*)
  then show *?case* **by** *simp*
  qed
next
case *False*
then show *?thesis*
  proof (*coinduction arbitrary*: *nells a rule*: *enat-coinduct*)
  case (*Eq-enat nells1*)
  then show *?case*
    proof −
     have *1*: (*nlength* (*lsum nells1 a*) = (*0*::*enat*)) = (*nlength nells1* = (*0*::*enat*))
       by (*metis Eq-enat nfinite-lsum-conv-b nlength-eq-enat-nfiniteD zero-enat-def*)
   show *?thesis*
   by (*metis 1 Eq-enat nbutlast-not-nfinite nfinite-lsum-conv-b nlength-nbutlast*)
  qed
 qed
qed

lemma *lsum-addzero-nfirst*:
nfirst (*addzero* (*lsum nells 0*)) = *0*
by (*metis addzero-def ndropn-nfirst ndropn-nfuse nellist.disc*(*1*) *nellist.disc*(*2*) *nfuse-leftneutral nfuse-nappend nlast-NNil nlength-NNil nnth-0 the-enat-0*)

lemma *lsum-addzero-nlength*:
assumes *nfinite*(*nfirst nells*)
shows (*nlength nells* = *0* ∧ *nlength*(*nfirst nells*) = *0* ⟶
  nlength (*addzero* (*lsum nells 0*)) = *0*)
 ∧
 (*nlength nells* = *0* ∧ *nlength*(*nfirst nells*) > *0* ⟶
  nlength (*addzero* (*lsum nells 0*)) = *1*)
 ∧
 (*nlength nells* > *0* ⟶
  nlength (*addzero* (*lsum nells 0*)) = (*nlength nells*) +*1*)

using *assms*
by (*simp add*: *addzero-def eSuc-plus-1 lsum-nfirst lsum-nlength*)
  (*metis enat.distinct*(*2*) *enat-the-enat nfinite-nlength-enat zero-enat-def*)

lemma *sum-nnth-help*:
assumes *i*>*0*
  i ≤ *nlength nells* +*1*
shows ($\sum$ *k* = *0*..(*i*−*1*). *nlength* (*nnth* (*nells*) *k*)) =
  ($\sum$ *k* = *1*..*i*. *nlength* (*nnth* (*nells*) (*k*−*1*)))
using *assms*
proof
 (*induct i*)
 case *0*
 then show *?case* **by** *blast*
 next
 case (*Suc i*)
 then show *?case*
 proof (*cases i*)
 case *0*
 then show *?thesis* **by** *simp*
 next
 case (*Suc nat*)
 then show *?thesis*
  proof −
   have *1*: (*0*::*nat*) < (*i*::*nat*)
    by (*simp add*: *Suc*)
   have *2*: *enat i* ≤ *nlength* (*nells*::*'a nellist nellist*) + (*1*::*enat*)
    using *Suc.prems*(*2*) *Suc-ile-eq* **by** *auto*
   have *3*: ($\sum$ *k*::*nat* = *0*::*nat*..*i* − (*1*::*nat*). *nlength* (*nnth nells k*)) =
     ($\sum$ *k*::*nat* = *1*::*nat*..*i*. *nlength* (*nnth nells* (*k* − (*1*::*nat*))))
    using *1 2 Suc.hyps* **by** *blast*
   have *4*: ($\sum$ *k*::*nat* = *0*::*nat*..*Suc i* − (*1*::*nat*). *nlength* (*nnth nells k*)) =
     ($\sum$ *k*::*nat* = *0*::*nat*..*i*. *nlength* (*nnth nells k*))
    by *simp*
   have *5*: ($\sum$ *k*::*nat* = *0*::*nat*..*i*. *nlength* (*nnth nells k*)) =
     nlength (*nnth nells i*) + ($\sum$ *k*::*nat* = *0*::*nat*..(*i*−*1*). *nlength* (*nnth nells k*))
   by (*simp add*: *Suc*)

have *6*: *nlength* (*nnth nells i*) + ($\sum$ *k*::*nat* = *1*::*nat*..*i*. *nlength* (*nnth nells* (*k* − (*1*::*nat*)))) =
($\sum$ *k*::*nat* = *1*::*nat*..*Suc i*. *nlength* (*nnth nells* (*k* − (*1*::*nat*))))
by *simp*
show *?thesis*
using *3 4 5 6* **by** *presburger*
qed
qed
qed

lemma *lsum-nnth*:
assumes *i*≤ *nlength nells*
all-nfinite nells
shows *nnth* (*lsum nells a*) *i* = *a*+($\sum$ *k*::*nat*= *0*..*i*. (*the-enat* (*nlength*(*nnth nells k*))))
using *assms*
proof
(*induct i arbitrary*: *a nells*)
case *0*
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **by** (*simp add*: *nnth-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis* **by** *simp*
qed
next
case (*Suc i*)
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *Suc.prems enat-0-iff*(*1*) **by** *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: *nnth* (*lsum nells a*) (*Suc i*) = *nnth* (*lsum x22* (*a* + (*the-enat* (*nlength x21*)))) *i*
by (*simp add*: *NCons*)
have *2*: *enat* (*i*::*nat*) ≤ *nlength x22*
using *NCons Suc.prems Suc-ile-eq* **by** *auto*
have *20*: *all-nfinite x22*
by (*simp add*: *NCons Suc.prems*(*2*))
have *3*: *nnth* (*lsum x22* (*a* + (*the-enat* (*nlength x21*)))) *i* =
(*a* + (*the-enat* (*nlength x21*))) +
($\sum$ *k*::*nat* = *0*::*nat*..*i*. (*the-enat* (*nlength* (*nnth x22 k*))))
using *Suc.hyps*[*of x22* (*a* + (*the-enat* (*nlength x21*)))] *2 20* **by** *blast*
have *4*: *a* + ($\sum$ *k*::*nat* = *0*::*nat*..*Suc i*. (*the-enat* (*nlength* (*nnth nells k*)))) =
a + (*the-enat* ((*nlength* (*nnth nells 0*)))) +
($\sum$ *k*::*nat* = *1*::*nat*..*Suc i*. (*the-enat* (*nlength* (*nnth nells k*))))
by (*simp add*: *sum.atLeast-Suc-atMost*)

have *5*: (*nlength* (*nnth nells 0*)) = *nlength x21*
by (*simp add*: *NCons*)
have *6*: ($\sum$ *k*::*nat* = *1*::*nat*..*Suc i*. (*the-enat* (*nlength* (*nnth nells k*)))) =
($\sum$ *k*::*nat* = *0*::*nat*.. *i*. (*the-enat* (*nlength* (*nnth nells* (*k*+*1*)))))
using *sum.shift-bounds-cl-nat-ivl*[*of* λ*k*. (*the-enat* (*nlength* (*nnth nells k*))) *0 1 i*]
by *simp*
have *7*: ($\sum$ *k*::*nat* = *0*::*nat*.. *i*. (*the-enat* (*nlength* (*nnth nells* (*k*+*1*))))) =
($\sum$ *k*::*nat* = *0*::*nat*.. *i*. (*the-enat* (*nlength* (*nnth x22* (*k*)))))
using *NCons* **by** *auto*
show *?thesis*
using *1 3 4 5 6 7* **by** *presburger*
qed
qed
qed

lemma *lsum-addzero-nnth*:
assumes *i*≤ *nlength* (*addzero* (*lsum nells 0*))
nfinite (*nfirst nells*)
shows (*nlength nells* = *0* ∧ *nlength*(*nfirst nells*) = *0* ⟶
nnth (*addzero* (*lsum nells 0*)) *i* = (*nnth* (*lsum nells 0*) *i*))
∧
(*nlength nells* = *0* ∧ *nlength*(*nfirst nells*) > *0* ⟶
nnth (*addzero* (*lsum nells 0*)) *i* = (*nnth* (*NCons 0* (*lsum nells 0*))) *i*)
∧
(*nlength nells* > *0* ⟶
nnth (*addzero* (*lsum nells 0*)) *i* = (*nnth* (*NCons 0* (*lsum nells 0*)) *i*))

using *assms* **using** *lsum-addzero-nlength*[*of nells*] *lsum-nlength*[*of nells 0*]
lsum-nfirst[*of nells 0*] **using** *addzero-def* **by** *auto*

lemma *lsum-nlast*:
assumes *nfinite nells*
all-nfinite nells
shows *nlast* (*lsum nells a*) = *a*+($\sum$ *k*::*nat*= *0*..(*the-enat* (*nlength nells*)). (*the-enat* (*nlength*(*nnth nells k*))))
using *assms*
by (*metis* (*no-types*, *lifting*) *enat.simps*(*3*) *enat-le-plus-same*(*2*) *enat-the-enat gen-nlength-def lsum-nlength lsum-nnth nfinite-lsum-conv-a nfinite-nlength-enat nlength-code nnth-nlast sum.cong*)

lemma *lsum-addzero-nlast*:
assumes *nfinite nells*
shows *nlast* (*addzero* (*lsum nells 0*)) = *nlast*(*lsum nells 0*)
by (*simp add*: *addzero-def*)

lemma *lsum-nnth-nfinite*:
assumes *i* ≤ *nlength nells*
all-gr-zero nells
all-nfinite nells

shows ($\sum$ *k::nat= 0..i.* (*the-enat* (*nlength*(*nnth nells k*)))) $< \infty$
using *assms*
using *enat-ord-code*(*4*) **by** *blast*

lemma *sum-finite*:
assumes *all-nfinite nells*
i≤ nlength nells
shows ($\sum$ *k= 0..i.* (*nlength*(*nnth nells k*))) $< \infty$
using *assms*
proof (*induction i arbitrary*: *nells*)
case *0*
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *0* **by** (*simp add*: *nfinite-nlength-enat nnth-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis* **using** *0* **using** *nfinite-nlength-enat* **by** *simp blast*
qed
next
case (*Suc i*)
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *Suc* **by** *simp* (*metis enat.inject old.nat.distinct*(*2*) *zero-enat-def*)
next
case (*NCons x21 x22*)
then show *?thesis*
proof –
have *1*: ($\sum$ *k* = *0..Suc i. nlength* (*nnth nells k*)) =
nlength (*nnth nells 0*) + ($\sum$ *k* = *1..Suc i. nlength* (*nnth nells k*))
by (*metis One-nat-def sum.atLeast0-atMost-Suc-shift sum.atLeast-Suc-atMost-Suc-shift*)
have *2*: ($\sum$ *k* = *1..Suc i. nlength* (*nnth nells k*)) =
($\sum$ *k* = *0.. i. nlength* (*nnth nells* (*k+1*)))
using *sum.shift-bounds-cl-nat-ivl*[*of* λ*k* . (*nlength* (*nnth nells k*)) *0 1 i*]
by (*metis One-nat-def Suc-eq-plus1*)
have *3*: ($\sum$ *k* = *0.. i. nlength* (*nnth nells* (*k+1*))) = ($\sum$ *k* = *0.. i. nlength* (*nnth x22 k*))
using *NCons* **by** *auto*
have *4*: *all-nfinite x22*
by (*simp add*: *NCons Suc.prems*(*1*))
have *5*: *i≤ nlength x22*
using *NCons Suc.prems*(*2*) *Suc-ile-eq* **by** *auto*
have *6*: ($\sum$ *k* = *0.. i. nlength* (*nnth x22 k*)) $< \infty$
using *Suc.IH*[*of x22*] *4 5* **by** *blast*
have *7*: *nlength* (*nnth nells 0*)$< \infty$
by (*metis Suc.prems*(*1*) *all-nfinite-nnth-b enat.distinct*(*2*) *enat-ord-simps*(*4*)
nfinite-nlength-enat zero-enat-def zero-le)
have *8*: *nlength* (*nnth nells 0*) + ($\sum$ *k* = *0.. i. nlength* (*nnth x22 k*)) $< \infty$
using *6 7* **by** *force*

show *?thesis* **using** *1 2 3 8* **by** *presburger*
qed
qed
qed

lemma *sum-the-enat*:
assumes *all-nfinite nells*
i≤ *nlength nells*
shows $(\sum k = 0..i.\ the\text{-}enat(nlength(nnth\ nells\ k))) = the\text{-}enat(\sum k = 0..i.\ (nlength(nnth\ nells\ k)))$
using *assms*
proof (*induction i arbitrary*: *nells*)
case *0*
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *0* **by** *simp*
next
case (*NCons x21 x22*)
then show *?thesis* **using** *0* **by** *simp*
qed
next
case (*Suc i*)
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *Suc* **by** *simp* (*metis enat-0-iff*(*2*) *old.nat.distinct*(*2*))
next
case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: $(\sum k = 0..Suc\ i.\ the\text{-}enat\ (nlength\ (nnth\ nells\ k))) =$
$the\text{-}enat(nlength\ (nnth\ nells\ 0)) + (\sum k = 1..Suc\ i.\ the\text{-}enat\ (nlength\ (nnth\ nells\ k)))$
by (*simp add*: *sum.atLeast-Suc-atMost*)
have *2*: $the\text{-}enat(nlength\ (nnth\ nells\ 0)) = the\text{-}enat(nlength\ x21)$
using *NCons* **by** *simp*
have *3*: $(\sum k = 1..Suc\ i.\ the\text{-}enat\ (nlength\ (nnth\ nells\ k))) =$
$(\sum k = 0..i.\ the\text{-}enat\ (nlength\ (nnth\ nells\ (k+1))))$
using *sum.shift-bounds-cl-nat-ivl*[*of* λ*k* .*the-enat* (*nlength* (*nnth nells k*)) *0 1 i*]
by (*metis One-nat-def Suc-eq-plus1*)
have *4*: $(\sum k = 0..i.\ the\text{-}enat\ (nlength\ (nnth\ nells\ (k+1)))) =$
$(\sum k = 0..i.\ the\text{-}enat\ (nlength\ (nnth\ x22\ k)))$
using *NCons* **by** *auto*
have *5*: *all-nfinite x22*
by (*simp add*: *NCons Suc.prems*(*1*))
have *6*: *i*≤ *nlength x22*
using *NCons Suc.prems*(*2*) *Suc-ile-eq* **by** *auto*
have *7*: $(\sum k = 0..i.\ the\text{-}enat\ (nlength\ (nnth\ x22\ k))) =$
$the\text{-}enat(\sum k = 0..i.\ (nlength\ (nnth\ x22\ k)))$
using *Suc.IH*[*of x22*] *5 6* **by** *blast*
have *8*: $the\text{-}enat(\sum k = 0..i.\ (nlength\ (nnth\ x22\ k))) =$

the-enat($\sum k = 0..i.$ (*nlength* (*nnth nells* (*k*+*1*))))
using *NCons* **by** *auto*
have *9*: *the-enat*($\sum k = 0..i.$ (*nlength* (*nnth nells* (*k*+*1*)))) =
the-enat($\sum k = 1..Suc\ i.$ (*nlength* (*nnth nells k*)))
using *sum.shift-bounds-cl-nat-ivl*[*of* λk . (*nlength* (*nnth nells k*)) *0 1 i*]
by (*metis One-nat-def Suc-eq-plus1*)
have *10*: ($\sum k = 1..Suc\ i.$ (*nlength* (*nnth nells k*))) < ∞
proof −
have *100*: ($\sum k = 1..Suc\ i.$ (*nlength* (*nnth nells k*))) =
($\sum k = 0..\ i.$ (*nlength* (*nnth nells* (*k*+*1*))))
using *sum.shift-bounds-cl-nat-ivl*[*of* λk . (*nlength* (*nnth nells k*)) *0 1 i*]
by (*metis One-nat-def Suc-eq-plus1*)
have *101*: ($\sum k = 0..\ i.$ (*nlength* (*nnth nells* (*k*+*1*)))) =
($\sum k = 0..\ i.$ (*nlength* (*nnth x22 k*)))
using *NCons* **by** *auto*
have *102*: ($\sum k = 0..\ i.$ (*nlength* (*nnth x22 k*))) < ∞
using *sum-finite*[*of x22 i*] *5 6* **by** *blast*
show *?thesis*
using *100 101 102* **by** *presburger*
qed
have *11*: $\exists$ *m* . (*enat m*) = ($\sum k = 1..Suc\ i.$ (*nlength* (*nnth nells k*)))
by (*metis 10 less-infinityE*)
obtain *m* **where** *12*: (*enat m*) = ($\sum k = 1..Suc\ i.$ (*nlength* (*nnth nells k*)))
using *11* **by** *blast*
have *13*: $\exists$ *n*. (*enat n*) = (*nlength* (*nnth nells 0*))
by (*metis Suc.prems*(*1*) *all-nfinite-nnth-b nfinite-nlength-enat zero-enat-def zero-le*)
obtain *n* **where** *14*: (*enat n*) = (*nlength* (*nnth nells 0*))
using *13* **by** *blast*
have *15*: *the-enat*(*nlength* (*nnth nells 0*)) + *the-enat*($\sum k = 1..Suc\ i.$ (*nlength* (*nnth nells k*))) =
the-enat(*nlength* (*nnth nells 0*) + ($\sum k = 1..Suc\ i.$ (*nlength* (*nnth nells k*))))
by (*metis 12 14 plus-enat-simps*(*1*) *the-enat.simps*)
have *16*: *the-enat*(*nlength* (*nnth nells 0*)) + *the-enat*($\sum k = 1..Suc\ i.$ (*nlength* (*nnth nells k*))) =
the-enat($\sum k = 0..Suc\ i.$ (*nlength* (*nnth nells k*)))
using *sum.atLeast-Suc-atMost*[*of 0 Suc i* λk. *nlength* (*nnth nells k*)]
by (*metis 15 One-nat-def zero-le*)
show *?thesis*
using *1 16 3 4 7 8 9* **by** *presburger*
qed
qed
qed

lemma *lsum-nnth-leq-Suc*:
assumes *i* < *nlength nells*
all-gr-zero nells
all-nfinite nells
a < ∞
shows *nnth* (*lsum nells a*) *i* < *nnth* (*lsum nells a*) (*Suc i*)
proof −

have *1*: *nnth* (*lsum nells a*) *i* = *a*+($\sum$ *k*::*nat*= *0*..*i*. (*the-enat* (*nlength*(*nnth nells k*))))
  using *assms less-imp-le-nat lsum-nnth* **by** (*metis order-less-imp-le*)
have *2*: *nnth* (*lsum nells a*) (*Suc i*) = *a*+($\sum$ *k*::*nat*= *0*..(*Suc i*). (*the-enat* (*nlength*(*nnth nells k*))))
  using *assms Suc-ile-eq lsum-nnth* **by** *blast*
have *3*: ($\sum$ *k*::*nat*= *0*..(*Suc i*). (*the-enat* (*nlength*(*nnth nells k*)))) =
  ($\sum$ *k*::*nat*= *0*..*i*. (*the-enat* (*nlength*(*nnth nells k*)))) + (*the-enat* (*nlength*(*nnth nells* (*Suc i*))))
  using *sum.atLeast0-atMost-Suc* **by** *blast*
have *4*: *nlength*(*nnth nells* (*Suc i*)) > *0*
  using *assms* **by** (*metis eSuc-enat ileI1 in-nset-conv-nnth*)
have *5*: ($\sum$ *k*::*nat*= *0*..*i*. (*the-enat* (*nlength*(*nnth nells k*)))) < ∞
  using *lsum-nnth-nfinite*[*of i nells*] *assms order.order-iff-strict* **by** *blast*
have *6*: *nlength*(*nnth nells* (*Suc i*)) < ∞
  using *assms*
  by (*metis all-nfinite-nnth-b eSuc-enat enat.distinct*(*2*) *enat-ord-simps*(*4*) *ileI1 nfinite-nlength-enat*)
have *7*: *a*+ ($\sum$ *k*::*nat*= *0*..*i*. (*the-enat* (*nlength*(*nnth nells k*)))) <
  a+ ($\sum$ *k*::*nat*= *0*..(*Suc i*). (*the-enat* (*nlength*(*nnth nells k*))))
  using *4 5 assms 6 enat-the-enat zero-enat-def* **by** *fastforce*
show *?thesis*
using *1 2 7* **by** *presburger*
qed

lemma *lsum-addzero-nnth-leq-Suc*:
assumes *i* < *nlength*(*addzero* (*lsum nells 0*))
  all-gr-zero nells
  all-nfinite nells
shows *nnth* (*addzero* (*lsum nells 0*)) *i* < *nnth* (*addzero* (*lsum nells 0*)) (*Suc i*)
using *assms*
proof (*cases i*)
case *0*
then show *?thesis*
  proof (*cases nells*)
  case (*NNil x1*)
  then show *?thesis* **using** *0 assms* **unfolding** *addzero-def* **by** *simp*
  (*metis enat-0-iff*(*2*) *ndropn-nfirst ndropn-nlast nfinite-code*(*1*) *nfinite-nlength-enat nlast-NNil*
  nlength-NNil nnth-nlast the-enat.simps)
  next
  case (*NCons x21 x22*)
  then show *?thesis* **using** *0 assms* **unfolding** *addzero-def* **by** *simp*
  (*metis eSuc-infinity enat-the-enat gr-zeroI ndropn-eq-NNil ndropn-nlast not-eSuc-ilei0*
  zero-enat-def)
  qed
  next
case (*Suc nat*)
then show *?thesis*
  proof (*cases nells*)
  case (*NNil x1*)
  then show *?thesis*
  using *Suc addzero-def assms*(*1*) *enat-0-iff*(*1*) **by** *fastforce*
  next

case (*NCons x21 x22*)
then show *?thesis*
proof −
have *1*: *nat* = *0* ⟹ *?thesis*
using *Suc assms* **unfolding** *addzero-def* **by** *auto*
(*simp add*: *Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc*,
metis Extended-Nat.eSuc-mono NCons One-nat-def assms(*1*) *assms*(*2*) *enat-ord-code*(*4*)
lsum-addzero-NCons lsum-nlength lsum-nnth-leq-Suc nlength-NCons one-eSuc one-enat-def
zero-enat-def)
have *2*: *nat*>*0* ⟹ *?thesis*
using *Suc assms* **unfolding** *addzero-def* **by** *auto*
(*simp add*: *Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc*,
metis NCons Suc-ile-eq assms(*1*) *assms*(*2*) *enat-ord-code*(*4*) *iless-Suc-eq lsum-addzero-NCons*
lsum-nlength lsum-nnth-leq-Suc nlength-NCons)
show *?thesis* **using** *1 2* **by** *blast*
qed
qed
qed

lemma *lsum-nidx*:
assumes *all-gr-zero nells*
all-nfinite nells
shows *nidx* (*lsum nells a*)
using *assms* **unfolding** *nidx-expand*
by (*simp add*: *Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc*)

lemma *lsum-addzero-nidx*:
assumes *all-gr-zero nells*
all-nfinite nells
shows *nidx* (*addzero* (*lsum nells 0*))
using *assms* **unfolding** *nidx-expand*
using *Suc-ile-eq lsum-addzero-nnth-leq-Suc* **by** *blast*

lemma *pfilt-nfuse-lsum-a*:
assumes *nlast nell* = *nfirst nell1*
nlength nell > *0*
nlength nell1 > *0*
nfinite nell
nfinite nell1
shows *pfilt* (*nfuse nell nell1*) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*))) =
pfilt nell1 (*lsum* (*NNil nell1*) *0*)
proof −
have *1*: (*pfilt* (*nfuse nell nell1*) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*)))) =
nmap (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*)))
unfolding *pfilt-nmap* **by** *simp*
have *2*: (*pfilt* (*nell1*) ((*lsum* (*NNil nell1*) *0*))) = *nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*))
unfolding *pfilt-nmap* **by** *simp*
have *3*: *nlength* (*nmap* (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*)))) =

nlength (*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*)))
by (*simp add*: *lsum-nlength*)
have *4*: ⋀*j*. *j* ≤ *nlength* (*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*))) ⟶
nnth ((*nmap* (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*))))) *j* =
nnth ((*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*)))) *j*
by (*metis assms*(*1*) *assms*(*4*) *lsum-NNil ndropn-nfuse ndropn-nnth nellist.simps*(*14*) *plus-nat.add-0*)
have *5*: *nmap* (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*))) =
(*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*)))
using *3 4*
nellist-eq-nnth-eq[*of nmap* (*nnth* (*nfuse nell nell1*)) (*lsum* (*NNil nell1*) (*the-enat* (*nlength nell*)))
(*nmap* (*nnth nell1*) (*lsum* (*NNil nell1*) (*0*::*nat*)))]
by *presburger*
show *?thesis* **using** *5 1 2*
by *force*
qed

lemma *pfilt-nfusecat-lsum-a*:
assumes *nlastnfirst* (*NCons nell nells*)
all-gr-zero nells
all-nfinite nells
nlength nell > *0*
nfinite nell
shows (*pfilt* (*nfuse nell* (*nfusecat nells*)) (*lsum nells* (*the-enat* (*nlength nell*)))) =
(*pfilt* (*nfusecat nells*) ((*lsum nells 0*)))
proof −
have *1*: (*pfilt* (*nfuse nell* (*nfusecat nells*)) (*lsum nells* (*the-enat* (*nlength nell*)))) =
nmap (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*)))
unfolding *pfilt-nmap* **by** *simp*
have *2*: (*pfilt* (*nfusecat nells*) ((*lsum nells 0*))) = *nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*))
unfolding *pfilt-nmap* **by** *simp*
have *3*: *nlength* (*nmap* (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*)))) =
nlength (*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))
by (*simp add*: *lsum-nlength*)
have *4*: ⋀*j*. (*enat j*) ≤ *nlength* (*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*))) ⟹
nnth ((*nmap* (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*))))) *j* =
nnth ((*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))) *j*
proof −
fix *j*
assume *a*: (*enat j*) ≤ *nlength* (*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))
show *nnth* ((*nmap* (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*))))) *j* =
nnth ((*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))) *j*
proof −
have *5*: *nnth* ((*nmap* (*nnth* (*nfuse nell* (*nfusecat nells*))) (*lsum nells* (*the-enat* (*nlength nell*))))) *j* =
nnth (*nfuse nell* (*nfusecat nells*)) (*nnth* (*lsum nells* (*the-enat* (*nlength nell*))) *j*)
using *nnth-nmap*[*of j* (*lsum nells* (*the-enat* (*nlength nell*))) (*nnth* (*nfuse nell* (*nfusecat nells*)))]
a 3 **by** *auto*
have *6*: (*nnth* (*lsum nells* (*the-enat* (*nlength nell*))) *j*) =
the-enat (*nlength nell*) + (∑ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))
using *lsum-nnth*[*of j nells* (*the-enat* (*nlength nell*))]

by (*metis a assms*(*3*) *lsum-nlength nlength-nmap*)
have *7*: *nnth* ((*nmap* (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*)))) *j* =
nnth (*nfusecat nells*) (*nnth* (*lsum nells* (*0*::*nat*)) *j*)
using *nnth-nmap*[*of j* (*lsum nells* (*0*::*nat*)) (*nnth* (*nfusecat nells*))] **using** *a* **by** *auto*
have *8*: (*nnth* (*lsum nells* (*0*::*nat*)) *j*) = ($\sum$ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))
by (*metis* (*no-types*, *lifting*) *6 a add-0 add-diff-cancel-left′ assms*(*3*) *lsum-nlength*
lsum-nnth nlength-nmap)
have *9*: *nlast nell* = *nfirst* (*nfusecat nells*)
by (*metis assms*(*1*) *nfirst-nfusecat-nfirst nlastnfirst-LCons*)
have *10*: (*the-enat* (*nlength nell*) + ($\sum$ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))) ≤
nlength (*nfuse nell* (*nfusecat nells*))
proof (*cases nfinite nells*)
case *True*
then show *?thesis*
proof −
have *101*: *j* ≤ *nlength nells*
by (*metis a lsum-nlength nlength-nmap*)
have *11*: *nlength* (*nfusecat nells*) =
($\sum$ *i*::*nat* = *0*::*nat*..*the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
using *nfusecat-nlength-nfinite*[*of nells*]
by (*metis True assms*(*1*) *assms*(*3*) *nlastnfirst-LCons*)
have *12*: *nlength* (*nfuse nell* (*nfusecat nells*)) =
nlength nell + ($\sum$ *i*::*nat* = *0*::*nat*..*the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
using *nfuse-nlength*[*of nell nfusecat nells*] *11* **by** *presburger*
have *13*: ($\sum$ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*))) =
(*the-enat* ($\sum$ *k*::*nat* = *0*::*nat*..*j*. (*nlength* (*nnth nells k*))))
by (*metis 101 assms*(*3*) *sum-the-enat*)
have *14*: *j*< *the-enat* (*nlength nells*) ⟹
($\sum$ *k*::*nat* = *0*::*nat*..*j*. (*nlength* (*nnth nells k*))) ≤
($\sum$ *i*::*nat* = *0*::*nat*..*the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
by (*metis bot-nat-0.extremum canonically-ordered-monoid-add-class.lessE*
enat-le-plus-same(*1*) *sum.ub-add-nat*)
have *141*: *j*= *the-enat* (*nlength nells*) ⟹
($\sum$ *k*::*nat* = *0*::*nat*..*j*. (*nlength* (*nnth nells k*))) ≤
($\sum$ *i*::*nat* = *0*::*nat*..*the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
by *blast*
have *142*: ($\sum$ *k*::*nat* = *0*::*nat*..*j*. (*nlength* (*nnth nells k*))) ≤
($\sum$ *i*::*nat* = *0*::*nat*..*the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
using *14 141 101 True nfinite-nlength-enat* **by** *fastforce*
have *143*: *enat* (*the-enat* (*nlength nell*)) = *nlength nell*
by (*simp add*: *assms*(*5*) *enat-the-enat nfinite-nlength-enat*)
have *144*: *enat* (*the-enat* ($\sum$ *k* = *0*..*j*. *nlength* (*nnth nells k*))) =
($\sum$ *k* = *0*..*j*. *nlength* (*nnth nells k*))
by (*metis 11 142 True assms*(*1*) *assms*(*3*) *dual-order.refl enat-ord-code*(*4*)
enat-the-enat leD nfinite-nlength-enat nfusecat-nlength-nfinite
nlastnfirst-LCons sum-finite)
have *143*: *enat* (*the-enat* (*nlength nell*) + ($\sum$ *k* = *0*..*j*. *the-enat* (*nlength* (*nnth nells k*)))) =
nlength nell + ($\sum$ *k* = *0*..*j*. *nlength* (*nnth nells k*))
using *13*
by (*metis 143 144 plus-enat-simps*(*1*))

have *15*: (*the-enat* (*nlength nell*) + ($\sum$ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))) $\leq$
nlength nell + ($\sum$ *i*::*nat* = *0*::*nat*..*the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
using *142 6 8 13* **by** (*metis 143 add-left-mono*)
show *?thesis*
using *12 15* **by** *presburger*
qed
next
case *False*
then show *?thesis*
proof –
have *200*: ¬*nfinite* (*nfuse nell* (*nfusecat nells*))
using *assms* **by** (*metis False nfuse-nfinite nfusecat-nfinite nlastnfirst-LCons*)
show *?thesis*
by (*metis 200 linorder-le-cases nfinite-ntaken ntaken-all*)
qed
qed
have *30*: *nnth* (*nfuse nell* (*nfusecat nells*)) (*the-enat* (*nlength nell*) +
($\sum$ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))) =
nnth (*nfusecat nells*) ($\sum$ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))
by (*metis 9 assms*(*5*) *ndropn-nfuse ndropn-nnth*)
show *?thesis*
using *30 5 6 7 8* **by** *presburger*
qed
qed
show *?thesis*
by (*metis 3 4 nellist-eq-nnth-eq pfilt-nmap*)
qed

lemma *pfilt-nfusecat-lsum*:
assumes *nlastnfirst* (*NCons nell nells*)
all-gr-zero nells
all-nfinite nells
nlength nell > *0*
nfinite nell
shows (*pfilt* (*nfusecat* (*NCons nell nells*)) (*addzero* (*lsum* (*NCons nell nells*) *0*))) =
(*NCons* (*nfirst nell*) (*NCons* (*nlast nell*) (*pfilt* (*nfusecat nells*) (*lsum nells 0*))))
proof –
have *1*: *nfusecat* (*NCons nell nells*) = *nfuse nell* (*nfusecat nells*)
by *simp*
have *2*: (*pfilt* (*nfusecat* (*NCons nell nells*)) (*addzero* (*lsum* (*NCons nell nells*) *0*))) =
(*pfilt* (*nfuse nell* (*nfusecat nells*)) (*addzero* (*lsum* (*NCons nell nells*) *0*)))
using *1* **by** *simp*
have *3*: *addzero* (*lsum* (*NCons nell nells*) *0*) =
(*NCons 0* (*NCons* (*the-enat* (*nlength nell*)) (*lsum nells* (*the-enat* (*nlength nell*)))))
using *lsum-addzero-NCons* **by** *auto*
have *4*: (*pfilt* (*nfuse nell* (*nfusecat nells*)) (*addzero* (*lsum* (*NCons nell nells*) *0*))) =
(*pfilt* (*nfuse nell* (*nfusecat nells*)) (*NCons 0* (*NCons* (*the-enat* (*nlength nell*))
(*lsum nells* (*the-enat* (*nlength nell*))))))
using *3* **by** *auto*
have *5*: (*pfilt* (*nfuse nell* (*nfusecat nells*)) (*NCons 0* (*NCons* (*the-enat* (*nlength nell*))

(*lsum nells* (*the-enat* (*nlength nell*)))))) =
(*NCons* (*nnth* (*nfuse nell* (*nfusecat nells*)) *0*)
(*NCons* (*nnth* (*nfuse nell* (*nfusecat nells*)) (*the-enat* (*nlength nell*)))
(*pfilt* (*nfuse nell* (*nfusecat nells*)) (*lsum nells* (*the-enat* (*nlength nell*))))))
by *simp*
have *6*: (*nnth* (*nfuse nell* (*nfusecat nells*)) *0*) = (*nnth nell 0*)
using *assms*
by (*metis nfirst-nfusecat-nfirst nfuse-nnth nlastnfirst-LCons zero-enat-def zero-le*)
have *7*: (*nnth* (*nfuse nell* (*nfusecat nells*)) (*the-enat* (*nlength nell*))) =
(*nnth nell* (*the-enat* (*nlength nell*)))
using *assms*
by (*metis ndropn-nfirst ndropn-nfuse nfirst-nfusecat-nfirst nlastnfirst-LCons nnth-nlast*)
have *8*: *nidx* (*addzero* (*lsum nells 0*))
using *assms lsum-addzero-nidx* **by** *blast*
have *9*: (*pfilt* (*nfuse nell* (*nfusecat nells*)) (*lsum nells* (*the-enat* (*nlength nell*)))) =
(*pfilt* (*nfusecat nells*) (*lsum nells 0*))
using *assms pfilt-nfusecat-lsum-a* **by** *blast*
show *?thesis*
using *3 6 7 9*
by (*metis 2 5 assms*(*5*) *nlast-NNil nnth-nlast ntaken-0 ntaken-nlast*)
qed

lemma *pfilt-nfusecat-lsum-1*:
assumes *nlastnfirst* (*NCons nell nells*)
all-gr-zero nells
all-nfinite nells
nlength nell > *0*
nfinite nell
shows (*pfilt* (*nfusecat* (*NCons nell nells*)) ((*lsum* (*NCons nell nells*) *0*))) =
(*NCons* (*nlast nell*) (*pfilt* (*nfusecat nells*) (*lsum nells 0*)))
using *assms*
pfilt-nfusecat-lsum[*of nell nells*]
by (*metis lsum-addzero-NCons nellist.inject*(*2*) *pfilt-code*(*2*))

lemma *pfilt-nfusecat-lsum-2*:
assumes *nlastnfirst* (*nells*)
all-gr-zero nells
all-nfinite nells
$j \leq$ *nlength nells*
shows (*nnth* (*pfilt* (*nfusecat* (*nells*)) ((*lsum* (*nells*) *0*))) *j*) = *nlast*(*nnth nells j*)
proof −
have *1*: (*pfilt* (*nfusecat* (*nells*)) ((*lsum* (*nells*) *0*))) =
nmap (*nnth* (*nfusecat nells*)) (*lsum nells* (*0*::*nat*))
using *pfilt-nmap*[*of* (*nfusecat* (*nells*)) (*lsum* (*nells*) *0*)] **by** *simp*
have *2*: (*nnth* (*pfilt* (*nfusecat* (*nells*)) ((*lsum* (*nells*) *0*))) *j*) =
nnth (*nfusecat nells*) (*nnth* (*lsum nells* (*0*::*nat*)) *j*)
using *1 nnth-nmap*[*of j* ((*lsum* (*nells*) *0*)) (*nnth* (*nfusecat nells*))]
by (*simp add*: *assms*(*4*) *lsum-nlength*)
have *3*: (*nnth* (*lsum nells* (*0*::*nat*)) *j*) = ($\sum$ *k*::*nat* = *0*::*nat*..*j*. *the-enat* (*nlength* (*nnth nells k*)))
by (*metis add-0 assms*(*3*) *assms*(*4*) *lsum-nnth*)

have *4*: ($\sum$ *k::nat = 0::nat..j. the-enat* (*nlength* (*nnth nells k*))) =
the-enat (($\sum$ *k::nat = 0::nat..j.* (*nlength* (*nnth nells k*))))
by (*metis assms*(*3*) *assms*(*4*) *sum-the-enat*)
have *40*: *nfinite* (*ntaken j nells*)
by *simp*
have *41*: *nlastnfirst* (*ntaken j nells*)
by (*simp add*: *assms*(*1*) *assms*(*4*) *nlastnfirst-ntaken*)
have *42*: *all-nfinite* (*ntaken j nells*)
by (*metis assms*(*3*) *nset-ntaken subset-iff*)
have *5*: *nnth* (*nfusecat nells*) (*the-enat* (($\sum$ *k::nat = 0::nat..j.* (*nlength* (*nnth nells k*))))) =
nlast(*nnth nells j*)
using *nlastfirst-nfusecat-nlast*[*of ntaken j nells*] *nfusecat-ntake*[*of j nells*] *assms*
nlastnfirst-ntaken[*of j nells*] *ntake-eq-ntaken ntaken-nlast*[*of j nells*]
by (*metis 3 4 40 42 enat-ord-simps*(*4*) *enat-the-enat ntaken-nlast sum-finite*)
show *?thesis*
using *2 3 4 5* **by** *presburger*
qed

lemma *pfilt-nfusecat-lsum-3*:
assumes *nlastnfirst* (*nells*)
all-gr-zero nells
all-nfinite nells
$j \leq$ *nlength* (*addzero* (*lsum nells 0*))
shows (*j=0* $\longrightarrow$ (*nnth* (*pfilt* (*nfusecat* (*nells*)) (*addzero*(*lsum* (*nells*) *0*))) *j*) = *nfirst*(*nfirst nells*))
$\wedge$
(*j>0* $\longrightarrow$ (*nnth* (*pfilt* (*nfusecat* (*nells*)) (*addzero*(*lsum* (*nells*) *0*))) *j*) = *nlast*(*nnth nells* (*j−1*)))

using *assms*
proof −
have *1*: $j \leq$ *nlength* (*addzero* (*lsum nells 0*)) $\wedge$ *j=0* $\longrightarrow$
(*nnth* (*pfilt* (*nfusecat* (*nells*)) (*addzero*(*lsum* (*nells*) *0*))) *j*) = *nfirst*(*nfirst nells*)
using *assms*
by (*metis lsum-addzero-nfirst nfinite-ntaken nfirst-nfusecat-nfirst nnth-NNil nnth-nlast*
ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth)
have *2*: $j \leq$ *nlength* (*addzero* (*lsum nells 0*)) $\wedge$ *j>0* $\longrightarrow$
(*nnth* (*pfilt* (*nfusecat nells*) (*addzero*(*lsum* (*nells*) *0*))) *j*) =
(*nnth* (*nfusecat nells*) (*nnth* (*addzero*(*lsum* (*nells*) *0*)) *j*))
by (*simp add*: *pfilt-nlength pfilt-nnth*)
have *3*: $j \leq$ *nlength* (*addzero* (*lsum nells 0*)) $\wedge$ *j>0* $\longrightarrow$
nnth (*addzero*(*lsum* (*nells*) *0*)) *j* = *nnth* (*lsum nells 0*) (*j−1*)
by (*metis Suc-diff-1 addzero-def enat-0-iff*(*1*) *le-zero-eq nat-less-le nnth-Suc-NCons*)
have *20*: $j \leq$ *nlength* (*addzero* (*lsum nells 0*)) $\wedge$ *j>0* $\longrightarrow$ *enat* ((*j::nat*) − (*1::nat*)) $\leq$ *nlength nells*
by (*metis Suc-diff-1 Suc-ile-eq addzero-def dual-order.strict-trans1 enat-0-iff*(*1*) *iless-Suc-eq*
lsum-nlength nlength-NCons not-gr-zero)
have *4*: $j \leq$ *nlength* (*addzero* (*lsum nells 0*)) $\wedge$ *j>0* $\longrightarrow$
(*nnth* (*nfusecat nells*) (*nnth* (*lsum nells 0*) (*j−1*))) = *nlast*(*nnth nells* (*j −1*))
using *assms 20 pfilt-nfusecat-lsum-2*[*of nells j−1*]
by (*metis lsum-nlength pfilt-expand*)

show *?thesis*
using *1 2 3 4*
by (*simp add*: *assms*(*4*))
qed

lemma *pfilt-nfusecat-lsum-4*:
assumes *nlastnfirst nells*
all-gr-zero nells
all-nfinite nells
nfinite nells
shows (*nnth* (*addzero* (*lsum nells 0*)) (*the-enat* (*nlength* (*addzero* (*lsum nells 0*))))) $\leq$ *nlength* (*nfusecat nells*)
proof –
have *2*: *nlength* (*nfusecat nells*) = ($\sum$ *i* = *0*..*the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
using *assms nfusecat-nlength-nfinite* **by** *blast*
have *3*: *nlast* (*lsum nells 0*) =
($\sum$ *k* = *0*..*the-enat* (*nlength nells*). *the-enat* (*nlength* (*nnth nells k*)))
by (*simp add*: *assms*(*3*) *assms*(*4*) *lsum-nlast*)
have *4*: ($\sum$ *k* = *0*..*the-enat* (*nlength nells*). *the-enat* (*nlength* (*nnth nells k*))) =
the-enat($\sum$ *k* = *0*..*the-enat* (*nlength nells*). (*nlength* (*nnth nells k*)))
by (*simp add*: *assms*(*3*) *assms*(*4*) *enat-the-enat nfinite-nlength-enat sum-the-enat*)
have *5*: *enat* ($\sum$ *k* = *0*..*the-enat* (*nlength nells*). *the-enat* (*nlength* (*nnth nells k*))) $\leq$
($\sum$ *i* = *0*..*the-enat* (*nlength nells*). *nlength* (*nnth nells i*))
using *assms* **by** (*metis 4 dual-order.refl enat-ord-code*(*3*) *enat-the-enat*)
have *6*: *nlast* (*addzero* (*lsum nells 0*)) $\leq$ *nlength* (*nfusecat nells*)
unfolding *addzero-def* **using** *3 2 5*
by *simp*
have *7*: *nlast* (*addzero* (*lsum nells 0*)) =
(*nnth* (*addzero* (*lsum nells 0*)) (*the-enat* (*nlength* (*addzero* (*lsum nells 0*)))))
by (*simp add*: *addzero-def assms*(*4*) *nfinite-lsum-conv-a nnth-nlast*)
show *?thesis* **using** *assms 6 7* **by** *auto*
qed

lemma *nfusecat-nlength-b*:
assumes *nlastnfirst nells*
all-nfinite nells
1$\leq$ *i*
i$\leq$ *nlength nells*
j $\leq$ *nlength*(*nnth nells i*)
nfinite nells
all-gr-zero nells
shows (*nnth* ((*lsum nells 0*)) (*i*−*1*) + *j* $\leq$ *nlength* (*nfusecat nells*))
proof –
have *0*: *i*−*1* $\leq$ *nlength nells*
by (*metis Suc-ile-eq assms*(*3*) *assms*(*4*) *le-add-diff-inverse order-less-imp-le plus-1-eq-Suc*)
have *1*: (*nnth* ((*lsum nells 0*)) (*i*−*1*) = ($\sum$ *k*::*nat*= *0*..(*i*−*1*). *the-enat*(*nlength*(*nnth nells k*))))
using *0 lsum-nnth*[*of i*−*1 nells 0*] *assms* **by** *presburger*
have *2*: *nlength* (*nfusecat nells*) = ($\sum$ *k*::*nat*= *0*..(*the-enat*(*nlength nells*)). *nlength*(*nnth nells k*))
using *assms nfusecat-nlength-nfinite* **by** *metis*

have *20*: $(\sum k{::}nat{=}\ 0..(i).\ the\text{-}enat(nlength(nnth\ nells\ k))) =$
$(\sum k = 0..i - 1.\ the\text{-}enat\ (nlength\ (nnth\ nells\ k))) +$
$the\text{-}enat\ (nlength\ (nnth\ nells\ (Suc\ (i - 1))))$
using *assms sum.atLeast0-atMost-Suc*[*of* $\lambda k.$ *the-enat*(*nlength*(*nnth nells k*)) *i−1*] **by** *simp*
have *21*: $(\sum k = 0..i - 1.\ the\text{-}enat\ (nlength\ (nnth\ nells\ k))) < \infty$
using *enat-ord-code*(*4*) **by** *blast*
have *22*: $the\text{-}enat\ (nlength\ (nnth\ nells\ (Suc\ (i - 1)))) < \infty$
by *auto*
have *23*: $(\sum k{::}nat{=}\ 0..(i).\ the\text{-}enat(nlength(nnth\ nells\ k))) < \infty$
using *enat-ord-simps*(*4*) **by** *blast*
have *3*: $(\sum k{::}nat{=}\ 0..(i-1).\ the\text{-}enat(nlength(nnth\ nells\ k))) + j \leq$
$(\sum k{::}nat{=}\ 0..(i).\ the\text{-}enat(nlength(nnth\ nells\ k)))$
using *20 assms* **by** *simp*
(*metis* (*no-types*, *lifting*) *all-nfinite-nnth-b enat-ord-simps*(*1*) *enat-the-enat infinity-ileE*
ndropn-eq-NNil ndropn-nlast)
have *4*: $(\sum k{::}nat{=}\ 0..(i).\ the\text{-}enat(nlength(nnth\ nells\ k))) \leq$
$(\sum k{::}nat{=}\ 0..(the\text{-}enat(nlength\ nells)).\ the\text{-}enat(nlength(nnth\ nells\ k)))$
using *sum.ub-add-nat*[*of 0 i* λ *k. the-enat*(*nlength*(*nnth nells k*)) (*the-enat*(*nlength nells*))−*i*]
using *assms*(*4*) *assms*(*6*) *nfinite-nlength-enat* **by** *fastforce*
have *5*: $(\sum k{::}nat{=}\ 0..(the\text{-}enat(nlength\ nells)).\ the\text{-}enat(nlength(nnth\ nells\ k))) \leq$
$(\sum k{::}nat{=}\ 0..(the\text{-}enat(nlength\ nells)).\ nlength(nnth\ nells\ k))$
using *sum-the-enat*[*of nells* (*the-enat*(*nlength nells*))] *assms*
by (*metis leI less-enatE order-less-imp-not-eq the-enat.simps*)
show *?thesis*
using *1 2 3 4 5* **by** *simp*
(*meson dual-order.trans enat-ord-simps*(*1*))
qed

lemma *nfusecat-nlength-b-alt*:
assumes *nlastnfirst nells*
all-nfinite nells
$1 \leq i$
$i \leq nlength\ nells$
$j \leq nlength(nnth\ nells\ i)$
$\neg nfinite\ nells$
all-gr-zero nells
shows $nnth\ ((lsum\ nells\ 0)\ )\ (i-1) + j \leq nlength\ (nfusecat\ nells)$
proof −
have *1*: ¬*nfinite* (*nfusecat nells*)
using *assms* **by** (*metis nfusecat-nfinite*)
show *?thesis*
by (*meson 1 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
qed

lemma *pfilt-nfusecat-lsum-5*:
assumes *nlastnfirst nells*
all-gr-zero nells
all-nfinite nells
$(enat\ i) \leq nlength\ (addzero\ (lsum\ nells\ 0))$
nfinite nells

shows $(nnth\ (addzero\ (lsum\ nells\ 0))\ i) \leq nlength\ (nfusecat\ nells)$
using *assms*
proof −
have *2*: $nlength\ (nfusecat\ nells) = (\sum i = 0..the\text{-}enat\ (nlength\ nells).\ nlength\ (nnth\ nells\ i))$
using *assms nfusecat-nlength-nfinite* **by** *blast*
have *3*: $nlength\ nells > 0 \Longrightarrow nlength\ (lsum\ nells\ 0) > 0$
by (*simp add*: *lsum-nlength*)
have *4*: $nlength\ nells > 0 \Longrightarrow (nnth\ (addzero\ (lsum\ nells\ 0))\ i) = nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i$
using *assms* **unfolding** *addzero-def* **using** *3* **by** *simp*
have *5*: $nlength\ nells > 0 \land i{=}0 \Longrightarrow (nnth\ (addzero\ (lsum\ nells\ 0))\ i) = 0$
using *4* **by** *auto*
have *6*: $nlength\ nells > 0 \land i{>}0 \Longrightarrow (nnth\ (addzero\ (lsum\ nells\ 0))\ i) = nnth\ (lsum\ nells\ 0)\ (i - 1)$
by (*metis 4 Suc-diff-1 nnth-Suc-NCons*)
have *7*: $nlength\ nells > 0 \land i{>}0 \Longrightarrow nnth\ (lsum\ nells\ 0)\ (i - 1) =$
$(\sum k = 0..(i{-}1).\ the\text{-}enat\ (nlength\ (nnth\ nells\ k)))$
using *assms lsum-nnth*[*of i−1 nells 0*]
by (*metis Suc-diff-1 Suc-ile-eq add-0 addzero-def i0-less iless-Suc-eq lsum-nlength nlength-NCons*)
have *8*: $nlength\ nells > 0 \land i{>}0 \Longrightarrow$
$(\sum k = 0..(i{-}1).\ the\text{-}enat\ (nlength\ (nnth\ nells\ k))) =$
$the\text{-}enat(\sum k = 0..(i{-}1).\ (nlength\ (nnth\ nells\ k)))$
using *assms sum-the-enat*[*of nells i−1*]
by (*metis Suc-diff-1 Suc-ile-eq addzero-def i0-less iless-Suc-eq lsum-nlength nlength-NCons*)
have *9*: $nlength\ nells > 0 \land i{>}0 \Longrightarrow nnth\ (lsum\ nells\ 0)\ (i - 1) \leq nlength\ (nfusecat\ nells)$
using *assms nfusecat-nlength-b*[*of nells i 0*] *pfilt-nfusecat-lsum-4*[*of nells*]
3 4 6 lsum-nlength[*of nells*] **unfolding** *addzero-def* **by** *simp*
(*metis iless-Suc-eq order.order-iff-strict the-enat.simps zero-enat-def zero-le*)
have *10*: $nlength\ nells = 0 \land i = 0 \Longrightarrow nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i \leq nlength\ (nfusecat\ nells)$
using *assms zero-enat-def* **by** *auto*
have *11*: $nlength\ nells = 0 \land i = 1 \Longrightarrow nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i \leq nlength\ (nfusecat\ nells)$
using *assms*
by (*metis addzero-def lsum-nlength nlength-NCons not-one-le-zero one-eSuc one-enat-def pfilt-nfusecat-lsum-4 the-enat.simps*)
have *12*: *nfinite* (*nfirst nells*)
by (*metis assms*(*3*) *assms*(*5*) *nconcat-expand nfinite-nappend nfinite-nconcat*)
have *13*: $nlength\ (nfirst\ nells) > 0$
using *assms*
by (*metis in-nset-conv-nnth nlast-NNil ntaken-0 ntaken-nlast zero-enat-def zero-le*)
have *14*: $nlength\ nells = 0 \Longrightarrow i \leq 1$
using *assms lsum-addzero-nlength*[*of nells*]
by (*metis 12 13 enat-ord-simps*(*1*) *one-enat-def*)
have *15*: $nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i \leq nlength\ (nfusecat\ nells)$
by (*metis 10 11 14 4 5 6 9 One-nat-def Suc-leI dual-order.antisym not-gr-zero zero-enat-def zero-le*)
show *?thesis*
by (*metis 12 13 15 assms*(*4*) *gr-zeroI lsum-addzero-nnth*)
qed

lemma *pfilt-nfusecat-lsum-5-alt*:
assumes *nlastnfirst* (*nells*)

all-gr-zero nells
all-nfinite nells
(*enat i*) ≤ *nlength* (*addzero* (*lsum nells 0*))
¬*nfinite nells*
shows (*nnth* (*addzero* (*lsum nells 0*)) *i*) ≤ *nlength* (*nfusecat nells*)
using *assms*
by (*metis enat-ile linorder-le-cases nfinite-conv-nlength-enat nfusecat-nfinite*)

lemma *lsum-shift*:
assumes *nlastnfirst nells*
all-gr-zero nells
all-nfinite nells
i≤*nlength nells*
shows *nnth* (*lsum nells a*) *i* = *a*+ *nnth* (*lsum nells 0*) *i*
using *assms* **by** (*simp add*: *lsum-nnth*)

lemma *lsum-nfusecat-nnth-lsum-nnth*:
assumes *nlastnfirst nells*
all-gr-zero nells
all-nfinite nells
i≤*nlength nells*
j≤ *nlength*(*nnth nells i*)
shows (*nnth* (*nfusecat nells*) ((*nnth* (*addzero* (*lsum nells 0*)) *i*) +*j*)) = (*nnth* (*nnth nells i*) *j*)
using *assms*
proof (*induction i arbitrary*: *nells j*)
case *0*
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *0* **by** *simp*
(*metis add-cancel-right-left addzero-def ndropn-nfirst ndropn-nlast nfinite-NNil nlast-NNil nnth-0 nnth-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis* **using** *0* **by** *simp*
(*metis 0.prems*(*1*) *0.prems*(*2*) *0.prems*(*3*) *add-cancel-right-left addzero-def eSuc-ne-0 nfirst-nfusecat-nfirst nfuse-nnth nfusecat-NCons nfusecat-nlength-a nlength-NCons nnth-0*)
qed
next
case (*Suc i*)
then show *?case*
proof (*cases nells*)
case (*NNil x1*)
then show *?thesis* **using** *Suc* **by** (*simp add*: *zero-enat-def*)
next
case (*NCons x21 x22*)
then show *?thesis*
proof −

have *0*: (*0*::*enat*) < *nlength nells*
  by (*simp add*: *NCons*)
have *1*: *nlastnfirst x22*
 using *NCons Suc.prems*(*1*) **by** *auto*
have *2*: *all-gr-zero x22*
 using *NCons Suc.prems*(*2*) **by** *auto*
have *3*: *all-nfinite x22*
 using *NCons Suc.prems*(*3*) **by** *auto*
have *4*: *enat* (*i*::*nat*) $\leq$ *nlength x22*
 using *NCons Suc.prems*(*4*) *Suc-ile-eq* **by** *auto*
have *5*: *nnth* (*nfusecat nells*) (*nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*) + *j*) =
 nnth (*nfuse x21* (*nfusecat x22*)) (*nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*) + *j*)
 by (*simp add*: *NCons*)
have *6*: *nlength* (*addzero* (*lsum nells* (*0*::*nat*))) = *nlength nells* + (*1*::*enat*)
  using *lsum-addzero-nlength*[*of nells*]
  by (*metis NCons lsum-addzero-NCons lsum-nlength nlength-NCons plus-1-eSuc*(*2*))
have *7*: *enat* (*Suc* (*i*::*nat*)) $\leq$ *nlength* (*addzero* (*lsum* (*nells*::'*a nellist nellist*) (*0*::*nat*)))
  by (*simp add*: *6 Suc.prems*(*4*) *order-less-imp-le plus-1-eSuc*(*2*))
have *8*: (*nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*)) =
 nnth (*NCons* (*0*::*nat*) (*lsum nells* (*0*::*nat*))) (*Suc i*)
 using *lsum-addzero-nnth*[*of* (*Suc i*) *nells*] **by** (*metis NCons lsum-addzero-NCons*)
have *9*: *nnth* (*NCons* (*0*::*nat*) (*lsum nells* (*0*::*nat*))) (*Suc i*) = *nnth* (*lsum nells 0*) *i*
 by *simp*
have *10*: *nnth* (*lsum nells 0*) *i* = ($\sum$ *k*::*nat* = *0*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*)))
 using *lsum-nnth*[*of i nells 0*]
 by (*metis Suc.prems*(*3*) *Suc.prems*(*4*) *Suc-ile-eq add-0 order-less-imp-le*)
have *11*: *nlength x21* $\leq$ ($\sum$ *k*::*nat* = *0*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*))) + *j*
 proof (*cases i*)
 case *0*
 then show *?thesis* **using** *NCons* **by** *simp*
 (*metis Suc.prems*(*3*) *enat-ord-simps*(*1*) *enat-the-enat infinity-ileE le-add1 ndropn-eq-NNil*
 ndropn-nlast nellist.set-intros(*2*))
 next
 case (*Suc nat*)
 then show *?thesis*
  proof −
  have *12*: ($\sum$ *k*::*nat* = *0*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*))) =
  the-enat (*nlength* (*nnth nells 0*)) +
  ($\sum$ *k*::*nat* = *1*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*)))
   by (*simp add*: *sum.atLeast-Suc-atMost*)
  have *13*: (*nlength* (*nnth nells 0*)) = *nlength x21*
   by (*simp add*: *NCons*)
  have *14*: *nlength x21* $\leq$
  the-enat (*nlength x21*) +
  ($\sum$ *k*::*nat* = *1*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*))) + *j*
   using *NCons Suc.prems*(*3*) *nfinite-nlength-enat* **by** *fastforce*
  show *?thesis*
  using *12 13 14* **by** *presburger*
  qed
 qed

have *15*: *nlast x21* = *nfirst* (*nfusecat x22*)
by (*metis NCons Suc.prems*(*1*) *nfirst-nfusecat-nfirst nlastnfirst-LCons*)
have *16*: *enat* (*nnth* (*addzero* (*lsum nells 0*)) (*Suc i*) + *j*) ≤ *nlength* (*nfuse x21* (*nfusecat x22*))
by (*metis 8 9 NCons Suc.prems*(*1*) *Suc.prems*(*2*) *Suc.prems*(*3*) *Suc.prems*(*4*) *Suc.prems*(*5*)
add-diff-cancel-left′ le-add1 nfusecat-NCons nfusecat-nlength-b nfusecat-nlength-b-alt
plus-1-eq-Suc)
have *17*: *nlength x21* ≤ *enat* (*nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*) + *j*)
using *10 11 8 9* **by** *presburger*
have *18*: *nnth* (*nfuse x21* (*nfusecat x22*)) (*nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*) + *j*) =
nnth (*nfusecat x22*) (*nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*) + *j* − (*the-enat*(*nlength x21*)))
using *nfuse-nnth-var*[*of* (*nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*) + *j*) *x21 nfusecat x22*]
using *15 16 17* **by** *blast*
have *19*: *i=0* ⟹ *nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*) = (*the-enat*(*nlength x21*))
using *8 NCons* **by** *auto*
have *20*: *i=0* ⟹ *nnth* (*nfusecat x22*) *0* = *nnth* (*nnth x22 i*) *0*
using *Suc.IH*[*of x22 0*]
by (*metis 1 2 3 4 add.right-neutral all-gr-zero-nnth-b lsum-addzero-NCons*
lsum-addzero-nfirst nnth-0 ntaken-0 ntaken-nlast order-less-imp-le zero-enat-def)
have *21*: *i=0* ⟹ *?thesis*
by (*metis 1 18 19 2 3 4 5 NCons Suc.IH Suc.prems*(*5*) *add.commute add.right-neutral*
add-diff-cancel-left′ lsum-addzero-NCons lsum-addzero-nfirst nnth-0 nnth-Suc-NCons ntaken-0
ntaken-nlast)
have *22*: *0<i* ⟹ ($\sum$ *k*::*nat* = *0*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*))) =
(*the-enat*(*nlength x21*)) + ($\sum$ *k*::*nat* = *0*::*nat*..(*i*−*1*). *the-enat* (*nlength* (*nnth x22 k*)))
proof −
assume *a1*: *0<i*
have *23*: ($\sum$ *k*::*nat* = *0*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*))) =
the-enat(*nlength x21*) + ($\sum$ *k*::*nat* = *1*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*)))
by (*simp add*: *NCons sum.atLeast-Suc-atMost*)
have *24*: ($\sum$ *k*::*nat* = *1*::*nat*..*i*. *the-enat* (*nlength* (*nnth nells k*))) =
($\sum$ *k*::*nat* = *0*::*nat*..(*i*−*1*). *the-enat* (*nlength* (*nnth nells* (*k*+*1*))))
using *sum.shift-bounds-cl-nat-ivl*[*of* λ*k* .*the-enat* (*nlength* (*nnth nells k*)) *0 1 i*−*1*]
by (*simp add*: *a1*)
have *25*: ($\sum$ *k*::*nat* = *0*::*nat*..(*i*−*1*). *the-enat* (*nlength* (*nnth nells* (*k*+*1*)))) =
($\sum$ *k*::*nat* = *0*::*nat*..(*i*−*1*). *the-enat* (*nlength* (*nnth x22* (*k*))))
using *NCons* **by** *auto*
show *?thesis*
using *23 24 25* **by** *presburger*
qed
have *26*: *0<i* ⟹ *nnth* (*addzero* (*lsum nells* (*0*::*nat*))) (*Suc i*) − (*the-enat*(*nlength x21*)) =
($\sum$ *k*::*nat* = *0*::*nat*..(*i*−*1*). *the-enat* (*nlength* (*nnth x22* (*k*))))
by (*simp add*: *10 22 8*)
have *260*: *0* < *nfirst* (*lsum x22 0*)
proof (*cases x22*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
(*metis 2 3 NNil-eq-ntake-iff enat-the-enat gr0I infinity-ileE less-numeral-extra*(*3*)
ndropn-eq-NNil ndropn-nlast nellist.set-intros(*1*) *nlast-NNil zero-enat-def*)
next
case (*NCons x21 x22*)

then show *?thesis* **by** *simp*
(*metis 2 3 gr0I less-numeral-extra*(*3*) *ndropn-nfirst ndropn-nfuse ndropn-nlast nellist.set-intros*(*2*) *nfinite-NNil nfuse-leftneutral nlast-NNil nlength-NNil nnth-0 the-enat-0*)
qed
have *27*: $0<i \Longrightarrow (\sum k::nat = 0::nat..(i-1).\ the\text{-}enat\ (nlength\ (nnth\ x22\ (k)))) =$
$nnth\ (addzero\ (lsum\ x22\ 0))\ (i)$
unfolding *addzero-def* **using** *NCons 26 3 4 8 260* **apply** *simp*
using *lsum-nnth*
by (*metis Suc-ile-eq Suc-pred add-0 nnth-Suc-NCons order-less-imp-le*)
have *28*: $0<i \Longrightarrow$
$nnth\ (nfusecat\ x22)\ (nnth\ (addzero\ (lsum\ nells\ (0::nat)))\ (Suc\ i) + j - (the\text{-}enat(nlength\ x21)))$
$=$
$nnth\ (nnth\ x22\ i)\ j$
using *Suc.IH*[*of x22 j*]
by (*metis 1 10 2 22 26 27 3 4 8 NCons Nat.add-diff-assoc2 Suc.prems*(*5*) *le-add1 nnth-Suc-NCons*)
have *29*: $0<i \Longrightarrow$ *?thesis*
using *18 28 NCons* **by** *fastforce*
show *?thesis*
using *21 29* **by** *blast*
qed
qed
qed

lemma *lcppl-lsum-less-th-equal*:
assumes *nidx ls*
$nnth\ ls\ 0 = 0$
nfinite ls
nfinite σ
$nlast\ ls = (the\text{-}enat\ (nlength\ \sigma))$
$(\forall\ i<nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ \models f\ fproj\ g)$
$nlength\ \sigma > 0$
$i < nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
shows $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) \leq$
$nlength(\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
proof −
have *1*: *nlastnfirst* (*lcppl f g* σ *ls*)
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat i0-less lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *2*: *all-gr-zero* (*lcppl f g* σ *ls*)
using *assms all-gr-zero-nnth-a lcppl-nlength-all-gr-zero* **by** *blast*
have *3*: *all-nfinite* (*lcppl f g* σ *ls*)
using *assms all-nfinite-nnth-a lcppl-nlength-all-nfinite* **by** *blast*
have *4*: $enat\ (Suc\ (i::nat)) \leq nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ (0::nat)))$
using *assms* **using** *Suc-ile-eq* **by** *blast*
have *5*: *nfinite* (*lcppl f g* σ *ls*)
using *assms* **using** *lcppl-nfinite* **by** *blast*
show *?thesis* **using**

pfilt-nfusecat-lsum-5[*of* (*lcppl f g σ ls*) *Suc i*]
using *1 2 3 4 5* **by** *blast*
qed

lemma *lcppl-lsum-less-th-equal-alt*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite ls*
¬*nfinite σ*
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
i < *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
shows (*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) ≤
nlength((*nfusecat* (*lcppl f g σ ls*)))
proof −
have *1*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *2*: *all-gr-zero* (*lcppl f g σ ls*)
using *assms all-gr-zero-nnth-a lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *3*: *all-nfinite* (*lcppl f g σ ls*)
using *assms all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt* **by** *blast*
have *4*: *enat* (*Suc* (*i*::*nat*)) ≤ *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) (*0*::*nat*)))
using *assms* **using** *Suc-ile-eq* **by** *blast*
show *?thesis*
using *1 2 3 4 pfilt-nfusecat-lsum-5-alt* **using** *assms*(*1*) *assms*(*3*) *lcppl-nfinite* **by** *blast*
qed

lemma *lcppl-lsum-nlength*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ
nlast ls = (*the-enat* (*nlength σ*))
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ >*0*
shows *nlength* (*addzero* (*lsum* ((*lcppl f g σ ls*)) *0*)) = *nlength ls*
proof −
have *1*: *nlength σ* >*0* ⟶ *nlength* (*lcppl f g σ ls*) = *nlength ls* −*1*
using *assms*
by (*metis epred-0 epred-conv-minus i0-less lcppl-nlength lcppl-nlength-zero*)
have *2*: *nlength σ*>*0* ⟶ *nlength ls* >*0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *3*: *nlength σ* > *0* ⟶ *nlength* (*nfirst* (*lcppl f g σ ls*)) > *0*
using *assms*
by (*metis lcppl-nlength-all-gr-zero ndropn-0 ndropn-nfirst nfinite-nlength-enat the-enat.simps zero-enat-def zero-le*)
have *30*: *nlastnfirst* (*lcppl f g σ ls*)
using *2 assms lcppl-nfusecat-nlastnfirst* **by** *blast*

have *31*: *nfinite* (*nfirst* (*lcppl f g σ ls*))
  using *assms*
  by (*metis all-nfinite-nnth-a lcppl-nfinite lcppl-nlength-all-nfinite nconcat-expand nfinite-nappend nfinite-nconcat*)
have *4*: *nlength σ >0* ⟶ *nlength* (*lcppl f g σ ls*) = *0* ⟶
  nlength (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) = *nlength ls*
  using *1 2 3 assms lsum-addzero-nlength*[*of* (*lcppl f g σ ls*)] *30 31*
  by (*metis co.enat.exhaust-sel i0-less lcppl-nlength one-eSuc*)
have *5*: *nlength σ >0* ⟶ *nlength* (*lcppl f g σ ls*) > *0* ⟶
  nlength (*addzero* (*lsum* ((*lcppl f g σ ls*)) *0*)) = *nlength ls*
  using *assms*
  by (*metis 2 4 addzero-def co.enat.exhaust-sel lcppl-nlength lcppl-nlength-zero lsum-nlength nlength-NCons*)
show *?thesis* **using** *4 5*
using *assms*(*7*) *gr-zeroI* **by** *blast*
qed

lemma *lcppl-lsum-nlength-alt*:
assumes *nidx ls*
  nnth ls 0 = *0*
  ¬*nfinite ls*
  ¬*nfinite σ*
  (∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows *nlength* (*addzero* (*lsum* ((*lcppl f g σ ls*)) *0*)) = *nlength ls*
proof −
have *1*: *nlength* (*lcppl f g σ ls*) = *nlength ls* −*1*
  using *assms*
  by (*simp add*: *epred-conv-minus lcppl-nlength-alt*)
have *2*: *nlength ls >0*
  using *assms*
  by (*simp add*: *nfinite-conv-nlength-enat*)
have *3*: *nlength* (*nfirst* (*lcppl f g σ ls*)) > *0*
  using *assms lcppl-nlength-all-gr-zero-alt*[*of ls σ f g*]
  by (*metis ndropn-0 ndropn-nfirst zero-enat-def zero-le*)
have *4*: *nlength* (*lcppl f g σ ls*) = *0* ⟶
  nlength (*addzero* (*lsum* ((*lcppl f g σ ls*)) *0*)) = *nlength ls*
  using *1 2 3 assms*
  by (*metis lcppl-nfinite nlength-eq-enat-nfiniteD zero-enat-def*)
have *5*: *nlength* (*lcppl f g σ ls*) > *0* ⟶
  nlength (*addzero* (*lsum* ((*lcppl f g σ ls*)) *0*)) = *nlength ls*
  using *assms*
  by (*metis 4 addzero-def co.enat.exhaust-sel lcppl-nlength-alt lcppl-nlength-zero lsum-nlength nlength-NCons*)
show *?thesis* **using** *4 5*
using *assms gr-zeroI* **by** *blast*
qed

lemma *lcppl-lsum-nnth*:
assumes *nidx ls*
  nnth ls 0 = *0*

nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ *i<nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ > 0
j ≤ nlength (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
shows (*j=0* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
nfirst(*nfirst* (*lcppl f g σ ls*)))
∧
(*j>0* ⟶ (*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
nlast(*nnth* (*lcppl f g σ ls*) (*j−1*)))

proof −
have *0*: *nlength σ >0* ⟶ *nlength ls >0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *1*: *nlength σ >0* ⟶ *nlastnfirst* (*lcppl f g σ ls*)
using *0 assms*
by (*simp add*: *enat-the-enat lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat*)
have *2*: *nlength σ>0* ⟶
(∀ *j ≤ nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)
have *3*: *nlength σ>0* ⟶
nlength (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) =
nlength (*lcppl f g σ ls*) *+1*
using *assms*
by (*metis 0 co.enat.exhaust-sel i0-less lcppl-lsum-nlength lcppl-nlength plus-1-eSuc*(*2*))
have *4*: *nlength σ > 0* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *0*) =
nfirst(*nfirst* (*lcppl f g σ ls*))
by (*metis lsum-addzero-nfirst nfirst-nfusecat-nfirst nlast-NNil ntaken-0 ntaken-nlast pfilt-nnth zero-enat-def zero-le*)
have *5*: *nlength σ > 0* ⟶
j ≤ nlength (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) ∧ *j>0* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) (*j*))
= *nlast*(*nnth* (*lcppl f g σ ls*) (*j−1*))
using *assms pfilt-nfusecat-lsum-3*[*of* (*lcppl f g σ ls*) *j*]
by (*metis 1 2 all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite*)
from *4 5* **show** *?thesis* **using** *assms*
by *blast*
qed

lemma *lcppl-lsum-nnth-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬*nfinite ls*
¬*nfinite σ*
(∀ *i<nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)

j ≤ *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
shows (*j=0* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
nfirst(*nfirst* (*lcppl f g σ ls*)))
∧
(*j>0* ⟶ (*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
nlast(*nnth* (*lcppl f g σ ls*) (*j*−*1*)))
proof −
have *1*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *2*: (∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)
using *assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *3*: *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) = *nlength* (*lcppl f g σ ls*) +*1*
using *assms*
by (*simp add*: *lcppl-lsum-nlength-alt lcppl-nlength-alt nfinite-conv-nlength-enat*)
have *4*: (*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *0*) =
nfirst(*nfirst* (*lcppl f g σ ls*))
by (*metis lsum-addzero-nfirst nfirst-nfusecat-nfirst nlast-NNil ntaken-0 ntaken-nlast pfilt-nnth zero-enat-def zero-le*)
have *5*: *j* ≤ *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) ∧ *j>0* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) (*j*))
= *nlast*(*nnth* (*lcppl f g σ ls*) (*j*−*1*))
using *assms pfilt-nfusecat-lsum-3*[*of* (*lcppl f g σ ls*) *j*]
using *1 2 all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt* **by** *blast*
from *4 5* **show** *?thesis* **using** *assms*
by *blast*
qed

lemma *lcppl-lsum-nnth-a*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
(∀ *i<nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ>0
j ≤ *nlength ls*
shows (*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
(*nnth ls j*)
proof −
have *1*: *nlength σ>0* ⟶ *nlength ls>0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *2*: *nlength σ>0* ⟶ *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) = *nlength ls*
by (*simp add*: *assms lcppl-lsum-nlength*)
have *3*: *nlength σ>0* ⟶
j ≤ *nlength ls* ⟶
(*j=0* ⟶

$(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ \ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ \ ls))\ \ 0)))\ j) =$
$nfirst(nfirst\ \ (lcppl\ f\ g\ \sigma\ \ ls))\ )$
$\wedge$
$(j{>}0 \longrightarrow (nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ \ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ \ ls))\ \ 0)))\ j) =$
$nlast(nnth\ \ (lcppl\ f\ g\ \sigma\ \ ls)\ \ (j-1)))$
by (*metis 2 assms lcppl-lsum-nnth*)
have *4*: $nlength\ \sigma{>}0 \longrightarrow$
$j \le nlength\ ls \wedge j{=}0 \longrightarrow nfirst(nfirst\ \ (lcppl\ f\ g\ \sigma\ \ ls)) = (nnth\ ls\ j)$
using *assms lcppl-nfirst*[*of ls σ f g*]
by (*metis 1 nlast-NNil ntaken-0 ntaken-nlast*)
have *5*: $nlength\ \sigma{>}0 \longrightarrow$
$j \le nlength\ ls \wedge j{>}0 \longrightarrow j-1 \le nlength\ \ (lcppl\ f\ g\ \sigma\ \ ls)$
using *assms*
by (*metis 1 Suc-diff-1 Suc-ile-eq co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength*)
have *6*: $nlength\ \sigma{>}0 \longrightarrow$
$j \le nlength\ ls \wedge j{>}0 \longrightarrow nlast(nnth\ \ (lcppl\ f\ g\ \sigma\ \ ls)\ \ (j-1)) = (nnth\ ls\ j)$
using *lcppl-nlast-nnth assms*
by (*metis 5 Suc-diff-1 enat.distinct*(*2*) *enat-the-enat nfinite-conv-nlength-enat*)
show *?thesis*
using *3 4 6* **using** *assms*
by (*metis not-gr-zero*)
qed

lemma *lcppl-lsum-nnth-a-alt*:
assumes *nidx ls*
$nnth\ ls\ 0 = 0$
$\neg nfinite\ ls$
$\neg nfinite\ \sigma$
$(\forall\ \ i{<}nlength\ ls.\ (nsubn\ \sigma\ \ (nnth\ ls\ i)\ \ (nnth\ ls\ \ (Suc\ i))\ ) \models f\ \ fproj\ \ g)$
$j \le nlength\ ls$
shows $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ \ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ \ ls))\ \ 0)))\ j) =$
$(nnth\ ls\ j)$
proof −
have *2*: $nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ \ ls)\ \ 0)) = nlength\ ls$
by (*simp add*: *assms lcppl-lsum-nlength-alt*)
have *3*: $j \le nlength\ ls \longrightarrow$
$(\ j{=}0 \longrightarrow$
$(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ \ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ \ ls))\ \ 0)))\ j) =$
$nfirst(nfirst\ \ (lcppl\ f\ g\ \sigma\ \ ls))\ )$
$\wedge$
$(j{>}0 \longrightarrow (nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ \ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ \ ls))\ \ 0)))\ j) =$
$nlast(nnth\ \ (lcppl\ f\ g\ \sigma\ \ ls)\ \ (j-1)))$
by (*metis 2 assms lcppl-lsum-nnth-alt*)
have *4*: $j \le nlength\ ls \wedge j{=}0 \longrightarrow nfirst(nfirst\ \ (lcppl\ f\ g\ \sigma\ \ ls)) = (nnth\ ls\ j)$
using *assms lcppl-nfirst-alt*[*of ls σ f g*] **by** (*metis ndropn-0 ndropn-nfirst*)
have *5*: $j \le nlength\ ls \wedge j{>}0 \longrightarrow j-1 \le nlength\ \ (lcppl\ f\ g\ \sigma\ \ ls)$
using *assms*
by (*meson enat-ile lcppl-nfinite linorder-le-cases nfinite-conv-nlength-enat*)
have *6*: $j \le nlength\ ls \wedge j{>}0 \longrightarrow nlast(nnth\ \ (lcppl\ f\ g\ \sigma\ \ ls)\ \ (j-1)) = (nnth\ ls\ j)$
using *lcppl-nlast-nnth-alt assms*

by (*metis 5 Suc-pred'*)
show *?thesis*
using *3 4 6* **using** *assms*
by (*metis not-gr-zero*)
qed

lemma *lcppl-pfilt-nfusecat-lsum*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ)
(∀ *i<nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ>0
shows (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) = *ls*
using *assms*
proof −
have *0*: *nlength σ>0* ⟶ *nlength ls > 0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *1*: *nlength σ>0* ⟶
nlength (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) = *nlength ls*
using *assms*
by (*simp add*: *lcppl-lsum-nlength pfilt-nlength*)
have *2*: *nlength σ>0* ⟶
(∀ *j*. *j≤nlength ls* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
(*nnth ls j*))
by (*simp add*: *assms lcppl-lsum-nnth-a*)
from *1 2* **show** *?thesis* **using** *assms nellist-eq-nnth-eq*
by *metis*
qed

lemma *lcppl-pfilt-nfusecat-lsum-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬*nfinite ls*
¬*nfinite σ*
(∀ *i<nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) = *ls*
using *assms*
proof −
have *1*: *nlength* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) = *nlength ls*
using *assms* **by** (*simp add*: *lcppl-lsum-nlength-alt pfilt-nlength*)
have *2*: (∀ *j*. *j≤nlength ls* ⟶
(*nnth* (*pfilt* (*nfusecat* ((*lcppl f g σ ls*))) (*addzero*(*lsum* ((*lcppl f g σ ls*)) *0*))) *j*) =
(*nnth ls j*))
by (*simp add*: *assms lcppl-lsum-nnth-a-alt*)
from *1 2* **show** *?thesis* **using** *assms nellist-eq-nnth-eq*
by *metis*

qed

lemma *lcppl-nfusecat-pfilt-powerinterval*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat (*nlength* σ)
($\forall$ *i*<*nlength ls*. (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models$ *f fproj g*)
nlength σ >*0*
shows *powerinterval g* (*pfilt* σ (*nfusecat* (*lcppl f g* σ *ls*))) (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*))
proof −
have *0*: *nlength* σ > *0* $\longrightarrow$ *nlength ls* >*0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *01*: ($\forall$ *i*<*nlength ls*.
g (*pfilt* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *assms cppl-fprojection* **by** *auto*
have *02*: *nlength* σ > *0* $\longrightarrow$ ($\forall$ *i*<*nlength ls*. *g* (*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)))
using *0 assms lcppl-nfusecat-pfilt-fpower-help*
by (*metis enat.distinct*(*2*) *enat-the-enat nfinite-conv-nlength-enat*)
have *03* : *powerinterval g* (*pfilt* σ (*nfusecat* (*lcppl f g* σ *ls*))) (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) =
($\forall$ *i*<*nlength* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)).
g (*nsubn* (*pfilt* σ (*nfusecat* (*lcppl f g* σ *ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*))
))
by (*simp add*: *powerinterval-def*)
have *04*: *nlength* σ >*0* $\longrightarrow$ *nlength* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) = *nlength ls*
using *assms lcppl-lsum-nlength* **by** *blast*
have *05*: *nlength* σ >*0* $\longrightarrow$
($\forall$ *i*<*nlength ls*.
(*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)) =
(*nsubn* (*pfilt* σ (*nfusecat* (*lcppl f g* σ *ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*))
))

proof −
have *06*: *nlength* σ >*0* $\longrightarrow$
($\forall$ *i*<*nlength ls*. *nlength* (*pfilt* σ (*nnth* (*lcppl f g* σ *ls*) *i*)) =
nlength (*nnth* (*lcppl f g* σ *ls*) *i*))
using *pfilt-nlength* **by** *blast*
have *07*: *nlength* σ >*0* $\longrightarrow$
($\forall$ *i*<*nlength ls*. *nlength* (*nnth* (*lcppl f g* σ *ls*) *i*) =
nlength (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))))

using *assms* **by** (*simp add*: *lcppl-nnth*)
have *08*: *nlength* σ >*0* $\longrightarrow$
($\forall$ *i*<*nlength ls*.

nlength (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))) =
nlength (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))))

using *nlength-nmap* **by** *blast*
have *09*: *nlength* σ >*0* $\longrightarrow$
($\forall$ *i*<*nlength ls*.
nlength (*nsubn* (*pfilt* σ (*nfusecat* (*lcppl f g* σ *ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*))
) =
(*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*)) −
(*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) *i*))
using *assms*
by (*metis 04 enat-minus-mono1 idiff-enat-enat lcppl-lsum-less-th-equal min.orderE nsubn-nlength pfilt-nlength*)
have *10*: *nlength* σ >*0* $\longrightarrow$
($\forall$ *i*<*nlength ls*. (*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*)) =
(*nnth* (*NCons 0* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*)))

using *04 addzero-def* **by** *auto*
have *11*: *nlength* σ >*0* $\longrightarrow$
($\forall$ *i*<*nlength ls*. (*nnth* (*NCons 0* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*)) =
($\sum$ *k*::*nat*= *0*..(*i*). *nlength*(*nnth* (*lcppl f g* σ *ls*) *k*)))
using *04*
proof *simp-all*
assume *nlength* $\sigma \neq$ (*0*::*enat*) $\longrightarrow$ *nlength* (*addzero* (*lsum* (*lcppl f g* σ *ls*) (*0*::*nat*))) = *nlength ls*
show *nlength* $\sigma \neq$ (*0*::*enat*) $\longrightarrow$
($\forall$ *i*::*nat*. *enat i* < *nlength ls* $\longrightarrow$ *enat* (*nnth* (*lsum* (*lcppl f g* σ *ls*) (*0*::*nat*)) *i*) =
($\sum$ *k*::*nat* = *0*::*nat*..*i*. *nlength* (*nnth* (*lcppl f g* σ *ls*) *k*)))
proof
assume *nlength* $\sigma \neq$ (*0*::*enat*)
show ($\forall$ *i*::*nat*. *enat i* < *nlength ls* $\longrightarrow$
enat (*nnth* (*lsum* (*lcppl f g* σ *ls*) (*0*::*nat*)) *i*) =
($\sum$ *k*::*nat* = *0*::*nat*..*i*. *nlength* (*nnth* (*lcppl f g* σ *ls*) *k*)))
proof
fix *i*
show *i* < *nlength ls* $\longrightarrow$
enat (*nnth* (*lsum* (*lcppl f g* σ *ls*) (*0*::*nat*)) *i*) =
($\sum$ *k*::*nat* = *0*::*nat*..*i*. *nlength* (*nnth* (*lcppl f g* σ *ls*) *k*))
proof
assume *a*: *i* < *nlength ls*
show *enat* (*nnth* (*lsum* (*lcppl f g* σ *ls*) *0*) *i*) =
($\sum$ *k* = *0*..*i*. *nlength* (*nnth* (*lcppl f g* σ *ls*) *k*))
proof −
have *110*: *all-nfinite* (*lcppl f g* σ *ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite* **by** *blast*
have *111*: *nnth* (*lsum* (*lcppl f g* σ *ls*) *0*) *i* =
($\sum$ *k* = *0*..*i*. *the-enat* (*nlength* (*nnth* (*lcppl f g* σ *ls*) *k*)))
using *lsum-nnth*[*of i* (*lcppl f g* σ *ls*) *0*] *a assms*
by (*metis 0 110 add-cancel-right-left co.enat.exhaust-sel iless-Suc-eq*

lcppl-nlength not-gr-zero)
have *112*: ($\sum$ *k* = *0..i. the-enat* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*))) =
the-enat($\sum$ *k* = *0..i.* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))
using *sum-the-enat*[*of* (*lcppl f g σ ls*) *i*] *assms*
by (*metis 0 110 a co.enat.exhaust-sel gr-implies-not-zero iless-Suc-eq lcppl-nlength*)
have *113*: ($\sum$ *k* = *0..i.* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*))) < ∞
using *sum-finite*[*of* (*lcppl f g σ ls*) *i*] *0 110 assms a*
by (*metis co.enat.exhaust-sel gr-implies-not-zero iless-Suc-eq lcppl-nlength*)
have *114*: *enat* (*the-enat*($\sum$ *k* = *0..i.* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))) =
($\sum$ *k* = *0..i.* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))
using *113 enat-ord-simps*(*4*) *enat-the-enat* **by** *blast*
show *?thesis* **using** *111 112 114* **by** *presburger*
qed
qed
qed
qed
qed
have *12*: *nlength σ >0* ⟶
(∀ *i<nlength ls.* (*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
(*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *i*))
using *04 addzero-def* **by** *auto*
have *121*: *nlength σ >0* ⟶ (*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *0*) = *0*
by *simp*
have *13*: *nlength σ >0* ⟶
(∀ *i<nlength ls.* (*nnth* (*NCons 0* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
(*if i* = *0 then 0*
else ($\sum$ *k::nat*= *0..*(*i−1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))))

using *11*
by (*metis 121 Suc-diff-1 Suc-ile-eq not-gr-zero order-less-imp-le zero-enat-def*)
have *14*: *nlength σ >0* ⟶
(∀ *i<nlength ls.*
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) −
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) =
($\sum$ *k::nat*= *0..*(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(*if i* = *0 then 0 else*
($\sum$ *k::nat*= *0..*(*i−1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))))
using *10 11 12 13* **by** *simp* (*metis One-nat-def idiff-enat-enat not-gr-zero*)
have *15*: *nlength σ >0* ⟶
(∀ *i<nlength ls.*
($\sum$ *k::nat*= *0..*(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(*if i* = *0 then 0 else*
($\sum$ *k::nat*= *0..*(*i−1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))) =
(*if i* = *0 then nlength*(*nnth* (*lcppl f g σ ls*) *0*) *else*
($\sum$ *k::nat*= *0..*(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
($\sum$ *k::nat*= *0..*(*i−1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))))
by (*simp add*: *Nitpick.case-nat-unfold*)
have *16*: *nlength σ >0* ⟶
(∀ *i<nlength ls.*

(*if i = 0 then nlength* (*nnth* (*lcppl f g σ ls*) *0*) *else*
(∑ *k::nat= 0..*(*i*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)) −
(∑ *k::nat= 0..*(*i−1*). *nlength*(*nnth* (*lcppl f g σ ls*) *k*))) =
(*if i = 0 then nlength*(*nnth* (*lcppl f g σ ls*) *0*) *else*
nlength(*nnth* (*lcppl f g σ ls*) *i*)))
using *13 sum.cl-ivl-Suc*[*of λk. nlength*(*nnth* (*lcppl f g σ ls*) *k*) *0*]
by (*metis Suc-diff-1 enat.distinct*(*2*) *enat-add-sub-same less-nat-zero-code not-gr-zero*)
have *17*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(*if i = 0 then nlength*(*nnth* (*lcppl f g σ ls*) *0*) *else*
nlength(*nnth* (*lcppl f g σ ls*) *i*)) =
nlength(*nnth* (*lcppl f g σ ls*) *i*))
by (*simp add*: *Nitpick.case-nat-unfold*)
have *18*: *nlength σ >0 ⟶*
(∀ *i<nlength ls. nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
nlength (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))

by (*simp add*: *09 14 15 16 pfilt-nlength*)
have *19*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) ≤
nlength (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))))
by (*simp add*: *04 assms lcppl-lsum-less-th-equal pfilt-nlength*)
have *22*: *nlength σ >0 ⟶ nlastnfirst* (*lcppl f g σ ls*)
using *0 assms*
by (*simp add*: *enat-the-enat lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat*)
have *23*: *nlength σ >0 ⟶* (∀ *j ≤ nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)
have *190*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) ≤
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))
using *0 04 06 09 18 23 lsum-nlength*[*of* (*lcppl f g σ ls*) *0*] *lcppl-nlength*[*of ls f g σ*] *assms*
by *simp*
(*metis* (*no-types*, *lifting*) *co.enat.exhaust-sel diff-is-0-eq′ iless-Suc-eq le-cases zero-enat-def*)
have *20*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
) *j*) =
(*nnth* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))

((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))))
using *nsubn-nnth*[*of* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))] **by** *simp*
(*metis 06 09 18 assms*(*7*) *enat-ord-simps*(*1*) *min.orderE*)
have *21*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*)) =
(*nnth σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*)))))
using *pfilt-nlength*[*of σ* (*nfusecat* (*lcppl f g σ ls*))]
pfilt-nnth[*of - σ* (*nfusecat* (*lcppl f g σ ls*))]
by (*metis nnth-0 pfilt-code*(*2*) *pfilt-pfilt*)
have *24*: *nlength σ >0 ⟶*
(∀ *i≤nlength* (*lcppl f g σ ls*).
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
((*nnth* (*nfusecat* (*lcppl f g σ ls*)) ((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))) =
((*nnth* (*nnth* (*lcppl f g σ ls*) *i*) *j*))))

using *assms lsum-nfusecat-nnth-lsum-nnth*[*of* (*lcppl f g σ ls*)] *lcppl-nlength*[*of ls f g σ*]
0 06 09 18 22 23
by (*metis all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite*)
have *241*: *nlength σ >0 ⟶ nlength* (*lcppl f g σ ls*) = *nlength ls −1*
using *0 assms*
by (*simp add*: *epred-conv-minus lcppl-nlength*)
have *25*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))) =
(*nnth* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) *j*)))
using *0 04 24 lsum-nlength*[*of* (*lcppl f g σ ls*) *0*] *pfilt-nlength*[*of σ*] *pfilt-nnth*
by (*metis addzero-def i0-less iless-Suc-eq nlength-NCons*)
have *26*: *nlength σ >0 ⟶*
(∀ *i<nlength ls.*
(∀ *j≤ nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) *j*) =
(*nnth* (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
) *j*)))

by (*simp add*: *20 21 25*)
from *18 26* **show** *?thesis* **using** *nellist-eq-nnth-eq*
by (*metis 06*)
qed
show *?thesis*
using *02 03 04 05 assms*(*7*) **by** *force*
qed

lemma *lcppl-nfusecat-pfilt-powerinterval-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
¬nfinite σ
(∀ *i<nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows *powerinterval g* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
proof –
have *01*: (∀ *i<nlength ls*.
g (*pfilt* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *assms cppl-fprojection* **by** *auto*
have *02*: (∀ *i<nlength ls*. *g* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)))
using *assms lcppl-nfusecat-pfilt-fpower-help-alt*
by (*metis enat-the-enat nfinite-conv-nlength-enat*)
have *03* : *powerinterval g* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) =
(∀ *i<nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)).
g (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))
by (*simp add*: *powerinterval-def*)
have *04*: *nlength* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) = *nlength ls*
using *assms lcppl-lsum-nlength-alt* **by** *blast*
have *05*: (∀ *i<nlength ls*.
(*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
(*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))
proof –
have *06*:
(∀ *i<nlength ls*. *nlength* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) =
nlength (*nnth* (*lcppl f g σ ls*) *i*))
using *pfilt-nlength* **by** *blast*
have *07*: (∀ *i<nlength ls*. *nlength* (*nnth* (*lcppl f g σ ls*) *i*) =
nlength (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *assms* **by** (*simp add*: *lcppl-nnth*)
have *08*: (∀ *i<nlength ls*.
nlength (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) =
nlength (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))
using *nlength-nmap* **by** *blast*
have *09*: (∀ *i<nlength ls*.
nlength (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
) =
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*)) –
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*))
using *assms*
by (*metis 04 enat-minus-mono1 idiff-enat-enat lcppl-lsum-less-th-equal-alt min-def*

nsubn-nlength pfilt-nlength)
have *10*: ($\forall$ *i<nlength ls.* (*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*)) =
(*nnth* (*NCons 0* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*)))
using *04 addzero-def* **by** *auto*
have *11*: ($\forall$ *i<nlength ls.* (*nnth* (*NCons 0* (*lsum* (*lcppl f g* σ *ls*) *0*)) (*Suc i*)) =
($\sum$ *k::nat= 0..*(*i*). *nlength*(*nnth* (*lcppl f g* σ *ls*) *k*)))
using *04*
proof *simp-all*
assume *nlength* (*addzero* (*lsum* (*lcppl f g* σ *ls*) (*0::nat*))) = *nlength ls*
show ($\forall$ *i::nat. enat i* < *nlength ls* $\longrightarrow$ *enat* (*nnth* (*lsum* (*lcppl f g* σ *ls*) (*0::nat*)) *i*) =
($\sum$ *k::nat* = *0::nat..i. nlength* (*nnth* (*lcppl f g* σ *ls*) *k*)))
proof
fix *i*
show *i* < *nlength ls* $\longrightarrow$
enat (*nnth* (*lsum* (*lcppl f g* σ *ls*) (*0::nat*)) *i*) =
($\sum$ *k::nat* = *0::nat..i. nlength* (*nnth* (*lcppl f g* σ *ls*) *k*))
proof
assume *a*: *i* < *nlength ls*
show *enat* (*nnth* (*lsum* (*lcppl f g* σ *ls*) *0*) *i*) =
($\sum$ *k* = *0..i. nlength* (*nnth* (*lcppl f g* σ *ls*) *k*))
proof –
have *110*: *all-nfinite* (*lcppl f g* σ *ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt* **by** *blast*
have *111*: *nnth* (*lsum* (*lcppl f g* σ *ls*) *0*) *i* = ($\sum$ *k* = *0..i. the-enat* (*nlength* (*nnth* (*lcppl f g* σ *ls*) *k*)))
using *lsum-nnth*[*of i* (*lcppl f g* σ *ls*) *0*] *a assms*
by (*metis 04 110 add-cancel-right-left addzero-def iless-Suc-eq lsum-nlength nlength-NCons nlength-eq-enat-nfiniteD zero-enat-def*)
have *112*: ($\sum$ *k* = *0..i. the-enat* (*nlength* (*nnth* (*lcppl f g* σ *ls*) *k*))) =
the-enat($\sum$ *k* = *0..i.* (*nlength* (*nnth* (*lcppl f g* σ *ls*) *k*)))
using *sum-the-enat*[*of* (*lcppl f g* σ *ls*) *i*] *assms*
by (*metis 110 lcppl-nfinite linorder-le-cases nfinite-ntaken ntaken-all*)
have *113*: ($\sum$ *k* = *0..i.* (*nlength* (*nnth* (*lcppl f g* σ *ls*) *k*))) < ∞
using *sum-finite*[*of* (*lcppl f g* σ *ls*) *i*] *110 04 assms a*
by (*metis addzero-def iless-Suc-eq lsum-nlength nlength-NCons nlength-eq-enat-nfiniteD zero-enat-def*)
have *114*: *enat* (*the-enat*($\sum$ *k* = *0..i.* (*nlength* (*nnth* (*lcppl f g* σ *ls*) *k*)))) =
($\sum$ *k* = *0..i.* (*nlength* (*nnth* (*lcppl f g* σ *ls*) *k*)))
using *113 enat-ord-simps*(*4*) *enat-the-enat* **by** *blast*
show *?thesis* **using** *111 112 114* **by** *presburger*
qed
qed
qed
qed
have *12*: ($\forall$ *i<nlength ls.* (*nnth* (*addzero* (*lsum* (*lcppl f g* σ *ls*) *0*)) *i*) =
(*nnth* (*NCons 0* (*lsum* (*lcppl f g* σ *ls*) *0*)) *i*))
using *04 addzero-def* **by** *auto*
have *121*: (*nnth* (*NCons 0* (*lsum* (*lcppl f g* σ *ls*) *0*)) *0*) = *0*
by *simp*
have *13*: ($\forall$ *i<nlength ls.* (*nnth* (*NCons 0* (*lsum* (*lcppl f g* σ *ls*) *0*)) *i*) =

(*if* $i = 0$ *then* 0
else $(\sum k::nat= 0..(i-1).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k))$))
using *11*
by (*metis 121 Suc-diff-1 Suc-ile-eq not-gr-zero order-less-imp-le zero-enat-def*)
have *14*: $(\forall\ i<nlength\ ls.$
$(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) -$
$(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i) =$
$(\sum k::nat= 0..(i).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)) -$
(*if* $i = 0$ *then* 0 *else*
$(\sum k::nat= 0..(i-1).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k))$))

using *10 11 12 13* **by** *simp* (*metis One-nat-def idiff-enat-enat not-gr-zero*)
have *15*: $(\forall\ i<nlength\ ls.$
$(\sum k::nat= 0..(i).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)) -$
(*if* $i = 0$ *then* 0 *else*
$(\sum k::nat= 0..(i-1).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k))$) =
(*if* $i = 0$ *then* $nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ 0)$ *else*
$(\sum k::nat= 0..(i).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)) -$
$(\sum k::nat= 0..(i-1).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k))$))

by (*simp add*: *Nitpick.case-nat-unfold*)
have *16*: $(\forall\ i<nlength\ ls.$
(*if* $i = 0$ *then* $nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ 0)$ *else*
$(\sum k::nat= 0..(i).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)) -$
$(\sum k::nat= 0..(i-1).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k))$) =
(*if* $i = 0$ *then* $nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ 0)$ *else*
$nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)$))
using *13* *sum.cl-ivl-Suc*[*of* $\lambda k.\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)\ 0$]
by (*metis Suc-diff-1 enat.distinct*(*2*) *enat-add-sub-same less-nat-zero-code not-gr-zero*)
have *17*: $(\forall\ i<nlength\ ls.$
(*if* $i = 0$ *then* $nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ 0)$ *else*
$nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)$) =
$nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i))$
by (*simp add*: *Nitpick.case-nat-unfold*)
have *18*: $(\forall\ i<nlength\ ls.\ nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)) =$
$nlength\ (nsubn\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
$(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i)$
$(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i))$
))
by (*simp add*: *09 14 15 16 pfilt-nlength*)
have *19*: $(\forall\ i<nlength\ ls.$
$(\forall\ j\le nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i).$
$(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) \le$
$nlength\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))))$
by (*simp add*: *04 assms lcppl-lsum-less-th-equal-alt pfilt-nlength*)
have *22*: $nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$
using *assms lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have *23*: $(\forall\ j \le nlength\ (lcppl\ f\ g\ \sigma\ ls).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) > 0)$
using *assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *190*: $(\forall\ i<nlength\ ls.$

(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) ≤
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
))
using *04 06 09 18 23 lsum-nlength*[*of* (*lcppl f g σ ls*) *0*] *lcppl-nlength-alt*[*of ls f g σ*] *assms*
by *simp*
(*metis* (*no-types*, *opaque-lifting*) *co.enat.exhaust-sel diff-is-0-eq′ iless-Suc-eq nfinite-ntaken not-le-imp-less ntaken-all order-less-imp-le zero-enat-def*)
have *20*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))
) *j*) =
(*nnth* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))))
using *nsubn-nnth*[*of* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))] **by** *simp*
(*metis 06 09 18 enat-ord-simps*(*1*) *min.orderE*)
have *21*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*)) =
(*nnth σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*)))))
using *pfilt-nlength*[*of σ* (*nfusecat* (*lcppl f g σ ls*))]
pfilt-nnth[*of - σ* (*nfusecat* (*lcppl f g σ ls*))]
by (*metis enat-ord-code*(*4*) *enat-the-enat not-le-imp-less ntaken-all ntaken-nlast order-less-imp-le*)
have *24*: (∀ *i*≤*nlength* (*lcppl f g σ ls*).
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
((*nnth* (*nfusecat* (*lcppl f g σ ls*)) ((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))) =
((*nnth* (*nnth* (*lcppl f g σ ls*) *i*) *j*))))
using *assms lsum-nfusecat-nnth-lsum-nnth*[*of* (*lcppl f g σ ls*)] *04 22 23*
all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt **by** *blast*
have *241*: *nlength* (*lcppl f g σ ls*) = *nlength ls* −*1*
using *assms* **by** (*simp add*: *epred-conv-minus lcppl-nlength-alt*)
have *25*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*))
((*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*) +*j*))) =
(*nnth* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) *j*)))
using *04 24 lsum-nlength*[*of* (*lcppl f g σ ls*) *0*] *pfilt-nlength*[*of σ*] *pfilt-nnth*
assms(*3*)
by (*metis 241 enat2-cases enat-ord-code*(*3*) *epred-Infty epred-conv-minus*
nfinite-conv-nlength-enat)
have *26*: (∀ *i*<*nlength ls*.
(∀ *j*≤ *nlength*(*nnth* (*lcppl f g σ ls*) *i*).
(*nnth* (*pfilt σ* (*nnth* (*lcppl f g σ ls*) *i*)) *j*) =
(*nnth* (*nsubn* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *i*)
(*nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) (*Suc i*))

) *j*)))
by (*simp add*: *20 21 25*)
from *18 26* **show** *?thesis* **using** *nellist-eq-nnth-eq*
by (*metis 06*)
qed
show *?thesis*
using *02 03 04 05 assms* **by** *force*
qed

lemma *lcppl-nfusecat-pfilt-nlength*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat (nlength σ)
(∀ *i<nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ >0
shows *nlast* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) =
the-enat(*nlength* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))))
proof −
have *0*: *nlength ls > 0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *1*: *nlength* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) = *nlength* ((*nfusecat* (*lcppl f g σ ls*)))
using *pfilt-nlength* **by** *blast*
have *10*: *nfinite* (*lcppl f g σ ls*)
using *assms*(*1*) *assms*(*3*) *lcppl-nfinite* **by** *blast*
have *11*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms*
by (*metis 0 lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat*)
have *12*: ∀ *lx* ∈ *nset* (*lcppl f g σ ls*). *0* < *nlength lx*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat in-nset-conv-nnth lcppl-nlength-all-gr-zero*
nfinite-conv-nlength-enat)
have *13*: ∀ *lx* ∈ *nset* (*lcppl f g σ ls*). *nfinite lx*
by (*metis* (*no-types*, *lifting*) *0 assms*(*1*) *assms*(*3*) *assms*(*6*) *co.enat.exhaust-sel*
cppl-fprojection i0-less iless-Suc-eq in-nset-conv-nnth lcppl-nlength lcppl-nnth nfinite-nmap)
have *141*: ∀ *i* ≤ *nlength* (*lcppl f g σ ls*). *0* < *nlength* (*nnth* (*lcppl f g σ ls*) *i*)
using *12 assms*
by (*metis enat.distinct*(*2*) *enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)
have *142*: ∀ *i* ≤ *nlength* (*lcppl f g σ ls*). *nfinite* (*nnth* (*lcppl f g σ ls*) *i*)
using *13 assms*
by (*meson in-nset-conv-nnth*)
have *15*: *nfinite* (*nfusecat* (*lcppl f g σ ls*))
using *nfusecat-nfinite*[*of* (*lcppl f g σ ls*)] *10 11 12 13* **by** *blast*
have *2*: *nlength* ((*nfusecat* (*lcppl f g σ ls*))) =
(*the-enat* ($\sum$ *k*::*nat*= *0*..(*the-enat*(*nlength* (*lcppl f g σ ls*))). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)))
using *0 assms*
by (*metis 10 11 13 15 nfinite-conv-nlength-enat nfusecat-nlength-nfinite the-enat.simps*)
have *3*:

nlast(*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) = *nlast*((*lsum* (*lcppl f g σ ls*) *0*))
using *lsum-addzero-nlast*
using *assms*(*1*) *assms*(*3*) *lcppl-nfinite* **by** *blast*
have *4*: *nlast*((*lsum* (*lcppl f g σ ls*) *0*)) =
($\sum$ *k*::*nat* = *0*::*nat*..*the-enat* (*nlength* (*lcppl f g σ ls*)). *the-enat* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*)))
using *assms lsum-nlast*[*of* (*lcppl f g σ ls*) *0*] *10 13 plus-nat.add-0* **by** *presburger*
have *5*: ($\sum$ *k*::*nat* = *0*..*the-enat* (*nlength* (*lcppl f g σ ls*)). *the-enat* (*nlength* (*nnth* (*lcppl f g σ ls*) *k*))) =
(*the-enat* ($\sum$ *k*::*nat* = *0*..*the-enat*(*nlength* (*lcppl f g σ ls*)). *nlength*(*nnth* (*lcppl f g σ ls*) *k*)))
using *assms lsum-nnth*[*of the-enat* (*nlength* (*lcppl f g σ ls*)) (*lcppl f g σ ls*) *0*]
sum-the-enat[*of* (*lcppl f g σ ls*) *the-enat*(*nlength* (*lcppl f g σ ls*))]
by (*metis 13 dual-order.refl enat-ord-code*(*3*) *enat-the-enat*)
show *?thesis* **using** *1 2 3 4 5 assms*
by (*simp add*: *lcppl-lsum-nlength nfinite-conv-nlength-enat*)
qed

10.3 Soundness of Projection Axioms

lemma *PJ00sem*:
σ ⊨ ¬(*f fproj inf*)
by (*auto simp add*: *fprojection-d-def infinite-defs nfinite-conv-nlength-enat pfilt-nlength*)

lemma *OPJ00sem*:
σ ⊨ ¬(*f oproj finite*)
by (*auto simp add*: *oprojection-d-def finite-defs nfinite-conv-nlength-enat pfilt-nlength*)

lemma *PJ01sem*:
σ ⊨ (*f fproj g*) ⟶ *finite*
by (*auto simp add*: *fprojection-d-def finite-defs nfinite-conv-nlength-enat pfilt-nlength*)

lemma *OPJ01sem*:
σ ⊨ (*f oproj g*) ⟶ *inf*
by (*auto simp add*: *oprojection-d-def infinite-defs nfinite-conv-nlength-enat pfilt-nlength*)

lemma *PJ02sem*:
σ ⊨ (*f fproj g*) = ((*f* ∧ *finite*) *fproj g*)
by (*auto simp add*: *fprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)
(*metis enat-ord-simps*(*2*) *nfinite-nlength-enat nsubn-nfinite*)

lemma *OPJ02sem*:
σ ⊨ (*f oproj g*) = ((*f* ∧ *finite*) *oproj g*)
by (*auto simp add*: *oprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)
(*metis nfinite-conv-nlength-enat nfinite-ntaken nsubn-def1*)

lemma *PJ03sem*:
σ ⊨ (*f fproj g*) = (*f fproj* (*g* ∧ *finite*))
by (*auto simp add*: *fprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)

lemma *OPJ03sem*:
$\sigma \models (f\ oproj\ g) = (f\ oproj\ (g \wedge inf))$
by (*auto simp add*: *oprojection-d-def infinite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)

10.3.1 PJ1

lemma *PJ1sem*:
$(\sigma \models f\ \ fproj\ \ (\ g \vee h) = (f\ \ fproj\ \ g \vee f\ \ fproj\ \ h))$
by (*simp add*: *fprojection-d-def*) *blast*

lemma *OPJ1sem*:
$(\sigma \models f\ oproj\ (\ g \vee h) = (f\ oproj\ g \vee f\ oproj\ h))$
by (*simp add*: *oprojection-d-def*) *blast*

10.3.2 PJ2

lemma *PJ2sem*:
$(\sigma \models f\ \ fproj\ \ empty = empty)$
proof *auto*
show $(\sigma \models f\ \ fproj\ \ empty) \Longrightarrow \sigma \models empty$
unfolding *fprojection-d-def empty-defs*
by (*metis enat.distinct*(*2*) *enat-the-enat nfinite-nlength-enat nnth-nlast pfilt-nlength the-enat-0*)
show $\sigma \models empty \Longrightarrow (\sigma \models f\ \ fproj\ \ empty)$
unfolding *fprojection-d-def empty-defs powerinterval-def nidx-expand*
by (*metis enat-ord-simps*(*2*) *leD nfinite-conv-nlength-enat nlast-NNil nlength-NNil nnth-NNil not-iless0 pfilt-nlength the-enat-0 zero-enat-def zero-less-Suc*)
qed

lemma *OPJ2sem*:
$(\sigma \models \neg(f\ oproj\ empty))$
unfolding *oprojection-d-def empty-defs pfilt-nlength*
using *nlength-eq-enat-nfiniteD* **by** (*simp add*: *zero-enat-def*) *blast*

10.3.3 PJ3

lemma *PJ3help*:
assumes *nfinite* σ
shows *nsubn* σ *0* (*the-enat* (*nlength* σ)) $= \sigma$
using *assms*
by (*metis diff-zero dual-order.refl ndropn-0 nfinite-conv-nlength-enat nsubn-def1 ntaken-all the-enat.simps*)

lemma *PJ3help1*:
assumes $f\ \sigma \wedge 0 < nlength\ \sigma \wedge nfinite\ \sigma$
shows $(\exists l.\ nidx\ \ l \wedge nnth\ l\ 0 = 0 \wedge nfinite\ l \wedge nfinite\ \sigma \wedge$
$nlast\ l = (the\text{-}enat\ (nlength\ \sigma)) \wedge$
$(\forall i{<}nlength\ l.\ f\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))\ )) \wedge$
$(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ l \wedge$
$(\forall i{\le}nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ l\ i)) \wedge nlength\ \sigma 1 = Suc\ 0))$

proof –
have *1*: *nidx* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ))))
using *assms* **unfolding** *nidx-expand* **by** *simp*
(*metis Suc-ile-eq assms enat-0-iff*(*1*) *enat-ord-simps*(*2*) *iless-eSuc0 nfinite-nlength-enat nnth-0 nnth-NNil the-enat.simps*)
have *2*: *nnth* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ)))) ((*the-enat*(*nlength* (*NCons 0* (*NNil* (*nlength* σ)))))) =
(*the-enat* (*nlength* σ))
by (*metis nfinite-NCons nfinite-NNil nlast-NCons nlast-NNil nlength-NCons nlength-NNil nnth-nlast*)
have *3*: ($\forall$ *i*<*nlength* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ)))).
f (*nsubn* σ (*nnth* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ)))) *i*)
(*nnth* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ)))) (*Suc i*))))
using *assms PJ3help*
by (*metis enat-0-iff*(*1*) *iless-eSuc0 nlength-NCons nlength-NNil nnth-0 nnth-NNil nnth-Suc-NCons*)
have *4*: *nlength* (*NCons* (*nnth* σ (*0*)) (*NNil* (*nnth* σ (*the-enat*(*nlength* σ))))) =
nlength (*NCons 0* (*NNil* (*the-enat*(*nlength* σ))))
by *simp*
have *5*: ($\forall$ *i*$\leq$*nlength* (*NCons* (*nnth* σ (*0*)) (*NNil* (*nnth* σ (*the-enat*(*nlength* σ))))).
nnth (*NCons* (*nnth* σ (*0*)) (*NNil* (*nnth* σ (*the-enat*(*nlength* σ))))) *i* =
nnth σ (*nnth* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ)))) *i*))
by (*metis iless-Suc-eq le-zero-eq ndropn-nlast nfinite-NCons nfinite-NNil nlast-NCons nlength-NCons nlength-NNil nnth-0 nnth-NNil nnth-zero-ndropn order-neq-le-trans the-enat.simps the-enat-0*)
have *6*: *nlength* (*NCons* (*nnth* σ (*0*)) (*NNil* (*nnth* σ (*the-enat*(*nlength* σ))))) = *Suc 0*
using *one-eSuc one-enat-def* **by** *auto*
have *7*: *nfinite* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ))))
by *simp*
have *8*: *nlast* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ)))) = (*the-enat* (*nlength* σ))
by *simp*
have *9*: *nnth* (*NCons 0* (*NNil* (*the-enat*(*nlength* σ)))) *0* = *0*
using *nnth-0* **by** *simp*
show *?thesis*
using *1 2 3 4 5 6 7 8 9 assms* **by** *blast*
qed

lemma *PJ3sem*:
($\sigma \models f$ *fproj* *skip* = (*f* $\wedge$ *more* $\wedge$ *finite*))
proof –
have *1*: ($\sigma \models f$ *fproj* *skip*) $\Longrightarrow$ ($\sigma \models f \wedge$ *more* $\wedge$ *finite*)
unfolding *cppl-fprojection pfilt-expand nidx-expand skip-defs more-defs finite-defs powerinterval-def*
by (*metis* (*no-types*, *lifting*) *One-nat-def PJ3help eSuc-enat i0-less ileI1 intensional-rews*(*3*) *less-numeral-extra*(*1*) *less-numeral-extra*(*3*) *nnth-nlast pfilt-nlength the-enat.simps zero-enat-def*)
have *2*: ($\sigma \models f \wedge$ *more* $\wedge$ *finite*) $\Longrightarrow$ ($\sigma \models f$ *fproj* *skip*)
by (*simp add*: *fprojection-d-def finite-defs skip-defs more-defs powerinterval-def pfilt-nlength*)
(*metis PJ3help1 i0-less nfinite-conv-nlength-enat the-enat.simps*)
show *?thesis*
using *1 2 unl-lift2* **by** *blast*
qed

lemma *OPJ3sem*:
($\sigma \models \neg$(*f oproj skip*))
unfolding *oprojection-d-def skip-defs pfilt-nlength*
using *nlength-eq-enat-nfiniteD* **by** *auto*

10.3.4 PJ4

lemma *PJ4semchaina*:
assumes ($\sigma \models f$ *fproj* (*g*;*h*))
shows ($\sigma \models$ (*f fproj g*) ; (*f fproj h*))
proof –
have *1*: ($\sigma \models f$ *fproj* (*g*;*h*))
using *assms* **by** *auto*
have *2*: ($\exists$ *ls*. *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ *nfinite ls* $\wedge$ *nfinite* σ $\wedge$
nlast ls = (*the-enat* (*nlength* σ)) $\wedge$
powerinterval f σ *ls* $\wedge$
($\exists$ *n*$\le$*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*))))
using *assms* **unfolding** *chop-defs fprojection-d-def*
by (*metis nfinite-conv-nlength-enat pfilt-nlength the-enat.simps*)
obtain *ls* **where** *3*: *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ *nfinite ls* $\wedge$ *nfinite* σ $\wedge$
nlast ls = (*the-enat* (*nlength* σ)) $\wedge$
powerinterval f σ *ls* $\wedge$
($\exists$ *n*$\le$*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*)))
using *2* **by** *auto*
have *4*: *nidx ls* $\wedge$ *nnth ls 0* = *0*
using *3* **by** *auto*
have *5*: *powerinterval f* σ *ls*
using *3* **by** *auto*
have *6*: *nlast ls* = *the-enat* (*nlength* σ)
using *3* **by** *auto*
have *7*: ($\exists$ *n*$\le$*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*)))
using *3* **by** *auto*
obtain *n* **where** *8*: *n*$\le$*nlength* (*pfilt* σ *ls*) $\wedge$ *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*))
using *7* **by** *auto*
have *9*: *n*$\le$*nlength* (*pfilt* σ *ls*)
using *8* **by** *auto*
have *10*: *g* (*ntaken n* (*pfilt* σ *ls*))
using *8* **by** *auto*
have *11*: *h* (*ndropn n* (*pfilt* σ *ls*))
using *8* **by** *auto*
have *12*: *nidx* (*ntaken n ls*) $\wedge$ *nnth* (*ntaken n ls*) *0* = *0*
using *4 8* **unfolding** *nidx-expand pfilt-nlength* **by** *simp*
(*metis Suc-leD bot-nat-0.extremum dual-order.trans enat-ord-simps*(*1*) *min.orderE ntaken-nnth*)
have *13*: *nidx* (*ndropn n ls*) $\wedge$ *nnth* (*ndropn n ls*) *0* = (*nnth ls n*)
using *4 9* **unfolding** *pfilt-nlength nidx-expand*
using *nidx-expand nidx-ndropn* **by** *auto*
have *131*: ((*nmap* (λx. *x*$-$ (*nnth ls n*)) (*ndropn n ls*))) = *ndropn n* (*nmap* (λx. *x* $-$ *nnth ls n*) *ls*)
using *ndropn-nmap*[*of n* (λx. *x*$-$ (*nnth ls n*)) *ls*] **by** *presburger*
have *132*: *nnth* (*ndropn n* (*nmap* (λx. *x* $-$ *nnth ls n*) *ls*)) *0* = *0*
by (*metis 13 131 diff-self-eq-0 nnth-nmap zero-enat-def zero-le*)

have *133*: $\bigwedge j.\ enat\ j \leq nlength\ ls \Longrightarrow nnth\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls)\ j = nnth\ ls\ j - nnth\ ls\ n$
using *nnth-nmap* **by** *blast*
have *134*: $nlength\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls)) = nlength\ ls - enat\ n$
by *simp*
have *135*: $\bigwedge j.\ j \leq nlength\ ls - enat\ n \Longrightarrow$
$nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))\ j = nnth\ ls\ (n+ j) - nnth\ ls\ n$
by (*metis 131 ndropn-nlength ndropn-nnth nnth-nmap*)
have *136*: $\bigwedge j.\ (Suc\ j) \leq nlength\ ls - enat\ n \Longrightarrow$
$nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))\ (Suc\ j) = nnth\ ls\ (n+ (Suc\ j)) - nnth\ ls\ n$
using *135* **by** *blast*
have *137*: $\bigwedge j.\ (Suc\ j) \leq nlength\ ls - enat\ n \Longrightarrow$
$nnth\ ls\ (n+ j) - nnth\ ls\ n < nnth\ ls\ (n+ (Suc\ j)) - nnth\ ls\ n$
by (*metis 13 Suc-ile-eq diff-less-mono le-add1 le-add-same-cancel1 ndropn-nlength ndropn-nnth nidx-expand nidx-less-eq order-less-imp-le*)
have *14*: $nidx\ (\ ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))$
by (*metis 134 135 137 Suc-ile-eq dual-order.strict-iff-order nidx-expand*)
have *15*: $g\ (pfilt\ \sigma\ (ntaken\ n\ ls))$
by (*metis 10 9 pfilt-nlength pfilt-ntaken*)
have *16*: $h\ (pfilt\ \sigma\ (ndropn\ n\ ls))$
by (*metis 8 pfilt-ndropn pfilt-nlength*)
have *17*: $g\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls))$
by (*metis 12 15 nfinite-ntaken nle-le ntaken-all ntaken-nlast pfilt-ntaken-nidx*)
have *170*: $nmap\ (\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x))\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ (ndropn\ n\ ls)) =$
$nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls)$
using *nellist.map-comp* **by** *blast*
have *171*: $nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls) =$
$nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)$
proof –
have *172*: $nlength\ (nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls)) =$
$nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))$
by *auto*
have *173*: $\bigwedge j.\ j\leq nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)) \Longrightarrow$
$nnth\ (nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls))\ j =$
$nnth\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))\ j$
by *auto*
(*metis 13 131 134 le-add1 le-add-diff-inverse le-add-same-cancel1 ndropn-nnth nidx-less-eq nlength-nmap*)
show *?thesis*
by (*metis 172 173 nellist-eq-nnth-eq*)
qed
have *171*: $(pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x- (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))) =$
$(pfilt\ \sigma\ (ndropn\ n\ ls))$
using *pfilt-nmap*[*of* (*ndropn* (*nnth ls n*) σ) ((*nmap* (λ*x. x*− (*nnth ls n*)) (*ndropn n ls*)))]
pfilt-nmap[*of* σ (*ndropn n ls*)] *170 171* **by** *force*

have *18*: $h\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x- (nnth\ ls\ n))\ \ (ndropn\ n\ ls))))$
using *16 171* **by** *auto*
have *19*: $powerinterval\ f\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls)$
by (*metis 3 9 nfinite-conv-nlength-enat pfilt-nlength powerinterval-splita*)
have *20*: $powerinterval\ f\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x- (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))$

using *powerinterval-split*[*of ls n σ f*]
by (*metis 3 9 pfilt-nlength*)
have *21*: (*nnth ls n*) ≤ *nlength σ*
by (*metis 3 9 enat-ord-simps*(*2*) *nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast order.order-iff-strict pfilt-nlength the-enat.simps*)
have *22*: *nnth* (*ntaken n ls*) (*the-enat*(*nlength* (*ntaken n ls*))) = (*nnth ls n*)
by (*metis 9 min-def ntaken-nlength ntaken-nnth pfilt-nlength the-enat.simps*)
have *222*: *nlast* (*ntaken n ls*) = (*nnth ls n*)
by (*simp add*: *ntaken-nlast*)
have *23*: *nnth* ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*)))
(*the-enat*(*nlength* ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*))))) =
(*the-enat*(*nlength σ*)) −(*nnth ls n*)
by (*metis 171 222 3 9 enat-le-plus-same*(*2*) *enat-ord-simps*(*3*) *enat-the-enat gen-nlength-def ndropn-nfirst nfinite-ndropn-a nfinite-ntaken nfuse-ntaken-ndropn nlast-nfuse nlength-code nnth-nlast nnth-nmap pfilt-nlength*)
have *223*: *nlast* ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*))) = (*the-enat*(*nlength σ*)) −(*nnth ls n*)
by (*metis 23 3 nfinite-ndropn-a nfinite-nmap nnth-nlast*)
have *224*: (*nmap* (*λx. x*+ (*nnth ls n*)) ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*)))) = *ndropn n ls*
proof −
have *2241*: *nlength* (*nmap* (*λx. x*+ (*nnth ls n*)) ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*)))) =
nlength (*ndropn n ls*)
by *auto*
have *2242*: $\bigwedge$*j. j*≤*nlength* (*ndropn n ls*) ⟶
nnth (*nmap* (*λx. x*+ (*nnth ls n*)) ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*)))) *j* =
nnth (*ndropn n ls*) *j*
by (*metis 13 eq-imp-le le-add-diff-inverse2 nidx-gr-first nlength-nmap nnth-nmap not-gr-zero order-less-imp-le*)
show *?thesis* **using** *2241 2242 nellist-eq-nnth-eq*
by *metis*
qed
have *24*: *ls* = *nfuse* (*ntaken n ls*)
(*nmap* (*λx. x*+ (*nnth ls n*)) ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*))))
using *nfuse-ntaken-ndropn*
by (*metis 224 9 pfilt-nlength*)
have *241*: *nfinite* (*ntaken n ls*)
by *simp*
have *242*: *nfinite* (*ntaken* (*nnth ls n*) *σ*)
by *simp*
have *243*: *nlast* (*ntaken n ls*) = (*the-enat*(*nlength* (*ntaken* (*nnth ls n*) *σ*)))
by (*simp add*: *21 222*)
have *25*: (∃ *ls1. nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧ *nfinite* (*ntaken* (*nnth ls n*) *σ*) ∧
nlast ls1 = (*the-enat*(*nlength* (*ntaken* (*nnth ls n*) *σ*))) ∧
powerinterval f (*ntaken* (*nnth ls n*) *σ*) *ls1* ∧ *g* (*pfilt* (*ntaken* (*nnth ls n*) *σ*) *ls1*))
using *12 17 19 241 242 243* **by** *blast*
have *251*: *nfinite* (*ndropn* (*nnth ls n*) *σ*)
by (*simp add*: *3*)
have *252*: *nlast* (*nmap* (*λx. x* − *nnth ls n*) (*ndropn n ls*)) = *the-enat*(*nlength* (*ndropn* (*nnth ls n*) *σ*))
by (*metis 223 3 idiff-enat-enat ndropn-nlength nfinite-nlength-enat the-enat.simps*)
have *26*: (∃ *ls2. nidx ls2* ∧ *nnth ls2 0* = *0* ∧ *nfinite ls2* ∧ *nfinite* (*ndropn* (*nnth ls n*) *σ*) ∧
nlast ls2 = *the-enat*(*nlength* (*ndropn* (*nnth ls n*) *σ*)) ∧

powerinterval f (*ndropn* (*nnth ls n*) σ) *ls2* $\wedge$ *h* (*pfilt* (*ndropn* (*nnth ls n*) σ) *ls2*))
by (*metis 131 132 14 18 20 252 3 nfinite-ndropn-a nfinite-nmap*)
have *27*: (*ntaken* (*nnth ls n*) σ) $\models$ *f fproj g*
by (*metis 25 fprojection-d-def*)
have *28*: (*ndropn* (*nnth ls n*) σ) $\models$ *f fproj h*
by (*metis 26 fprojection-d-def nfinite-nlength-enat the-enat.simps*)
show *?thesis*
using *21 27 28 chop-defs* **by** *auto*
qed

lemma *OPJ4semchaina*:
assumes (σ $\models$ *f oproj* ((*g* $\wedge$ *finite*);*h*))
shows (σ $\models$ (*f fproj g*) ; (*f oproj h*))
proof –
have *1*: (σ $\models$ *f oproj* ((*g* $\wedge$ *finite*);*h*))
using *assms* **by** *auto*
have *2*: ($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
powerinterval f σ *ls* $\wedge$
($\exists$ *n*$\leq$*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*))))
using *assms* **unfolding** *chop-defs oprojection-d-def finite-defs*
by *auto*
obtain *ls* **where** *3*: *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
powerinterval f σ *ls* $\wedge$
($\exists$ *n*$\leq$*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*)))
using *2* **by** *auto*
have *4*: *nidx ls* $\wedge$ *nnth ls 0* = *0*
using *3* **by** *auto*
have *5*: *powerinterval f* σ *ls*
using *3* **by** *auto*
have *7*: ($\exists$ *n*$\leq$*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*)))
using *3* **by** *auto*
obtain *n* **where** *8*: *n*$\leq$*nlength* (*pfilt* σ *ls*) $\wedge$ *g* (*ntaken n* (*pfilt* σ *ls*)) $\wedge$ *h* (*ndropn n* (*pfilt* σ *ls*))
using *7* **by** *auto*
have *9*: *n*$\leq$*nlength* (*pfilt* σ *ls*)
using *8* **by** *auto*
have *10*: *g* (*ntaken n* (*pfilt* σ *ls*))
using *8* **by** *auto*
have *11*: *h* (*ndropn n* (*pfilt* σ *ls*))
using *8* **by** *auto*
have *12*: *nidx* (*ntaken n ls*) $\wedge$ *nnth* (*ntaken n ls*) *0* = *0*
using *4 8* **unfolding** *nidx-expand pfilt-nlength* **by** *simp*
(*metis 3 Suc-leD bot-nat-0.extremum min.orderE nfinite-ntaken nle-le ntaken-all ntaken-nnth*)
have *13*: *nidx* (*ndropn n ls*) $\wedge$ *nnth* (*ndropn n ls*) *0* = (*nnth ls n*)
using *4 9* **unfolding** *pfilt-nlength nidx-expand*
using *nidx-expand nidx-ndropn* **by** *auto*
have *131*: ((*nmap* (λ*x. x*– (*nnth ls n*)) (*ndropn n ls*))) = *ndropn n* (*nmap* (λ*x. x* – *nnth ls n*) *ls*)
using *ndropn-nmap*[*of n* (λ*x. x*– (*nnth ls n*)) *ls*] **by** *presburger*
have *132*: *nnth* (*ndropn n* (*nmap* (λ*x. x* – *nnth ls n*) *ls*)) *0* = *0*
by (*metis 13 131 diff-self-eq-0 nnth-nmap zero-enat-def zero-le*)
have *133*: $\bigwedge$*j. enat j* $\leq$ *nlength ls* $\Longrightarrow$ *nnth* (*nmap* (λ*x. x* – *nnth ls n*) *ls*) *j* = *nnth ls j* – *nnth ls n*

using *nnth-nmap* **by** *blast*
have *134*: $nlength\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls)) = nlength\ ls - enat\ n$
by *simp*
have *135*: $\bigwedge j.\ j \le nlength\ ls - enat\ n \Longrightarrow$
$nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))\ j = nnth\ ls\ (n+ j) - nnth\ ls\ n$
by (*metis 131 ndropn-nlength ndropn-nnth nnth-nmap*)
have *136*: $\bigwedge j.\ (Suc\ j) \le nlength\ ls - enat\ n \Longrightarrow$
$nnth\ (ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))\ (Suc\ j) = nnth\ ls\ (n+ (Suc\ j)) - nnth\ ls\ n$
using *135* **by** *blast*
have *137*: $\bigwedge j.\ (Suc\ j) \le nlength\ ls - enat\ n \Longrightarrow$
$nnth\ ls\ (n+ j) - nnth\ ls\ n < nnth\ ls\ (n+ (Suc\ j)) - nnth\ ls\ n$
by (*metis 13 Suc-ile-eq diff-less-mono le-add1 le-add-same-cancel1 ndropn-nlength ndropn-nnth nidx-expand nidx-less-eq order-less-imp-le*)
have *14*: $nidx\ (\ ndropn\ n\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ ls))$
by (*metis 134 135 137 Suc-ile-eq dual-order.order-iff-strict nidx-expand*)
have *15*: $g\ (pfilt\ \sigma\ (ntaken\ n\ ls))$
by (*metis 10 9 pfilt-nlength pfilt-ntaken*)
have *16*: $h\ (pfilt\ \sigma\ (ndropn\ n\ ls))$
by (*metis 8 pfilt-ndropn pfilt-nlength*)
have *17*: $g\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls))$
by (*metis 12 15 nfinite-ntaken nle-le ntaken-all ntaken-nlast pfilt-ntaken-nidx*)
have *170*: $nmap\ (\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x))\ (nmap\ (\lambda x.\ x - nnth\ ls\ n)\ (ndropn\ n\ ls)) =$
$nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls)$
using *nellist.map-comp* **by** *blast*
have *171*: $nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls) =$
$nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)$
proof −
have *172*: $nlength\ (nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls)) =$
$nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))$
by *auto*
have *173*: $\bigwedge j.\ j\le nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)) \Longrightarrow$
$nnth\ (nmap\ ((\lambda x.\ nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x.\ x - nnth\ ls\ n))\ (ndropn\ n\ ls))\ j =$
$nnth\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))\ j$
by *auto*
(*metis 13 131 134 le-add1 le-add-diff-inverse le-add-same-cancel1 ndropn-nnth nidx-less-eq nlength-nmap*)
show *?thesis*
by (*metis 172 173 nellist-eq-nnth-eq*)
qed
have *171*: $(pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x- (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))) =$
$(pfilt\ \sigma\ (ndropn\ n\ ls))$
using *pfilt-nmap*[*of* $(ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x- (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))$]
pfilt-nmap[*of* $\sigma\ (ndropn\ n\ ls)$] *170 171* **by** *force*
have *18*: $h\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x- (nnth\ ls\ n))\ \ (ndropn\ n\ ls))))$
using *16 171* **by** *auto*
have *19*: $powerinterval\ f\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls)$
by (*metis 3 8 pfilt-nlength powerinterval-splita-alt*)
have *20*: $powerinterval\ f\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x.\ x- (nnth\ ls\ n))\ \ (ndropn\ n\ ls)))$
using *powerinterval-split*[*of ls n σ f*]
by (*metis 3 8 pfilt-nlength powerinterval-splitb-alt*)

have *21*: (*nnth ls n*) ≤ *nlength σ*
  by (*metis 3 linorder-le-cases nfinite-ntaken ntaken-all*)
have *22*: *nnth* (*ntaken n ls*) (*the-enat*(*nlength* (*ntaken n ls*))) = (*nnth ls n*)
  by (*metis 9 min-def ntaken-nlength ntaken-nnth pfilt-nlength the-enat.simps*)
have *222*: *nlast* (*ntaken n ls*) = (*nnth ls n*)
  by (*simp add*: *ntaken-nlast*)
have *224*: (*nmap* (*λx. x*+ (*nnth ls n*)) ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*)))) = *ndropn n ls*
  proof −
   have *2241*: *nlength* (*nmap* (*λx. x*+ (*nnth ls n*)) ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*)))) =
     nlength (*ndropn n ls*)
    by *auto*
   have *2242*: ⋀*j. j*≤*nlength* (*ndropn n ls*) ⟶
     nnth (*nmap* (*λx. x*+ (*nnth ls n*)) ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*)))) *j* =
     nnth (*ndropn n ls*) *j*
   by (*metis 13 132 9 diff-is-0-eq le-add-diff-inverse2 nidx-gr-first nlength-nmap nnth-nmap nnth-zero-ndropn not-gr-zero order-less-imp-le pfilt-nlength*)
   show *?thesis* **using** *2241 2242 nellist-eq-nnth-eq*
   by *metis*
  qed
have *24*: *ls* = *nfuse* (*ntaken n ls*)
    (*nmap* (*λx. x*+ (*nnth ls n*)) ((*nmap* (*λx. x*− (*nnth ls n*)) (*ndropn n ls*))))
  using *nfuse-ntaken-ndropn*
  by (*metis 224 9 pfilt-nlength*)
have *25*: (∃ *ls1. nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧ *nfinite* (*ntaken* (*nnth ls n*) *σ*) ∧
    nlast ls1 = (*the-enat*(*nlength* (*ntaken* (*nnth ls n*) *σ*))) ∧
    powerinterval f (*ntaken* (*nnth ls n*) *σ*) *ls1* ∧ *g* (*pfilt* (*ntaken* (*nnth ls n*) *σ*) *ls1*))
  using *12 17 19 21 222 3*
  by (*metis min.orderE nfinite-ntaken ntaken-nlength the-enat.simps*)
have *26*: (∃ *ls2. nidx ls2* ∧ *nnth ls2 0* = *0* ∧ ¬*nfinite ls2* ∧ ¬*nfinite* (*ndropn* (*nnth ls n*) *σ*) ∧
    powerinterval f (*ndropn* (*nnth ls n*) *σ*) *ls2* ∧ *h* (*pfilt* (*ndropn* (*nnth ls n*) *σ*) *ls2*))
  using *assms 14 18 20 3 132 131* **by** (*metis nfinite-ndropn nfinite-nmap*)
have *27*: (*ntaken* (*nnth ls n*) *σ*) ⊨ *f fproj g*
  by (*metis 25 fprojection-d-def*)
have *28*: (*ndropn* (*nnth ls n*) *σ*) ⊨ *f oproj h*
  by (*metis 26 oprojection-d-def*)
show *?thesis*
using *21 27 28 chop-defs* **by** *auto*
qed

lemma *PJ4semchainb*:
assumes (*σ* ⊨ (*f fproj g*) ; (*f fproj h*))
shows (*σ* ⊨ *f fproj* (*g*;*h*))
proof −
have *1*: (∃ *n*≤*nlength σ*.
  (∃ *ls. nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite* (*ntaken n σ*) ∧
    nlast ls = (*the-enat*(*nlength* (*ntaken n σ*))) ∧
    powerinterval f (*ntaken n σ*) *ls* ∧ *g* (*pfilt* (*ntaken n σ*) *ls*)) ∧
  (∃ *ls. nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite* (*ndropn n σ*) ∧
    nlast ls = (*the-enat*(*nlength* (*ndropn n σ*))) ∧
    powerinterval f (*ndropn n σ*) *ls* ∧ *h* (*pfilt* (*ndropn n σ*) *ls*)))

using *assms* **unfolding** *chop-defs fprojection-d-def* **by** *simp blast*
obtain *cp* **where** *2*: $cp \le nlength\ \sigma\ \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls)) \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
$nlast\ ls = (the\text{-}enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using *1* **by** *auto*
have *3*: $cp \le nlength\ \sigma$
using *2* **by** *auto*
have *4*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls))$
using *2* **by** *auto*
obtain *ls1* **where** *5*: $nidx\ \ ls1 \wedge nnth\ ls1\ 0 = 0 \wedge nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls1\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1 \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using *4* **by** *auto*
have *6*: $nidx\ \ ls1 \wedge nnth\ ls1\ 0 = 0$
using *5* **by** *auto*
have *61*: $nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma)$
using *5* **by** *auto*
have *7*: $nlast\ ls1\ = the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))$
using *5* **by** *auto*
have *8*: $powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1$
using *5* **by** *auto*
have *9*: $g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using *5* **by** *auto*
have *10*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
$nlast\ ls = (the\text{-}enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using *2* **by** *auto*
obtain *ls2* **where** *11*: $nidx\ \ ls2 \wedge nnth\ ls2\ 0 = 0 \wedge nfinite\ ls2 \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
$nlast\ ls2 = (the\text{-}enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2 \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$
using *10* **by** *auto*
have *12*: $nidx\ \ ls2 \wedge nnth\ ls2\ 0 = 0$
using *11* **by** *auto*
have *13*: $nlast\ ls2\ = the\text{-}enat(nlength\ (ndropn\ cp\ \sigma))$
using *11* **by** *auto*
have *14*: $powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2$
using *11* **by** *auto*
have *15*: $h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$
using *11* **by** *auto*
have *150*: $nlast\ ls1 = cp$
by (*simp add*: *2 7*)
have *151*: $nfirst\ (nmap\ (\lambda x.\ x+\ cp)\ ls2) = cp$
by (*metis 11 NNil-eq-nmap-conv nlast-NNil ntake-eq-NNil-iff ntake-nmap ntaken-0 ntaken-nlast plus-nat.add-0*)

have *152*: *nnth* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) *0* = *0*
by (*metis 150 151 5 nfuse-nnth zero-enat-def zero-le*)
have *153*: *nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧ *nlast ls1* = *cp*
by (*simp add*: *150 5*)
have *154*: *nidx ls2* ∧ *nnth ls2 0* = *0* ∧ *nfinite ls2* ∧ *nfinite* σ ∧ *nlast ls2* = *the-enat* (*nlength* σ) − *cp*
by (*metis 11 idiff-enat-enat ndropn-nlength nfinite-conv-nlength-enat nfinite-ndropn the-enat.simps*)
have *16*: *nidx* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) ∧ *nnth* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) *0* = *0*
using *nidx-nfuse-nidx*[*of ls1 ls2 cp* σ] *153 154 3* **by** *fastforce*
have *17*: *nlast ls1* = *nfirst* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)
by (*simp add*: *150 151*)
have *18*: *nnth* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) (*the-enat*(*nlength ls1*)) = *cp*
by (*metis 151 17 5 ntaken-nfuse ntaken-nlast*)
have *19*: *nlast* (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) = (*the-enat* (*nlength* σ))
by (*metis 154 17 3 5 diff-add enat.distinct*(*2*) *enat-ord-simps*(*1*) *enat-the-enat nfinite-conv-nlength-enat nlast-nfuse nlast-nmap*)
have *20*: *powerinterval f* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*))
using *powerinterval-nfuse*[*of ls1* (*ls2*) *cp* σ *f*] *11 150 154 3 5* **by** *fastforce*
have *21*: σ = *nfuse* (*ntaken cp* σ) (*ndropn cp* σ)
by (*simp add*: *2 nfuse-ntaken-ndropn*)
have *22*: *nnth* ((*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*))) (*the-enat*(*nlength ls1*)) = *cp*
using *18* **by** *blast*
have *23*: (*ntaken* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)))) = (*pfilt* (*ntaken cp* σ) *ls1*)
by (*metis 151 17 2 5 enat.distinct*(*2*) *enat-le-plus-same*(*1*) *enat-the-enat nfinite-conv-nlength-enat nfuse-nlength ntaken-nfuse pfilt-ntaken pfilt-ntaken-nidx*)
have *24*: *g* (*ntaken* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*))))
by (*simp add*: *23 9*)
have *25*: (*ndropn* (*the-enat*(*nlength ls1*)) (*pfilt* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)))) = (*pfilt* (*ndropn cp* σ) *ls2*)
using *pfilt-ndropn*[*of* (*the-enat*(*nlength ls1*)) (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)) σ] *pfilt-ndropn-nidx*[*of cp* σ *ls2*]
by (*metis 12 17 23 3 5 enat-ord-code*(*4*) *enat-the-enat min-def ndropn-nfuse nle-le ntaken-nlength order-less-imp-le pfilt-nlength*)
have *26*: *nlength ls1* ≤ *nlength* (*pfilt* σ (*nfuse ls1* (*nmap* ($\lambda x.$ *x*+ *cp*) *ls2*)))
by (*simp add*: *nfuse-nlength pfilt-nlength*)
have *27*: (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite* σ ∧
nlast ls = *the-enat*(*nlength* σ) ∧
powerinterval f σ *ls* ∧
(∃ *n*≤*nlength* (*pfilt* σ *ls*). *g* (*ntaken n* (*pfilt* σ *ls*)) ∧ *h* (*ndropn n* (*pfilt* σ *ls*))))
using *11 16 19 20 24 25 26*
by (*metis 5 nfinite-ndropn nfinite-nlength-enat nfinite-nmap pfilt-nmap the-enat.simps*)
show *?thesis* **unfolding** *fprojection-d-def* **using** *chop-nfuse nfuse-ntaken-ndropn 27*
by (*metis ndropn-nfirst nfinite-conv-nlength-enat nfinite-ntaken ntaken-nlast*)
qed

lemma *OPJ4semchainb*:
assumes (σ ⊨ (*f fproj g*) ; (*f oproj h*))
shows (σ ⊨ *f oproj* ((*g* ∧ *finite*);*h*))
proof −

have *1*: $(\exists\, n \leq nlength\ \sigma.$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ n\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ n\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ n\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ n\ \sigma)\ ls)) \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ (ndropn\ n\ \sigma) \wedge$
$powerinterval\ f\ (ndropn\ n\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ n\ \sigma)\ ls)))$
using *assms* **unfolding** *chop-defs fprojection-d-def oprojection-d-def*
by *simp blast*
obtain *cp* **where** *2*: $cp \leq nlength\ \sigma\ \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls)) \wedge$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ (ndropn\ cp\ \sigma) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using *1* **by** *auto*
have *3*: $cp \leq nlength\ \sigma$
using *2* **by** *auto*
have *4*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls))$
using *2* **by** *auto*
obtain *ls1* **where** *5*: $nidx\ \ ls1 \wedge nnth\ ls1\ 0 = 0 \wedge nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
$nlast\ ls1\ = (the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
$powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1 \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using *4* **by** *auto*
have *6*: $nidx\ \ ls1 \wedge nnth\ ls1\ 0 = 0$
using *5* **by** *auto*
have *61*: $nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma)$
using *5* **by** *auto*
have *7*: $nlast\ ls1\ = the\text{-}enat(nlength\ (ntaken\ cp\ \sigma))$
using *5* **by** *auto*
have *8*: $powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1$
using *5* **by** *auto*
have *9*: $g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using *5* **by** *auto*
have *10*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ (ndropn\ cp\ \sigma) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using *2* **by** *auto*
obtain *ls2* **where** *11*: $nidx\ \ ls2 \wedge nnth\ ls2\ 0 = 0 \wedge \neg nfinite\ ls2 \wedge \neg nfinite\ (ndropn\ cp\ \sigma) \wedge$
$powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2 \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$
using *10* **by** *auto*
have *12*: $nidx\ \ ls2 \wedge nnth\ ls2\ 0 = 0$
using *11* **by** *auto*
have *14*: $powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2$
using *11* **by** *auto*
have *15*: $h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$
using *11* **by** *auto*
have *150*: $nlast\ ls1 = cp$
by (*simp add*: *2 7*)
have *151*: $nfirst\ (nmap\ (\lambda x.\ x+\ cp)\ ls2) = cp$

by (*metis 11 NNil-eq-nmap-conv nlast-NNil ntake-eq-NNil-iff ntake-nmap ntaken-0 ntaken-nlast plus-nat.add-0*)
have *152*: *nnth* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*)) *0* = *0*
by (*metis 150 151 5 nfuse-nnth zero-enat-def zero-le*)
have *153*: *nidx ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧ *nlast ls1* = *cp*
by (*simp add*: *150 5*)
have *154*: *nidx ls2* ∧ *nnth ls2 0* = *0* ∧ ¬*nfinite ls2* ∧ ¬*nfinite σ*
using *11 nfinite-ndropn-a* **by** *blast*
have *16*: *nidx* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*)) ∧ *nnth* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*)) *0* = *0*
using *nidx-nfuse-nidx-infinite*[*of ls1 ls2 cp σ*] *150 154 5* **by** *fastforce*
have *17*: *nlast ls1* = *nfirst* (*nmap* (*λx. x+ cp*) *ls2*)
by (*simp add*: *150 151*)
have *18*: *nnth* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*)) (*the-enat*(*nlength ls1*)) = *cp*
by (*metis 150 17 5 ntaken-nfuse ntaken-nlast*)
have *20*: *powerinterval f σ* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*))
using *powerinterval-nfuse-alt*[*of ls1* (*ls2*) *cp σ f*]
using *14 154 3 5* **by** *fastforce*
have *21*: *σ* = *nfuse* (*ntaken cp σ*) (*ndropn cp σ*)
by (*simp add*: *2 nfuse-ntaken-ndropn*)
have *22*: *nnth* ((*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*))) (*the-enat*(*nlength ls1*)) = *cp*
using *18* **by** *blast*
have *23*: (*ntaken* (*the-enat*(*nlength ls1*)) (*pfilt σ* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*)))) =
(*pfilt* (*ntaken cp σ*) *ls1*)
by (*metis 150 17 2 5 enat.distinct*(*2*) *enat-le-plus-same*(*1*) *enat-the-enat nfinite-conv-nlength-enat nfuse-nlength ntaken-nfuse pfilt-ntaken pfilt-ntaken-nidx*)
have *24*: *g* (*ntaken* (*the-enat*(*nlength ls1*)) (*pfilt σ* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*))))
by (*simp add*: *23 9*)
have *25*: (*ndropn* (*the-enat*(*nlength ls1*)) (*pfilt σ* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*)))) =
(*pfilt* (*ndropn cp σ*) *ls2*)
using *pfilt-ndropn*[*of* (*the-enat*(*nlength ls1*)) (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*)) *σ*]
pfilt-ndropn-nidx[*of cp σ ls2*]
by (*metis 12 17 2 23 5 enat-ord-code*(*4*) *enat-the-enat min-def ndropn-nfuse nle-le ntaken-nlength order-less-imp-le pfilt-nlength*)
have *26*: *nlength ls1* ≤ *nlength* (*pfilt σ* (*nfuse ls1* (*nmap* (*λx. x+ cp*) *ls2*)))
by (*simp add*: *nfuse-nlength pfilt-nlength*)
have *27*: (∃ *ls. nidx ls* ∧ *nnth ls 0* = *0* ∧ ¬*nfinite ls* ∧ ¬*nfinite σ* ∧
powerinterval f σ ls ∧
(∃ *n*≤*nlength* (*pfilt σ ls*). *g* (*ntaken n* (*pfilt σ ls*)) ∧ *h* (*ndropn n* (*pfilt σ ls*))))
using *11 16 20 24 25 26*
by (*metis 154 5 nfinite-nlength-enat nfinite-nmap nfuse-nfinite the-enat.simps*)
show *?thesis* **unfolding** *fprojection-d-def oprojection-d-def chop-nfuse nfuse-ntaken-ndropn finite-defs* **by** *simp*
(*metis 27 ndropn-nfirst nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast*)
qed

lemma *PJ4sem*:
(*σ* ⊨ *f fproj* (*g*;*h*) = (*f fproj g*) ; (*f fproj h*))
using *PJ4semchaina PJ4semchainb unl-lift2* **by** *blast*

lemma *OPJ4sem*:

($\sigma \models f$ *oproj* (($g \wedge$ *finite*);h) = (f *fproj* g) ; (f *oproj* h))
using *OPJ4semchaina OPJ4semchainb unl-lift2* **by** *blast*

10.3.5 PJ5

lemma *PJ5sem*:
($\sigma \models f$ *fproj* *init*(g) $\longrightarrow$ *init*(g))
by (*simp add*: *fprojection-d-def init-defs*)
(*metis ndropn-0 ndropn-nfirst ntaken-0 pfilt-code*(*1*) *pfilt-ntaken zero-enat-def zero-le*)

lemma *OPJ5sem*:
($\sigma \models f$ *oproj* *init*(g) $\longrightarrow$ *init*(g))
by (*simp add*: *oprojection-d-def init-defs*)
(*metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le*)

10.3.6 PJ6

lemma *PJ6help1*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength* σ)
shows ($\forall$ *i*. *0*$\leq$ *i* $\wedge$ *i*<*nlength ls* $\longrightarrow$ *nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
= (*nnth ls* (*Suc i*)) − (*nnth ls i*))
proof
fix *i*
show *0* $\leq$ *i* $\wedge$ *i* < *nlength ls* $\longrightarrow$
nlength (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) = *nnth ls* (*Suc i*) − *nnth ls i*
using *assms nsubn-nlength*[*of* σ] **unfolding** *nidx-expand*
by (*metis assms*(*1*) *dual-order.order-iff-strict eSuc-enat enat-minus-mono1 enat-ord-simps*(*1*) *idiff-enat-enat ileI1 min.orderE nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast the-enat.simps*)
qed

lemma *OPJ6help1*:
assumes *nidx ls*
nnth ls 0 = *0*
$\neg$*nfinite ls*
$\neg$*nfinite* σ
shows ($\forall$ *i*. *0*$\leq$ *i* $\wedge$ *i*<*nlength ls* $\longrightarrow$ *nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
= (*nnth ls* (*Suc i*)) − (*nnth ls i*))
proof
fix *i*
show *0* $\leq$ *i* $\wedge$ *i* < *nlength ls* $\longrightarrow$
nlength (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) = *nnth ls* (*Suc i*) − *nnth ls i*
using *assms* **unfolding** *nidx-expand nsubn-def1* **by** (*simp add*: *nfinite-conv-nlength-enat*)
qed

lemma *PJ6help2*:

assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength* σ) $\wedge$
($\forall$ *i*<*nlength ls*. *nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*)
shows ($\forall$ *i*$\leq$*nlength ls*. *nnth ls i* = *i*)
proof
fix *i*
show *i*$\leq$*nlength ls* $\longrightarrow$ *nnth ls i* = *i*
proof
(*induct i*)
case *0*
then show *?case* **using** *assms* **by** *blast*
next
case (*Suc i*)
then show *?case*
by (*metis One-nat-def Suc-ile-eq assms*(*1*) *assms*(*5*) *diff-add nidx-expand order-less-imp-le plus-1-eq-Suc*)
qed
qed

lemma *OPJ6help2*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite ls*
¬*nfinite* σ
($\forall$ *i*<*nlength ls*. *nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*)
shows ($\forall$ *i*$\leq$*nlength ls*. *nnth ls i* = *i*)
proof
fix *i*
show *i*$\leq$*nlength ls* $\longrightarrow$ *nnth ls i* = *i*
proof
(*induct i*)
case *0*
then show *?case* **using** *assms* **by** *blast*
next
case (*Suc i*)
then show *?case*
by (*metis One-nat-def Suc-ile-eq assms*(*1*) *assms*(*5*) *diff-add nidx-expand order-less-imp-le plus-1-eq-Suc*)
qed
qed

lemma *PJ6help3*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ

nlast ls = *the-enat*(*nlength σ*)
(∀ *i*≤*nlength ls*. *nnth ls i* = *i*)
shows (∀ *i*<*nlength ls*. *nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*)
proof
fix *i*
show *i*<*nlength ls* ⟶ *nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*
by (*metis One-nat-def add-diff-cancel-right′ assms*(*6*) *eSuc-enat ileI1 order-less-imp-le plus-1-eq-Suc*)
qed

lemma *OPJ6help3*:
assumes *nidx ls*
nnth ls 0 = *0*
¬*nfinite ls*
¬*nfinite σ*
(∀ *i*≤*nlength ls*. *nnth ls i* = *i*)
shows (∀ *i*<*nlength ls*. *nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*)
proof
fix *i*
show *i*<*nlength ls* ⟶ *nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*
by (*metis One-nat-def add-diff-cancel-right′ assms*(*5*) *eSuc-enat ileI1 order-less-imp-le plus-1-eq-Suc*)
qed

lemma *PJ6help4*:
assumes *nfinite σ*
shows (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite σ* ∧
ls = *ntaken* (*the-enat* (*nlength σ*)) (*niterates Suc 0*) ∧
nlast ls = *the-enat*(*nlength σ*) ∧
(∀ *i*≤*nlength ls*. *nnth ls i* = *i*) ∧
(∃ *σ1*. *nlength σ1* = *nlength ls* ∧ (∀ *i*≤*nlength σ1*. *nnth σ1 i* = *nnth σ* (*i*)) ∧ *σ1* = *σ*))

proof −
have *1*: *nidx* (*ntaken* (*the-enat*(*nlength σ*)) (*niterates Suc 0*))
using *nidx-ntaken-niterates-Suc* **by** *presburger*
have *2*: *nnth* (*ntaken* (*the-enat* (*nlength σ*)) (*niterates Suc 0*)) *0* = *0*
by (*simp add*: *ntaken-nnth*)
have *3*: *nlast* (*ntaken* (*the-enat* (*nlength σ*)) (*niterates Suc 0*)) = *the-enat*(*nlength σ*)
by (*simp add*: *ntaken-nlast*)
have *4*: (∀ *i*≤*nlength* (*ntaken* (*the-enat* (*nlength σ*)) (*niterates Suc 0*)).
nnth (*ntaken* (*the-enat* (*nlength σ*)) (*niterates Suc 0*)) *i* = *i*)
by (*simp add*: *ntaken-nnth*)
have *5*: (∃ *σ1*. *nlength σ1* = *nlength* (*ntaken* (*the-enat* (*nlength σ*)) (*niterates Suc 0*)) ∧
(∀ *i*≤*nlength σ1*. *nnth σ1 i* = *nnth σ i*) ∧ *σ1* = *σ*)
using *assms* **by** (*simp add*: *enat-the-enat nfinite-nlength-enat*)
show *?thesis*
using *1 2 3 4 5 assms nfinite-ntaken* **by** *blast*
qed

lemma *OPJ6help4*:
assumes $\neg$*nfinite* σ
shows $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$ls = (niterates\ Suc\ 0)\ \wedge$
$(\forall\, i \le nlength\ ls.\ nnth\ ls\ i = i) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge \sigma 1 = \sigma))$

proof –
have *1*: $nidx\ ((niterates\ Suc\ 0))$
using *nidx-expand nnth-niterates-Suc* **by** *presburger*
have *2*: $nnth\ (\ (niterates\ Suc\ 0))\ 0 = 0$
by (*simp*)
have *4*: $(\forall\, i \le nlength\ (\ (niterates\ Suc\ 0)).\ nnth\ (\ (niterates\ Suc\ 0))\ i = i)$
by (*simp*)
have *5*: $(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ (\ (niterates\ Suc\ 0)) \wedge$
$(\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ i) \wedge \sigma 1 = \sigma)$
using *assms* **by** (*simp add*: *nfinite-conv-nlength-enat*)
show *?thesis*
using *1 2 4 5 assms* **by** *blast*
qed

lemma *PJ6help5*:
assumes *nfinite* σ
shows $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ \ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (\ nnth\ ls\ i)) \wedge g\ \sigma 1))$
$= g\ \sigma$
proof –
have $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ \ ls\ = the\text{-}enat\ (nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$
$=$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ ls.\ nnth\ ls\ (Suc\ i) - nnth\ ls\ i = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (\ nnth\ ls\ i)) \wedge g\ \sigma 1))$
using *PJ6help1*[*of* - σ] *assms* **by** *auto*
also have ... =
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ \ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i \le nlength\ ls.\ nnth\ ls\ i = i) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

using *PJ6help2 PJ6help3* **by** *blast*
also have ... =

($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ *nfinite ls* $\wedge$ *nfinite* σ $\wedge$
nlast ls = *the-enat*(*nlength* σ) $\wedge$
($\forall$ *i*$\leq$*nlength ls. nnth ls i* = *i*) $\wedge$
($\exists$ σ*1. nlength* σ*1* = *nlength ls* $\wedge$ ($\forall$ *i*$\leq$*nlength* σ*1. nnth* σ*1 i* = *nnth* σ (*i*)) $\wedge$ *g* σ*1*))

using *assms* **by** *auto* (*metis*, *metis*)
also have ... =
($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ *nfinite ls* $\wedge$ *nfinite* σ $\wedge$
nlast ls = *the-enat*(*nlength* σ) $\wedge$
($\forall$ *i*$\leq$*nlength ls. nnth ls i* = *i*) $\wedge$
($\exists$ σ*1. nlength* σ*1* = *nlength ls* $\wedge$ ($\forall$ *i*$\leq$*nlength* σ*1. nnth* σ*1 i* = *nnth* σ (*i*)) $\wedge$ σ*1* = σ $\wedge$ *g* σ))
by *auto*
(*metis dual-order.refl enat.distinct*(*2*) *enat-the-enat nellist-eq-nnth-eq nfinite-nlength-enat nnth-nlast*)
also have ... =
g σ

using *PJ6help4 assms* **by** *auto*
finally show ($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ *nfinite ls* $\wedge$ *nfinite* σ $\wedge$
nlast ls = *the-enat*(*nlength* σ) $\wedge$
($\forall$ *i*<*nlength ls. nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) = *Suc 0*) $\wedge$
($\exists$ σ*1. nlength* σ*1* = *nlength ls* $\wedge$ ($\forall$ *i*$\leq$*nlength* σ*1. nnth* σ*1 i* = *nnth* σ (*nnth ls i*)) $\wedge$ *g* σ*1*))
= *g* σ
.
qed

lemma *OPJ6help5*:
assumes $\neg$*nfinite* σ
shows ($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
($\forall$ *i*<*nlength ls. nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) = *Suc 0*) $\wedge$
($\exists$ σ*1. nlength* σ*1* = *nlength ls* $\wedge$ ($\forall$ *i*$\leq$*nlength* σ*1. nnth* σ*1 i* = *nnth* σ (*nnth ls i*)) $\wedge$ *g* σ*1*))
= *g* σ
proof –
have ($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
($\forall$ *i*<*nlength ls. nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) = *Suc 0*) $\wedge$
($\exists$ σ*1. nlength* σ*1* = *nlength ls* $\wedge$ ($\forall$ *i*$\leq$*nlength* σ*1. nnth* σ*1 i* = *nnth* σ (*nnth ls i*)) $\wedge$ *g* σ*1*))
=
($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
($\forall$ *i*<*nlength ls. nnth ls* (*Suc i*) − *nnth ls i* = *Suc 0*) $\wedge$
($\exists$ σ*1. nlength* σ*1* = *nlength ls* $\wedge$ ($\forall$ *i*$\leq$*nlength* σ*1. nnth* σ*1 i* = *nnth* σ (*nnth ls i*)) $\wedge$ *g* σ*1*))
using *OPJ6help1*[*of* - σ] *assms* **by** *auto*
also have ... =
($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
($\forall$ *i*$\leq$*nlength ls. nnth ls i* = *i*) $\wedge$
($\exists$ σ*1. nlength* σ*1* = *nlength ls* $\wedge$ ($\forall$ *i*$\leq$*nlength* σ*1. nnth* σ*1 i* = *nnth* σ (*nnth ls i*)) $\wedge$ *g* σ*1*))

using *OPJ6help2*[*of* - σ] *OPJ6help3*[*of* - σ] **by** *blast*
also have ... =
($\exists$ *ls. nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
($\forall$ *i*$\leq$*nlength ls. nnth ls i* = *i*) $\wedge$

$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge g\ \sigma 1))$

using *assms* **by** *auto* (*metis*, *metis*)
also have ... =
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i \le nlength\ ls.\ nnth\ ls\ i = i) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge \sigma 1 = \sigma \wedge g\ \sigma))$

by *auto* (*metis OPJ6help4 nellist-eq-nnth-eq*)
also have ... =
$g\ \sigma$

using *OPJ6help4 assms* **by** *auto*
finally show $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall\, i < nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (\ nnth\ ls\ i)) \wedge g\ \sigma 1))$
$= g\ \sigma$
.
qed

lemma *PJ6sem*:
$(\sigma \models skip\ \ fproj\ \ g = (g \wedge finite))$
proof −
have *1*: $(\sigma \models skip\ \ fproj\ \ g = (g \wedge finite)) =$
$((\exists\, l.\ nidx\ \ l \wedge nnth\ l\ 0 = 0 \wedge nfinite\ l \wedge nfinite\ \sigma \wedge nlast\ l\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ l.\ \ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))\ ) \models skip) \wedge g\ (pfilt\ \sigma\ l)) =$
$(g\ \sigma \wedge nfinite\ \sigma)\ )$
by (*simp add*: *fprojection-d-def powerinterval-def finite-defs*)
have *2*: $(\exists\, ls.\ nidx\ \ ls \wedge\ \ nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge g\ (pfilt\ \sigma\ \ ls)) =$
$(\exists\, ls\ \sigma 1\ .\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge nlast\ \ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge$
$nlength\ \sigma 1 = nlength\ ls \wedge$
$(\forall\ \ i \le\ nlength\ \sigma 1.\ (nnth\ \sigma 1\ i) = (nnth\ \sigma\ (nnth\ ls\ i))) \wedge$
$g\ \sigma 1)$
using *pfilt-expand*[*of - σ*]
by *auto* (*blast*,*metis*)
have *3*: $(\exists\, ls\ \sigma 1\ .\ nidx\ \ ls \wedge\ \ \ nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
$nlast\ \ ls = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ ls.\ \ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge$
$nlength\ \sigma 1 = nlength\ ls \wedge$
$(\forall\ \ i \le\ nlength\ \sigma 1.\ (nnth\ \sigma 1\ i) = (nnth\ \sigma\ (nnth\ ls\ i))) \wedge$
$g\ \sigma 1) =$
$(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge nlast\ ls\ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$
by (*simp add*: *skip-defs*)
have *4*: $(\exists\, ls.\ nidx\ \ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge\ \ nlast\ ls\ \ = the\text{-}enat(nlength\ \sigma) \wedge$
$(\forall\, i < nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists\, \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall\, i \le nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1)) =$

$((g\ \sigma) \wedge nfinite\ \sigma)$
using *PJ6help5*[*of* σ *g*] **by** *auto*
from *1 2 3 4* **show** *?thesis*
by *simp*
qed

lemma *OPJ6sem*:
$(\sigma \models skip\ oproj\ g = (g \wedge inf))$
proof –
have *1*: $(\sigma \models skip\ oproj\ g = (g \wedge inf)) =$
$((\exists ls.\ nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall i<nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge g\ (pfilt\ \sigma\ ls)) =$
$(g\ \sigma \wedge \neg nfinite\ \sigma)\ )$
by (*simp add*: *oprojection-d-def powerinterval-def infinite-defs*)
have *2*: $(\exists ls.\ nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall i<nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge g\ (pfilt\ \sigma\ ls)) =$
$(\exists ls\ \sigma 1\ .\ nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall i<nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge$
$nlength\ \sigma 1 = nlength\ ls \wedge$
$(\forall\ i\leq nlength\ \sigma 1.\ (nnth\ \sigma 1\ i) = (nnth\ \sigma\ (nnth\ ls\ i))) \wedge$
$g\ \sigma 1)$
using *pfilt-expand*[*of* - σ] **by** *auto* (*blast*,*metis*)
have *3*: $(\exists ls\ \sigma 1\ .\ nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall i<nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models skip) \wedge$
$nlength\ \sigma 1 = nlength\ ls \wedge$
$(\forall\ i\leq nlength\ \sigma 1.\ (nnth\ \sigma 1\ i) = (nnth\ \sigma\ (nnth\ ls\ i))) \wedge$
$g\ \sigma 1) =$
$(\exists ls.\ nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall i<nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall i\leq nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$
by (*simp add*: *skip-defs*)
have *4*: $(\exists ls.\ nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
$(\forall i<nlength\ ls.\ nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = Suc\ 0) \wedge$
$(\exists \sigma 1.\ nlength\ \sigma 1 = nlength\ ls \wedge (\forall i\leq nlength\ \sigma 1.\ nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1)) =$
$((g\ \sigma) \wedge \neg nfinite\ \sigma)$
using *OPJ6help5*[*of* σ *g*] **by** *auto*
from *1 2 3 4* **show** *?thesis*
by *simp*
qed

10.3.7 PJ7

lemma *PJemptyImp*:
assumes $nlength\ \sigma = 0$
shows $(\sigma \models (f\ \ fproj\ \ g) = g)$
using *assms*
by (*auto simp add*: *fprojection-d-def powerinterval-def nsubn-def1*)
(*metis i0-less less-numeral-extra*(*3*) *ndropn-0 ndropn-nlast nfinite-conv-nlength-enat*
nidx-less-last-1 nnth-nlast pfilt-code(*1*) *the-enat.simps the-enat-0*,
metis PJ6help4 ndropn-0 ndropn-nlast nfinite-ntaken not-less-zero ntaken-all ntaken-nlast

pfilt-code(*1*) *zero-le*)

lemma *PJ7empty*:
 assumes *nlength* σ = *0*
 shows ($\sigma \models f$ *fproj* (*g* *fproj* *h*) = (*f* *fproj* *g*) *fproj* *h*)
proof −
 have *1*: ($\sigma \models f$ *fproj* (*g* *fproj* *h*) = (*g* *fproj* *h*))
  using *PJemptyImp assms* **by** *blast*
 have *2*: ($\sigma \models$ (*g* *fproj* *h*) = *h*)
  using *PJemptyImp assms* **by** *blast*
 have *3*: ($\sigma \models$ (*f* *fproj* *g*) *fproj* *h* = *h*)
  using *PJemptyImp assms* **by** *blast*
 from *1 2 3* **show** *?thesis* **by** *simp*
qed

lemma *PJ7helpchain1a-help-1*:
 assumes *nidx* *ls*
  nnth ls 0 = *0*
  nfinite ls
  nfinite σ
  nlast ls = *the-enat*(*nlength* σ)
  ($\forall$ *i*<*nlength ls*. (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models f$ *fproj* *g*)
  nlength σ > *0*
 shows *nidx* (*nfusecat* ((*lcppl f g* σ *ls*))) $\wedge$ *nnth* (*nfusecat* ((*lcppl f g* σ *ls*))) *0* = *0*
proof −
 have *0*: *nlength* σ > *0* $\longrightarrow$ *nlength ls* >*0*
  using *assms*
  by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
 have *01*: *nfirst ls* = *0*
  using *assms*
  by (*metis ndropn-0 ndropn-nfirst*)
 have *02*: *nlastnfirst* (*lcppl f g* σ *ls*)
  using *assms lcppl-nfusecat-nlastnfirst*[*of ls* σ *f g*]
  by (*metis 0*)
 have *1*: *nfirst* (*nfusecat* (*lcppl f g* σ *ls*)) = *0*
  using *assms 0 01 02 lcppl-nfirst*[*of ls* σ *f g*] **by** (*metis nfirst-nfusecat-nfirst*)
 from *1 0* **show** *?thesis* **using** *assms lcppl-nfusecat-nidx*[*of ls* σ *f g*]
 by (*metis 01 ntaken-0 ntaken-nlast*)
qed

lemma *PJ7helpchain1a-help-1-alt*:
 assumes *nidx* *ls*
  nnth ls 0 = *0*
  $\neg$*nfinite ls*
  $\neg$*nfinite* σ
  ($\forall$ *i*<*nlength ls*. (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) $\models f$ *fproj* *g*)
  nlength σ > *0*
 shows *nidx* (*nfusecat* ((*lcppl f g* σ *ls*))) $\wedge$ *nnth* (*nfusecat* ((*lcppl f g* σ *ls*))) *0* = *0*
proof −
 have *0*: *nlength* σ > *0* $\longrightarrow$ *nlength ls* >*0*

using *assms*
by (*metis enat-the-enat nfinite-conv-nlength-enat*)
have *01*: *nfirst ls* = *0*
using *assms*
by (*metis ndropn-0 ndropn-nfirst*)
have *02*: *nlastnfirst* (*lcppl f g σ ls*)
using *assms lcppl-nfusecat-nlastnfirst-alt*[*of ls σ f g*]
by (*metis*)
have *1*: *nfirst* (*nfusecat* (*lcppl f g σ ls*)) = *0*
using *assms 0 01 02*
lcppl-nfirst-alt[*of ls σ f g*]
by (*metis nfirst-nfusecat-nfirst*)
from *1 0* **show** *?thesis* **using** *assms lcppl-nfusecat-nidx-alt*[*of ls σ f g*]
by (*metis 01 ntaken-0 ntaken-nlast*)
qed

lemma *PJ7helpchain1a-help-2*:
assumes *nidx ls*
nnth ls 0 = *0*
nfinite ls
nfinite σ
nlast ls = *the-enat*(*nlength σ*)
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
nlength σ > *0*
shows *powerinterval f σ* (*nfusecat* ((*lcppl f g σ ls*)))
proof −
have *1*: (∀ *i*<*nlength ls*.
powerinterval f (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))
using *assms cppl-fprojection* **by** *blast*
have *2*: ∀ *i*<*nlength ls*.
∀ *ia*<*nlength* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))).
f (*nsubn* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*))
)
using *1* **by** (*simp add*: *powerinterval-def*)
have *3*: ∀ *i*<*nlength ls*.
∀ *ia*<*nlength* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))).
(*nsubn* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*))
) =
(*nsubn σ*
(*nnth* (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) *ia*)
(*nnth* (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
)
proof *auto*
fix *i*
fix *ia*

assume *a0*: *enat i* < *nlength ls*
assume *a1*: *enat ia* < *nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
show *nsubn* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) =
nsubn σ (*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia* + *nnth ls i*)
(*nnth* (*nmap* (λ*x*. *x* + *nnth ls i*) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
proof −
have *300*: *0* < *nlength* σ
using *assms*(*7*) **by** *auto*
have *301*: *nnth ls i* < *nnth ls* (*Suc i*)
by (*metis a0 assms*(*1*) *eSuc-enat ileI1 nidx-expand*)
have *302*: (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ (*f fproj g*)
by (*simp add*: *a0 assms*(*6*))
have *303*: *the-enat* (*nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) = (*nnth ls* (*Suc i*)) − (*nnth ls i*)
by (*simp add*: *PJ6help1 a0 assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*))
have *304*: *nlast* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) = (*nnth ls* (*Suc i*)) − (*nnth ls i*)
using *a0 a1 cppl-fprojection*[*of f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))]
using *302 303* **by** *presburger*
have *305*: (*Suc ia*) ≤ *nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
by (*simp add*: *Suc-ile-eq a1*)
have *306*: *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia* ≤
nnth (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)
by (*metis 302 a1 cppl-fprojection eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *307*: *nlast* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))=
nnth (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
(*the-enat*(*nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *302 cppl-fprojection nnth-nlast* **by** *auto*
have *308*: *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*) ≤
nlast (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
by (*metis 302 305 cppl-fprojection nidx-all-le-nlast*)
have *310*: *enat* (*nnth ls* (*Suc i*)) ≤ *nlength* σ
using *a0 assms*
by (*metis dual-order.refl eSuc-enat enat-ord-simps*(*1*) *enat-ord-simps*(*3*) *enat-the-enat*
ileI1 nfinite-conv-nlength-enat nidx-less-eq nnth-nlast the-enat.simps)
have *30*: *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*) ≤ *nnth ls* (*Suc i*) − *nnth ls i*
using *assms cppl-bounds*[*of* (*nnth ls i*) (*nnth ls* (*Suc i*)) σ *f g Suc ia*]
PJ6help1[*of ls* σ] *a0 a1 cppl-fprojection*[*of f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))]
by (*metis 303 308*)
have *311*: *nsubn* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) =
nsubn σ (*nnth ls i* + *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth ls i* + *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*))
using *nsubn-nsubn-1*[*of* (*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) (*nnth ls i*) (*nnth ls* (*Suc i*)) σ]
using *30 301 306 310 order-less-imp-le* **by** *blast*
have *312*: (*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia* + *nnth ls i*) =
(*nnth ls i* + *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
by *simp*

have *313*: (*nnth ls i* + *nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) =
(*nnth* (*nmap* (λ*x*. *x* + *nnth ls i*) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
using *305* **by** *force*
show *?thesis*
using *311 312 313* **by** *presburger*
qed
qed
have *4*: ∀ *i*<*nlength ls*.
∀ *ia*<*nlength* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))).
f (*nsubn σ* (*nnth* (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) *ia*)
(*nnth* (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
)
using *2 3* **by** *auto*
have *5*: *nlength*(*lcppl f g σ ls*) = *nlength ls* −*1*
using *assms lcppl-nlength lcppl-nlength-zero*
by (*metis epred-0 epred-conv-minus i0-less*)
have *6*: *nlength σ* >*0* ⟶ *nlength ls* >*0*
using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *7*: ∀ *i*< *nlength ls*.
∀ *ia*<*nlength* ((*nnth* (*lcppl f g σ ls*) *i*)).
f (*nsubn σ* (*nnth* (*nnth* (*lcppl f g σ ls*) *i*) *ia*)
(*nnth* (*nnth* (*lcppl f g σ ls*) *i*) (*Suc ia*))
)
using *assms 4 lcppl-nnth* **by** (*metis nlength-nmap*)
have *8*: *nlength σ* >*0* ⟶
(∀ *i*≤ *nlength*(*lcppl f g σ ls*).
∀ *ia*<*nlength* ((*nnth* (*lcppl f g σ ls*) *i*)).
f (*nsubn σ* (*nnth* (*nnth* (*lcppl f g σ ls*) *i*) *ia*)
(*nnth* (*nnth* (*lcppl f g σ ls*) *i*) (*Suc ia*))
))
using *5 6 7 assms* **by** (*metis co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength*)
have *9*: *nlength σ* >*0* ⟶
powerinterval f σ (*nfusecat* ((*lcppl f g σ ls*))) =
(∀ *i*<*nlength* (*nfusecat* (*lcppl f g σ ls*)).
f (*nsubn σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*)) *i*) (*nnth* (*nfusecat* (*lcppl f g σ ls*)) (*Suc i*))))
by (*simp add*: *powerinterval-def*)
have *10*: *nlength σ* >*0* ⟶ *nlastnfirst* (*lcppl f g σ ls*)
using *6 assms lcppl-nfusecat-nlastnfirst*
by (*metis nfinite-conv-nlength-enat*)
have *11*: *nlength σ* >*0* ⟶
(∀ *j* ≤ *nlength* (*lcppl f g σ ls*). *nlength*(*nnth* (*lcppl f g σ ls*) *j*) > *0*)

using *assms*
by (*metis enat.distinct*(*2*) *enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)
have *110*: *all-gr-zero* (*lcppl f g σ ls*)
using *11 all-gr-zero-nnth-a assms*(*7*) **by** *blast*
have *111*: *all-nfinite* (*lcppl f g σ ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite* **by** *blast*
have *12*: *nlength σ* >*0* ⟶

(∀ *i*≤ *nlength*(*lcppl f g σ ls*).
  (∀ *ia*<*nlength* ((*nnth* (*lcppl f g σ ls*) *i*)).
    f (*nsubn σ* (*nnth* (*nnth* (*lcppl f g σ ls*) *i*) *ia*)
          (*nnth* (*nnth* (*lcppl f g σ ls*) *i*) (*Suc ia*))
          ))) =
      (∀ *j*<*nlength* (*nfusecat* (*lcppl f g σ ls*)).
    f (*nsubn σ* (*nnth* (*nfusecat* (*lcppl f g σ ls*)) *j*)
          (*nnth* (*nfusecat* (*lcppl f g σ ls*)) (*Suc j*))
          ))

using *nfusecat-split-nsubn*[*of* (*lcppl f g σ ls*) *f σ*] *6 10 11 110 111 assms*
unfolding *nridx-expand*
by (*simp add*: *Suc-ile-eq*)
show *?thesis* **using** *12 8 9* **using** *assms*
by *meson*
qed

lemma *PJ7helpchain1a-help-2-alt*:
assumes *nidx ls*
  nnth ls 0 = *0*
  ¬*nfinite ls*
  ¬*nfinite σ*
  (∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
shows *powerinterval f σ* (*nfusecat* ((*lcppl f g σ ls*)))
proof −
have *1*: (∀ *i*<*nlength ls*.
    powerinterval f (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
        (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))))
using *assms cppl-fprojection* **by** *blast*
have *2*: ∀ *i*<*nlength ls*.
  ∀ *ia*<*nlength* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))).
  f (*nsubn* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
      (*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
      (*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*))
)
using *1* **by** (*simp add*: *powerinterval-def*)
have *3*: ∀ *i*<*nlength ls*.
  ∀ *ia*<*nlength* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))).
    (*nsubn* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))
      (*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
      (*nnth* (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*))
) =
  (*nsubn σ*
    (*nnth* (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) *ia*)
    (*nnth* (*nmap* (λ*x*. *x*+ (*nnth ls i*)) (*cppl f g* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
)
proof *auto*
fix *i*
fix *ia*
assume *a0*: *enat i* < *nlength ls*

assume *a1*: *enat ia* < *nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
show *nsubn* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) =
nsubn σ (*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia* + *nnth ls i*)
(*nnth* (*nmap* (λ*x*. *x* + *nnth ls i*) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
proof −
have *300*: *0* < *nlength* σ
by (*metis assms*(*4*) *gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def*)
have *301*: *nnth ls i* < *nnth ls* (*Suc i*)
by (*metis a0 assms*(*1*) *eSuc-enat ileI1 nidx-expand*)
have *302*: (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ (*f fproj g*)
by (*simp add*: *a0 assms*(*5*))
have *303*: *the-enat* (*nlength* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) = (*nnth ls* (*Suc i*)) − (*nnth ls i*)
by (*simp add*: *OPJ6help1 a0 assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*))
have *304*: *nlast* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) = (*nnth ls* (*Suc i*)) − (*nnth ls i*)
using *a0 a1 cppl-fprojection*[*of f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))]
using *302 303* **by** *presburger*
have *305*: (*Suc ia*) ≤ *nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
by (*simp add*: *Suc-ile-eq a1*)
have *306*: *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia* ≤
nnth (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)
by (*metis 302 a1 cppl-fprojection eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *307*: *nlast* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))=
nnth (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
(*the-enat*(*nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))))
using *302 cppl-fprojection nnth-nlast* **by** *auto*
have *308*: *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*) ≤
nlast (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))
by (*metis 302 305 cppl-fprojection nidx-all-le-nlast*)
have *310*: *enat* (*nnth ls* (*Suc i*)) ≤ *nlength* σ
using *a0 assms*
by (*metis enat-ord-simps*(*3*) *enat-the-enat nfinite-conv-nlength-enat*)
have *30*: *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*) ≤ *nnth ls* (*Suc i*) − *nnth ls i*
using *assms cppl-bounds*[*of* (*nnth ls i*) (*nnth ls* (*Suc i*)) σ *f g Suc ia*]
PJ6help1[*of ls* σ] *a0 a1 cppl-fprojection*[*of f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))]
by (*metis 303 308*)
have *311*: *nsubn* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) =
nsubn σ (*nnth ls i* + *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth ls i* + *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*))
using *nsubn-nsubn-1*[*of* (*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
(*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) (*nnth ls i*) (*nnth ls* (*Suc i*)) σ]
using *30 301 306 310 order-less-imp-le* **by** *blast*
have *312*: (*nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia* + *nnth ls i*) =
(*nnth ls i* + *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) *ia*)
by *simp*
have *313*: (*nnth ls i* + *nnth* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))) (*Suc ia*)) =
(*nnth* (*nmap* (λ*x*. *x* + *nnth ls i*) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))

using *305* **by** *force*
show *?thesis*
using *311 312 313* **by** *presburger*
qed
qed
have *4*: $\forall$ *i<nlength ls.*
$\forall$ *ia<nlength* (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*)))).
f (*nsubn* σ (*nnth* (*nmap* (λx. *x+* (*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))) *ia*)
(*nnth* (*nmap* (λx. *x+* (*nnth ls i*)) (*cppl f g* (*nsubn* σ (*nnth ls i*) (*nnth ls* (*Suc i*))))) (*Suc ia*))
)
using *2 3* **by** *auto*
have *5*: *nlength*(*lcppl f g* σ *ls*) = *nlength ls* −*1*
using *assms lcppl-nlength lcppl-nlength-zero*
by (*simp add*: *epred-conv-minus lcppl-nlength-alt*)
have *6*: *nlength* σ >*0* $\longrightarrow$ *nlength ls* >*0*
using *assms*
by (*simp add*: *nfinite-conv-nlength-enat*)
have *7*: $\forall$ *i< nlength ls.*
$\forall$ *ia<nlength* ((*nnth* (*lcppl f g* σ *ls*) *i*)).
f (*nsubn* σ (*nnth* (*nnth* (*lcppl f g* σ *ls*) *i*) *ia*)
(*nnth* (*nnth* (*lcppl f g* σ *ls*) *i*) (*Suc ia*))
)
using *assms 4 lcppl-nnth*
by (*metis nlength-nmap*)
have *8*: ($\forall$ *i*≤ *nlength*(*lcppl f g* σ *ls*).
$\forall$ *ia<nlength* ((*nnth* (*lcppl f g* σ *ls*) *i*)).
f (*nsubn* σ (*nnth* (*nnth* (*lcppl f g* σ *ls*) *i*) *ia*)
(*nnth* (*nnth* (*lcppl f g* σ *ls*) *i*) (*Suc ia*))
))
using *5 6 7 assms enat-ile nfinite-conv-nlength-enat not-le-imp-less* **by** *blast*
have *9*: *powerinterval f* σ (*nfusecat* ((*lcppl f g* σ *ls*))) =
($\forall$ *i<nlength* (*nfusecat* (*lcppl f g* σ *ls*)).
f (*nsubn* σ (*nnth* (*nfusecat* (*lcppl f g* σ *ls*)) *i*) (*nnth* (*nfusecat* (*lcppl f g* σ *ls*)) (*Suc i*))))
by (*simp add*: *powerinterval-def*)
have *10*: *nlastnfirst* (*lcppl f g* σ *ls*)
using *6 assms lcppl-nfusecat-nlastnfirst-alt*
by (*metis*)
have *11*: ($\forall$ *j* ≤ *nlength* (*lcppl f g* σ *ls*). *nlength*(*nnth* (*lcppl f g* σ *ls*) *j*) > *0*)
using *assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *110*: *all-gr-zero* (*lcppl f g* σ *ls*)
using *11 all-gr-zero-nnth-a assms* **by** *blast*
have *111*: *all-nfinite* (*lcppl f g* σ *ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt* **by** *blast*
have *12*: ($\forall$ *i*≤ *nlength*(*lcppl f g* σ *ls*).
($\forall$ *ia<nlength* ((*nnth* (*lcppl f g* σ *ls*) *i*)).
f (*nsubn* σ (*nnth* (*nnth* (*lcppl f g* σ *ls*) *i*) *ia*)
(*nnth* (*nnth* (*lcppl f g* σ *ls*) *i*) (*Suc ia*))
))) =
($\forall$ *j<nlength* (*nfusecat* (*lcppl f g* σ *ls*)).
f (*nsubn* σ (*nnth* (*nfusecat* (*lcppl f g* σ *ls*)) *j*)

(*nnth* (*nfusecat* (*lcppl f g σ ls*)) (*Suc j*))
))

using *nfusecat-split-nsubn*[*of* (*lcppl f g σ ls*) *f σ*] *10 11 110 111 assms*
unfolding *nridx-expand*
by (*simp add*: *Suc-ile-eq*)
show *?thesis* **using** *12 8 9* **using** *assms*
by *meson*
qed

lemma *PJ7helpchain1a-help-3*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(*nlength σ*)
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
h (*pfilt σ ls*)
nlength σ >*0*
shows *nlast* (*nfusecat* (*lcppl f g σ ls*)) = *nlength σ*
proof −
have *0*: *nlength σ* > *0* ⟶ *nlength ls* >*0*
using *assms*
by (*metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *00*: *nfinite* (*lcppl f g σ ls*)
using *assms*(*1*) *assms*(*3*) *lcppl-nfinite* **by** *blast*
have *01*: *nlastnfirst* (*lcppl f g σ ls*)
using *0 assms lcppl-nfusecat-nlastnfirst* **by** *blast*
have *02*: *all-nfinite* (*lcppl f g σ ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite* **by** *blast*
have *1*: *nlength σ* >*0* ⟶
nlast (*nfusecat* (*lcppl f g σ ls*)) = *nlast*(*nlast* ((*lcppl f g σ ls*)))
using *assms 0 nlastfirst-nfusecat-nlast*[*of* (*lcppl f g σ ls*)] *00 01 02* **by** *blast*
have *2*: *nlength σ* >*0* ⟶
nlast ((*lcppl f g σ ls*))
= (*nnth* (*lcppl f g σ ls*) (*the-enat*(*nlength ls*) −*1*))
using *assms*
by (*metis 0 epred-enat lcppl-nfinite lcppl-nlength nfinite-nlength-enat nnth-nlast the-enat.simps*)
have *3*: *nlength σ* >*0* ⟶
(*nnth* (*lcppl f g σ ls*) (*the-enat*(*nlength ls*) −*1*)) =
(*nmap* (λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
(*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))

using *assms 0 lcppl-nnth*[*of ls* (*the-enat*(*nlength ls*) −*1*) *f g σ*]
by (*metis Suc-n-not-le-n add.commute diff-add enat.distinct*(*2*) *enat-ord-simps*(*1*) *enat-ord-simps*(*4*) *enat-the-enat ileI1 not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc*)
have *4*: *nlength σ* >*0* ⟶
nlast(((*cppl f g* (*nsubn σ* (*nnth ls* (*the-enat*(*nlength ls*) −*1*))

(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))) =
the-enat(*nlength* ((*nsubn* σ (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))
using *0 assms cppl-fprojection*[*of f g*
(*nsubn* σ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)) (*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))]
by (*metis Suc-n-not-le-n add.commute diff-add enat.distinct*(*2*) *enat-ord-simps*(*1*)
enat-the-enat ileI1 nfinite-conv-nlength-enat not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc)

have *5*: *nlength* σ >*0* ⟶
the-enat(*nlength* ((*nsubn* σ (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*)))))) =
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) − (*nnth ls* (*the-enat*(*nlength ls*) −*1*))

using *0 PJ6help1 assms*
by (*metis Suc-n-not-le-n add.commute diff-add enat-ord-simps*(*1*) *ileI1 le-add-same-cancel1*
nfinite-conv-nlength-enat not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc the-enat.simps)
have *60*:*nfinite* (*cppl f g* (*nsubn* σ (*nnth ls* (*the-enat* (*nlength ls*) − *1*))
(*nnth ls* (*Suc* (*the-enat* (*nlength ls*) − *1*)))))
by (*metis 0 Suc-n-not-le-n add.commute assms*(*3*) *assms*(*6*) *assms*(*8*) *cppl-fprojection diff-add*
enat.distinct(*2*) *enat-ord-simps*(*1*) *enat-the-enat ileI1 nfinite-conv-nlength-enat*
not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc)
have *6*: *nlength* σ >*0* ⟶
nlast ((*nmap* (λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
(*cppl f g* (*nsubn* σ (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))) =
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
(*nlast* (*cppl f g* (*nsubn* σ (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))))))
using *nlast-nmap*[*of* (*cppl f g* (*nsubn* σ (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*)))))
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))] *60* **by** *blast*
have *7*: *nlength* σ >*0* ⟶
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
(*nlast* (*cppl f g* (*nsubn* σ (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
(*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*)))))) =
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
((*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) − (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
using *4 5* **by** *auto*
have *8*: *nlength* σ >*0* ⟶
(λ*x*. *x*+ (*nnth ls* (*the-enat*(*nlength ls*) −*1*)))
((*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) − (*nnth ls* (*the-enat*(*nlength ls*) −*1*))) =
((*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) −
(*nnth ls* (*the-enat*(*nlength ls*) −*1*))) + (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
by (*simp add*: *shift-def*)
have *9*: *nlength* σ >*0* ⟶
((*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*))) −
(*nnth ls* (*the-enat*(*nlength ls*) −*1*))) + (*nnth ls* (*the-enat*(*nlength ls*) −*1*))
= (*nnth ls* (*Suc* (*the-enat*(*nlength ls*) −*1*)))
using *assms 0*
by (*metis Suc-diff-1 cancel-comm-monoid-add-class.diff-cancel diff-add diff-is-0-eq enat-ord-simps*(*1*)

enat-ord-simps(*2*) *nfinite-nlength-enat nidx-expand order-less-imp-le the-enat.simps zero-enat-def*)
have *10*: *nlength σ >0 ⟶* (*nnth ls* (*Suc* (*the-enat*(*nlength ls*) *−1*))) = *nlength σ*
using *assms 0*
by (*metis One-nat-def add.commute diff-add eSuc-enat enat.distinct*(*2*) *enat-ord-simps*(*1*) *enat-the-enat ileI1 nfinite-conv-nlength-enat nnth-nlast plus-1-eq-Suc zero-enat-def*)
show *?thesis*
using *1 10 2 3 6 7 8 9 assms* **by** *presburger*
qed

lemma *PJ7helpchain1a-help-4*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(*nlength σ*)
(∀ *i<nlength ls.* (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
h (*pfilt σ ls*)
nlength σ >0
shows ((*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) ⊨ *g fproj h*)
proof −
have *0*: *nlength σ > 0 ⟶ nlength ls >0*
using *assms*
by (*metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *00*: *all-gr-zero* (*lcppl f g σ ls*)
using *all-gr-zero-nnth-a assms lcppl-nlength-all-gr-zero* **by** *blast*
have *01*: *all-nfinite* (*lcppl f g σ ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite* **by** *blast*
have *02*: *nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *0 = 0*
using *0 assms*
by (*metis addzero-def lcppl-lsum-nlength less-numeral-extra*(*3*) *nnth-0*)
have *1*: *nlength σ >0 ⟶ nidx* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) ∧ *nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *0 = 0*
using *assms lsum-addzero-nidx*[*of* (*lcppl f g σ ls*)] *lcppl-nlength-all-gr-zero*[*of ls σ f g*]
using *00 01 02* **by** *blast*
have *2*: *nlength σ > 0 ⟶*
nlast (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) =
the-enat(*nlength* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))))
using *assms lcppl-lsum-nlength*[*of ls σ f g*] *lcppl-nfusecat-pfilt-nlength*[*of ls σ f g*]
by *fastforce*
have *3*: *nlength σ >0 ⟶*
powerinterval g (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
by (*simp add*: *assms lcppl-nfusecat-pfilt-powerinterval*)
have *4*: *nlength σ >0 ⟶*
h (*pfilt* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)))
by (*simp add*: *assms pfilt-nmap-pfilt lcppl-pfilt-nfusecat-lsum*)
show *?thesis* **using** *1 2 3 4 assms fprojection-d-def*
by (*metis 0 00 01 lcppl-lsum-nlength lcppl-nfinite lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat nfinite-nmap nfusecat-nfinite pfilt-nmap*)
qed

lemma *PJ7helpchain1a-help-4-alt*:
assumes *nidx ls*
nnth ls 0 = 0
¬nfinite ls
¬nfinite σ
(∀ *i*<*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*)
h (*pfilt σ ls*)
shows ((*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) ⊨ *g oproj h*)
proof −
have *00*: *all-gr-zero* (*lcppl f g σ ls*)
using *all-gr-zero-nnth-a assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have *01*: *all-nfinite* (*lcppl f g σ ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt* **by** *blast*
have *02*: *nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *0 = 0*
using *assms*
by (*metis addzero-def lcppl-nfinite nfinite-lsum-conv-b nlength-eq-enat-nfiniteD nnth-0 zero-enat-def*)
have *1*: *nidx* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) ∧ *nnth* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)) *0 = 0*
using *assms lsum-addzero-nidx 00 01 02* **by** *blast*
have *3*: *powerinterval g* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
by (*simp add*: *assms lcppl-nfusecat-pfilt-powerinterval-alt*)
have *4*: *h* (*pfilt* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) (*addzero* (*lsum* (*lcppl f g σ ls*) *0*)))
by (*simp add*: *assms lcppl-pfilt-nfusecat-lsum-alt pfilt-nmap-pfilt*)
have *5*: ¬ *nfinite* (*addzero* (*lsum* (*lcppl f g σ ls*) *0*))
using *assms*
by (*simp add*: *lcppl-lsum-nlength-alt nfinite-conv-nlength-enat*)
have *6*: ¬*nfinite* (*nfusecat* (*lcppl f g σ ls*))
using *assms 00 01 lcppl-nfinite lcppl-nfusecat-nlastnfirst-alt nfusecat-nfinite* **by** *blast*
have *7*: ¬ *nfinite* (*pfilt σ* (*nfusecat* (*lcppl f g σ ls*)))
by (*simp add*: *6 pfilt-nmap*)
show *?thesis* **using** *1 3 4 assms* **unfolding** *oprojection-d-def*
using *5 7* **by** *blast*
qed

lemma *PJ7helpchain1a*:
assumes *nlength σ > 0*
(*σ* ⊨ (*f fproj g*) *fproj h*)
shows (*σ* ⊨ *f fproj* (*g fproj h*))
proof −
have *1*: *nlength σ > 0*
using *assms* **by** *auto*
have *2*: (∃ *ls*. *nidx ls* ∧ *nnth ls 0 = 0* ∧ *nfinite ls* ∧ *nfinite σ* ∧ *nlast ls = the-enat*(*nlength σ*) ∧
powerinterval (*LIFT*(*f fproj g*)) *σ ls* ∧
h (*pfilt σ ls*))
using *assms* **using** *cppl-fprojection*[*of LIFT*(*f fproj g*) *h σ*] **by** *blast*
obtain *ls* **where** *3*: *nidx ls* ∧ *nnth ls 0 = 0* ∧ *nfinite ls* ∧ *nfinite σ* ∧
nlast ls = the-enat(*nlength σ*) ∧
powerinterval (*LIFT*(*f fproj g*)) *σ ls* ∧
h (*pfilt σ ls*)
using *2* **by** *blast*

have *4*: *nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite σ* ∧ *nlast ls* = *the-enat*(*nlength σ*) ∧
(∀ *i* <*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*) ∧
h (*pfilt σ ls*)
using *3* **by** (*simp add*: *powerinterval-def*)
have *6*: *nlength ls*>*0*
using *1 4*
by (*metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have *7*: *nidx* (*nfusecat* (*lcppl f g σ ls*)) ∧ *nnth* (*nfusecat* (*lcppl f g σ ls*)) *0* = *0*
by (*metis* (*no-types*, *lifting*) *1 4 PJ7helpchain1a-help-1*)
have *8*: *powerinterval f σ* (*nfusecat* (*lcppl f g σ ls*))
using *4 6 PJ7helpchain1a-help-2 assms* **by** *auto*
have *9*: *nlast* (*nfusecat* (*lcppl f g σ ls*)) = *the-enat*(*nlength σ*)
by (*metis 1 4 PJ7helpchain1a-help-3 the-enat.simps*)
have *10*: ((*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) ⊨ *g fproj h*)
using *1 4 PJ7helpchain1a-help-4* **by** *blast*
show *?thesis*
using *10 7 8 9* **by** (*metis 3 fprojection-d-def nfinite-nmap pfilt-nmap*)
qed

lemma *OPJ7helpchain1a*:
assumes (*σ* ⊨ (*f fproj g*) *oproj h*)
shows (*σ* ⊨ *f oproj* (*g oproj h*))
proof −
have *2*: (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ ¬*nfinite ls* ∧ ¬*nfinite σ* ∧
powerinterval (*LIFT*(*f fproj g*)) *σ ls* ∧
h (*pfilt σ ls*))
using *assms* **using** *cppl-oprojection*[*of LIFT*(*f fproj g*) *h σ*] **by** *blast*
obtain *ls* **where** *3*: *nidx ls* ∧ *nnth ls 0* = *0* ∧ ¬*nfinite ls* ∧ ¬*nfinite σ* ∧
powerinterval (*LIFT*(*f fproj g*)) *σ ls* ∧
h (*pfilt σ ls*)
using *2* **by** *blast*
have *4*: *nidx ls* ∧ *nnth ls 0* = *0* ∧ ¬*nfinite ls* ∧ ¬*nfinite σ* ∧
(∀ *i* <*nlength ls*. (*nsubn σ* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⊨ *f fproj g*) ∧
h (*pfilt σ ls*)
using *3* **by** (*simp add*: *powerinterval-def*)
have *6*: *nlength ls*>*0*
by (*metis 4 gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def*)
have *7*: *nidx* (*nfusecat* (*lcppl f g σ ls*)) ∧ *nnth* (*nfusecat* (*lcppl f g σ ls*)) *0* = *0*
using *4 PJ7helpchain1a-help-1-alt*
by (*metis 6 OPJ6help4*)
have *8*: *powerinterval f σ* (*nfusecat* (*lcppl f g σ ls*))
using *4 PJ7helpchain1a-help-2-alt* **by** *blast*
have *10*: ((*pfilt σ* (*nfusecat* (*lcppl f g σ ls*))) ⊨ *g oproj h*)
using *4 PJ7helpchain1a-help-4-alt* **by** *blast*
show *?thesis*
using *10 7 8* **by** (*metis 3 nfinite-conv-nlength-enat oprojection-d-def pfilt-nlength*)
qed

lemma *PJ7helpchain1b*:
assumes *nlength* σ > *0*
(σ ⊨ *f fproj* (*g fproj h*))
shows (σ ⊨ (*f fproj g*) *fproj h*)
proof −
have *1*: *nlength* σ > *0*
using *assms* **by** *auto*
have *2*: (∃ *ls*. *nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite* σ ∧ *nlast ls* = *the-enat*(*nlength* σ) ∧
powerinterval f σ *ls* ∧
((*pfilt* σ *ls*) ⊨ *g fproj h*))
using *assms cppl-fprojection* **by** *blast*
obtain *ls* **where** *3*: *nidx ls* ∧ *nnth ls 0* = *0* ∧ *nfinite ls* ∧ *nfinite* σ ∧ *nlast ls* = *the-enat*(*nlength* σ) ∧
powerinterval f σ *ls* ∧
((*pfilt* σ *ls*) ⊨ *g fproj h*)
using *2* **by** *blast*
have *4*: (∃ *lsa*. *nidx lsa* ∧ *nnth lsa 0* = *0* ∧ *nfinite lsa* ∧ *nfinite* (*pfilt* σ *ls*) ∧
nlast lsa = *the-enat*(*nlength*(*pfilt* σ *ls*)) ∧
powerinterval g (*pfilt* σ *ls*) *lsa* ∧
((*pfilt* (*pfilt* σ *ls*) *lsa*) ⊨ *h*))
using *3* **using** *cppl-fprojection* **by** *blast*
obtain *lsa* **where** *5*: *nidx lsa* ∧ *nnth lsa 0* = *0* ∧ *nfinite lsa* ∧ *nfinite* (*pfilt* σ *ls*) ∧
nlast lsa = *the-enat*(*nlength*(*pfilt* σ *ls*)) ∧
powerinterval g (*pfilt* σ *ls*) *lsa* ∧
((*pfilt* (*pfilt* σ *ls*) *lsa*) ⊨ *h*)
using *4* **by** *blast*
have *6*: *nlength ls* >*0*
using *1 3*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *7*: *nlength*(*pfilt* σ *ls*) = *nlength ls*
by (*simp add*: *pfilt-nlength*)
have *8*: (*pfilt* (*pfilt* σ *ls*) *lsa*) = (*pfilt* σ (*pfilt ls lsa*))
using *pfilt-nmap-pfilt* **by** *blast*
have *9*: *nlast* (*pfilt ls lsa*) = *the-enat*(*nlength* σ)
by (*metis 3 5 7 nlast-nmap nnth-nlast pfilt-nmap*)
have *10*: *nlength lsa* >*0*
using *5 6 7*
by (*metis enat.distinct*(*2*) *enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have *11*: *nlength* (*pfilt ls lsa*) > *0*
by (*metis 10 pfilt-nlength*)
have *111*: *nnth* (*pfilt ls lsa*) *0* = *0*
by (*metis 3 5 pfilt-nnth zero-enat-def zero-le*)
have *112*: *nlength*(*pfilt ls lsa*) = *nlength lsa*
using *pfilt-expand* **by** *blast*
have *113*: (∀ *i*< *nlength lsa*. (*nnth* (*pfilt ls lsa*) *i*) = (*nnth ls* (*nnth lsa i*)))
by (*simp add*: *pfilt-nmap*)
have *114*: (∀ *i*< *nlength lsa*. (*nnth* (*pfilt ls lsa*) (*Suc i*)) = (*nnth ls* (*nnth lsa* (*Suc i*))))
by (*metis 112 eSuc-enat ileI1 pfilt-nnth*)
have *1141*: ⋀*j*. *j* ≤ *nlength lsa*⟶ *nnth lsa j* ≤ *nlength ls*
by (*metis 5 7 enat-ord-simps*(*1*) *ndropn-eq-NNil ndropn-nlast nfinite-conv-nlength-enat*

nidx-less-eq the-enat.simps)
have *115*: ($\forall$ *i*< *nlength lsa*. (*nnth ls* (*nnth lsa i*)) < (*nnth ls* (*nnth lsa* (*Suc i*))))
by (*metis 1141 3 5 eSuc-enat ileI1 less-imp-Suc-add nidx-expand nidx-less*)
have *12*: *nidx* (*pfilt ls lsa*) $\wedge$ *nnth* (*pfilt ls lsa*) *0* = *0*
by (*simp add*: *111 112 115 Suc-ile-eq nidx-expand pfilt-nnth*)
have *2021*: ($\forall$ *i*< *nlength lsa*. *nfinite* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))))
by (*simp add*: *nsubn-def1*)
have *2022*: $\bigwedge$*i*. *i*< *nlength lsa* $\Longrightarrow$ (*nnth ls* (*nnth lsa i*)) $\leq$ *nlength* σ
by (*metis 3 5 7 enat.distinct*(*2*) *enat-ord-simps*(*1*) *enat-the-enat*
nfinite-conv-nlength-enat nidx-all-le-nlast order-less-imp-le)
have *2023*: $\bigwedge$*j*. *j* $\leq$ *nlength lsa*$\longrightarrow$ *nnth lsa j* $\leq$ *nlength ls*
by (*metis 5 7 enat-ord-simps*(*1*) *ndropn-eq-NNil ndropn-nlast nfinite-conv-nlength-enat*
nidx-less-eq the-enat.simps)
have *203*: ($\forall$ *i*< *nlength lsa*. *nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) =
(*nnth ls* (*nnth lsa* (*Suc i*))) − (*nnth ls* (*nnth lsa i*)))
by (*metis 112 113 114 12 3 5 9 PJ6help1 le-add1 le-add-same-cancel1*
nfinite-conv-nlength-enat)
have *2041*: ($\forall$ *i*< *nlength lsa*. (*nnth lsa* (*Suc i*)) $\leq$ *nlength ls*)
by (*metis 2023 eSuc-enat ileI1*)
have *204*: ($\forall$ *i*< *nlength lsa*.
nlength (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) =
(*nnth lsa* (*Suc i*)) − (*nnth lsa i*))
by (*simp add*: *3 5 PJ6help1 pfilt-nlength*)
have *205*: ($\forall$ *i*< *nlength lsa*.
($\forall$ *j* $\leq$ (*nnth lsa* (*Suc i*)) − (*nnth lsa i*).
(*nnth* (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))) *j*) =
(*nnth ls* ((*nnth lsa i*) +*j*))
)
)
using *2041 5* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *2060*: ($\forall$ *i*< *nlength lsa*. (*nnth lsa i*) $\leq$ (*nnth lsa* (*Suc i*)))
using *5* **by** (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *206*: ($\forall$ *i*< *nlength lsa*.
($\forall$ *j* $\leq$ (*nnth lsa* (*Suc i*)) − (*nnth lsa i*).
(*nnth ls* (*nnth lsa i*)) $\leq$ (*nnth ls* ((*nnth lsa i*) +*j*))
)
)
using *2041 3 2060*
by (*metis le-add1 nidx-all-le-nlast nidx-less-eq nnth-beyond not-le-imp-less order-less-imp-le*)
have *207*: ($\forall$ *i*< *nlength lsa*.
($\forall$ *j* $\leq$ (*nnth lsa* (*Suc i*)) − (*nnth lsa i*).
(*nnth* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *j*) =
(*nnth ls* ((*nnth lsa i*) +*j*)) − (*nnth ls* (*nnth lsa i*))
))
using *204 205* **by** *auto*
have *208*: ($\forall$ *i*< *nlength lsa*.
(*nnth* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *0*) = *0*)
by (*simp add*: *207*)

have *209*: ($\forall$ *i*< *nlength lsa*.
nlast (*nmap* (λx. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) =
(*nnth ls* (*nnth lsa* (*Suc i*))) − (*nnth ls* (*nnth lsa i*))
)

using *204 207 5 2060* **by** (*simp add*: *nsubn-def1 ntaken-ndropn-nlast*)
have *210*: ($\forall$ *i*< *nlength lsa*.
nidx (*nmap* (λx. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) $\wedge$
nnth (*nmap* (λx. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *0* = *0*
)
using *3 5 7* *nidx-expand nidx-shiftm nidx-nsubn* **by** (*simp add*: *2041 Suc-ile-eq order-less-imp-le*)
have *20*: *powerinterval* (*LIFT*(*f fproj g*)) σ (*pfilt ls lsa*)
proof −
have *201*: *powerinterval* (*LIFT*(*f fproj g*)) σ (*pfilt ls lsa*) =
($\forall$ *i*< *nlength lsa*. (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) $\models$ *f* *fproj* *g*)
unfolding *powerinterval-def* **by** (*auto simp add*: *Suc-ile-eq pfilt-nlength pfilt-nnth*)
have *202*: ($\forall$ *i*< *nlength lsa*. (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) $\models$ *f* *fproj*
g) =
($\forall$ *i*< *nlength lsa*.
($\exists$ *ls1*. *nidx* *ls1* $\wedge$ *nnth ls1 0* = *0* $\wedge$ *nfinite ls1* $\wedge$
nfinite (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) $\wedge$
nlast ls1 = *the-enat*(*nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))) $\wedge$
powerinterval f (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1* $\wedge$
((*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1*) $\models$ *g*)
))

by (*simp add*: *fprojection-d-def*)
have *fg*: ($\forall$ *i*< *nlength lsa*.
($\exists$ *ls1*. *nidx* *ls1* $\wedge$ *nnth ls1 0* = *0* $\wedge$ *nfinite ls1* $\wedge$
nfinite (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) $\wedge$
nlast ls1 = *the-enat*(*nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))) $\wedge$
powerinterval f (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1* $\wedge$
((*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1*) $\models$ *g*)
))
proof *auto*
fix *i*::*nat*
assume *a*: *enat i* < *nlength lsa*
show $\exists$ *ls1*. *nidx ls1* $\wedge$
nnth ls1 0 = *0* $\wedge$
nfinite ls1 $\wedge$
nfinite (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) $\wedge$
nlast ls1 = *the-enat* (*nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))) $\wedge$
powerinterval f (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1* $\wedge$
g (*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1*)
proof −
have *fg1*: *nnth* (*nmap* (λx. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *0* = *0*
using *210 a* **by** *blast*
have *fg2*: *nidx* (*nmap* (λx. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))
using *210 a* **by** *blast*

have *fg3*: *nfinite* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
using *2021 a* **by** *blast*
have *fg4*: *nlast* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) =
the-enat (*nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))))
using *203 209 a the-enat.simps* **by** *presburger*
have *fg5*: *powerinterval f* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))
proof (*auto simp add*: *powerinterval-def*)
fix *ia*::*nat*
assume *aa*: *enat ia* < *nlength* (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))
show *f* (*nsubn* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nnth* (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))) *ia* − *nnth ls* (*nnth lsa i*))
(*nnth* (*nmap* (λ*x*. *x* − *nnth ls* (*nnth lsa i*)) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))
(*Suc ia*)))
proof −
have *f1*: *ia* < (*nnth lsa* (*Suc i*)) − (*nnth lsa i*)
by (*metis 5 7 PJ6help1 a aa enat-ord-simps*(*2*) *le-add1 le-add-same-cancel1 nsubn-nlength*)
have *f2*: *nsubn* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nnth ls* (*nnth lsa i* + *ia*) − *nnth ls* (*nnth lsa i*))
(*nnth ls* (*nnth lsa i* + *Suc ia*) − *nnth ls* (*nnth lsa i*)) =
nsubn σ (*nnth ls* (*nnth lsa i* + *ia*)) (*nnth ls* (*nnth lsa i* + *Suc ia*))
proof −
have *f3*: (*nnth ls* (*nnth lsa i* + *ia*) − *nnth ls* (*nnth lsa i*) ≤
nnth ls (*nnth lsa i* + *Suc ia*) − *nnth ls* (*nnth lsa i*))
by (*metis 2041 205 2060 3 Suc-leI a aa diff-le-mono eSuc-enat f1 ileI1*
le-add2 nidx-less-eq nidx-nsubn order-less-imp-le plus-1-eq-Suc)
have *f4*: *nnth ls* (*nnth lsa i*) ≤ *nnth ls* (*nnth lsa* (*Suc i*))
using *115 a dual-order.order-iff-strict* **by** *blast*
have *f5*: *enat* (*nnth ls* (*nnth lsa* (*Suc i*))) ≤ *nlength* σ
by (*metis 2041 3 a enat-ord-code*(*4*) *enat-ord-simps*(*1*) *enat-the-enat nidx-all-le-nlast*
order-less-imp-le)
have *f6*: *nnth ls* (*nnth lsa i* + *Suc ia*) − *nnth ls* (*nnth lsa i*) ≤
nnth ls (*nnth lsa* (*Suc i*)) − *nnth ls* (*nnth lsa i*)
by (*metis 2041 2060 3 Suc-leI a add-diff-cancel-left′ diff-le-mono f1 le-add1*
le-diff-iff nidx-less-eq)
show *?thesis* **using** *nsubn-nsubn-1*[*of* (*nnth ls* (*nnth lsa i* + *ia*) − *nnth ls* (*nnth lsa i*))
(*nnth ls* (*nnth lsa i* + *Suc ia*) − *nnth ls* (*nnth lsa i*))
(*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))) σ]
by (*metis 206 Suc-leI a f1 f3 f4 f5 f6 le-add-diff-inverse order-less-imp-le*)
qed
have *f7*: (*nnth* (*nmap* (λ*x*. *x* − *nnth ls* (*nnth lsa i*))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) (*Suc ia*)) =
(*nnth* (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))) (*Suc ia*) − *nnth ls* (*nnth lsa i*))
by (*metis aa eSuc-enat ileI1 nnth-nmap*)
have *f8*: *f* (*nsubn* σ (*nnth ls* ((*nnth lsa i*) +*ia*)) ((*nnth ls* ((*nnth lsa i*)+(*Suc ia*)))))
using *2041 3* **unfolding** *powerinterval-def*
by (*metis a add.commute add-Suc-right enat-ord-simps*(*2*) *f1 less-diff-conv*
order-less-le-trans)
show *?thesis* **using** *205 a f1 f2 f7 f8* **by** *fastforce*
qed

qed
have *fg6*: *g* (*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nmap* (λx. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))))
proof −
have *g1*: (*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nmap* (λx. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))) =
nmap (*nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))))
(*nmap* (λx. *x* − *nnth ls* (*nnth lsa i*)) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))
using *pfilt-nmap*[*of* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nmap* (λx. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))]
by *blast*
have *g2*: ... =
nmap (((*nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))))) $\circ$
(λx. *x*− (*nnth ls* (*nnth lsa i*))))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))
using *nellist.map-comp* **by** *blast*
have *g3*: ... =
(*nsubn* (*nmap* (((*nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))))) $\circ$
(λx. *x*− (*nnth ls* (*nnth lsa i*)))) *ls*)
(*nnth lsa i*) (*nnth lsa* (*Suc i*)))
unfolding *nsubn-def1* **by** (*simp add*: *ndropn-nmap*)
have *g4*: $\bigwedge$*y*. (((*nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))))) $\circ$
(λx. *x*− (*nnth ls* (*nnth lsa i*)))) *y* =
nnth (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*y* − *nnth ls* (*nnth lsa i*))
by *simp*
have *g5*: (*nsubn* (*nmap* (((*nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))))) $\circ$
(λx. *x*− (*nnth ls* (*nnth lsa i*)))) *ls*)
(*nnth lsa i*) (*nnth lsa* (*Suc i*))) =
(*nsubn* (*nmap* (λy. *nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*y* − *nnth ls* (*nnth lsa i*))) *ls*)
(*nnth lsa i*) (*nnth lsa* (*Suc i*)))
using *g4* **by** *presburger*
have *g6*: $\bigwedge$*j*. (*nnth lsa i*) $\le$*j* $\wedge$ *j* $\le$ (*nnth lsa* (*Suc i*)) $\longrightarrow$
nnth (*nmap* (λy. *nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*y* − *nnth ls* (*nnth lsa i*))) *ls*) *j* =
nnth (*nmap* (*nnth* σ) *ls*) *j*
proof
fix *j*::*nat*
assume *aaa*: *nnth lsa i* $\le$ *j* $\wedge$ *j* $\le$ *nnth lsa* (*Suc i*)
show *nnth* (*nmap* (λy. *nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*y* − *nnth ls* (*nnth lsa i*))) *ls*) *j* = *nnth* (*nmap* (*nnth* σ) *ls*) *j*
proof −
have *g8*: *nnth* (*nmap* (*nnth* σ) *ls*) *j* = *nnth* σ (*nnth ls j*)
by (*metis nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap*)
have *g9*: *nnth* (*nmap* (λy. *nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*y* − *nnth ls* (*nnth lsa i*))) *ls*) *j* =
(λy. *nnth* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*y* − *nnth ls* (*nnth lsa i*))) (*nnth ls j*)
by (*metis* (*no-types*, *lifting*) *nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap*)

have *g10*: ... =
nnth (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nnth ls j* $-$ *nnth ls* (*nnth lsa i*))
by *auto*
have *g11*: ... = *nnth* σ ((*nnth ls* (*nnth lsa i*)) + (*nnth ls j* $-$ *nnth ls* (*nnth lsa i*)))
using *nsubn-nnth*[*of* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))
(*nnth ls j* $-$ *nnth ls* (*nnth lsa i*))]
by (*simp add*: *2041 3 a aaa diff-le-mono nidx-less-eq*)
have *g12*: ... = *nnth* σ (*nnth ls j*)
using *aaa*
by (*metis 3 le-add-diff-inverse nidx-all-le-nlast nidx-less-eq nnth-beyond*
not-le-imp-less order-less-imp-le)
show *?thesis*
using *g11 g12 g8 g9* **by** *presburger*
qed
qed
have *g13*: (*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nmap* ($\lambda x.\ x-$ (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))) =
(*nsubn* (*pfilt* σ *ls*) (*nnth lsa i*) (*nnth lsa* (*Suc i*)))
by (*simp add*: *2041 2060 a g2 g3 g5 g6 nsubn-eq pfilt-nmap*)
show *?thesis* **using** *5 a* **by** (*simp add*: *g13 powerinterval-def*)
qed
show *?thesis*
using *203 2041 2060 209 210 a fg3 fg5 fg6 nsubn-nfinite* **by** *fastforce*
qed
qed
show *?thesis*
using *201 202 fg* **by** *blast*
qed
show *?thesis*
by (*metis 12 20 3 5 8 9 fprojection-d-def nfinite-nmap pfilt-nmap*)
qed

lemma *OPJ7helpchain1b*:
assumes ($\sigma \models f$ *oproj* (*g oproj h*))
shows ($\sigma \models$ (*f fproj g*) *oproj h*)
proof $-$
have *2*: ($\exists$ *ls*. *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
powerinterval f σ *ls* $\wedge$
((*pfilt* σ *ls*) $\models$ *g oproj h*))
using *assms cppl-oprojection* **by** *blast*
obtain *ls* **where** *3*: *nidx ls* $\wedge$ *nnth ls 0* = *0* $\wedge$ $\neg$*nfinite ls* $\wedge$ $\neg$*nfinite* σ $\wedge$
powerinterval f σ *ls* $\wedge$
((*pfilt* σ *ls*) $\models$ *g oproj h*)
using *2* **by** *blast*
have *4*: ($\exists$ *lsa*. *nidx lsa* $\wedge$ *nnth lsa 0* = *0* $\wedge$ $\neg$*nfinite lsa* $\wedge$ $\neg$*nfinite* (*pfilt* σ *ls*) $\wedge$
powerinterval g (*pfilt* σ *ls*) *lsa* $\wedge$
((*pfilt* (*pfilt* σ *ls*) *lsa*) $\models$ *h*))
using *3* **using** *cppl-oprojection* **by** *blast*

obtain *lsa* **where** *5*: $nidx\ \ lsa \wedge nnth\ lsa\ 0 = 0 \wedge \neg nfinite\ lsa \wedge \neg nfinite\ (pfilt\ \sigma\ ls) \wedge$
$powerinterval\ g\ (pfilt\ \sigma\ ls)\ lsa \wedge$
$((pfilt\ (pfilt\ \sigma\ ls)\ lsa) \models h)$
using *4* **by** *blast*
have *7*: $nlength(pfilt\ \sigma\ ls) = nlength\ ls$
by (*simp add*: *pfilt-nlength*)
have *8*: $(pfilt\ (pfilt\ \sigma\ ls)\ lsa) = (pfilt\ \sigma\ (pfilt\ ls\ lsa))$
using *pfilt-nmap-pfilt* **by** *blast*
have *11*: $nlength\ (pfilt\ ls\ lsa) > 0$
by (*metis 5 gr-zeroI nlength-eq-enat-nfiniteD pfilt-nlength zero-enat-def*)
have *111*: $nnth\ (pfilt\ ls\ lsa)\ 0 = 0$
by (*metis 3 5 pfilt-nnth zero-enat-def zero-le*)
have *112*: $nlength(pfilt\ ls\ lsa) = nlength\ lsa$
using *pfilt-expand* **by** *blast*
have *113*: $(\forall\ \ i< nlength\ lsa.\ (nnth\ (pfilt\ ls\ lsa)\ i) = (nnth\ ls\ (nnth\ lsa\ i)))$
by (*simp add*: *pfilt-nmap*)
have *114*: $(\forall\ \ i< nlength\ lsa.\ (nnth\ (pfilt\ ls\ lsa)\ (Suc\ i)) = (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
by (*metis 112 eSuc-enat ileI1 pfilt-nnth*)
have *1141*: $\bigwedge j.\ j \le nlength\ lsa \longrightarrow nnth\ lsa\ j \le nlength\ ls$
by (*meson 3 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
have *115*: $(\forall\ \ i< nlength\ lsa.\ (nnth\ ls\ (nnth\ lsa\ i)) < (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))\ )$
by (*metis 1141 3 5 eSuc-enat ileI1 less-imp-Suc-add nidx-expand nidx-less*)
have *12*: $nidx\ \ (pfilt\ ls\ lsa) \wedge nnth\ (pfilt\ ls\ lsa)\ 0 = 0$
by (*simp add*: *111 112 115 Suc-ile-eq nidx-expand pfilt-nnth*)
have *2021*: $(\forall\ \ i< nlength\ lsa.\ nfinite\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))))$
by (*simp add*: *nsubn-def1*)
have *2022*: $\bigwedge i.\ i< nlength\ lsa \Longrightarrow (nnth\ ls\ (nnth\ lsa\ i)) \le nlength\ \sigma$
by (*meson 3 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
have *203*: $(\forall\ \ i< nlength\ lsa.\ \ nlength\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))\ ) =$
$(nnth\ ls\ (nnth\ lsa\ (Suc\ i))) - (nnth\ ls\ (nnth\ lsa\ i)))$
by (*metis 112 113 114 12 3 5 OPJ6help1 le-add1 le-add-same-cancel1 nfinite-nmap pfilt-nmap*)
have *2041*: $(\forall\ \ i< nlength\ lsa.\ \ (nnth\ lsa\ (Suc\ i)) \le nlength\ ls\ )$
by (*metis 1141 eSuc-enat ileI1*)
have *204*: $(\forall\ \ i< nlength\ lsa.$
$nlength\ (nmap\ (\lambda x.\ x- (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ )) =$
$(nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i))$
by (*simp add*: *3 5 OPJ6help1*)
have *205*: $(\forall\ \ i< nlength\ lsa.$
$(\forall\ \ j \le (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i).$
$(nnth\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ )\ j) =$
$(nnth\ ls\ ((nnth\ lsa\ i)\ +j))$
$)$
$)$
using *2041 5* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *2060*: $(\forall\ \ i< nlength\ lsa.\ (nnth\ lsa\ i) \le (nnth\ lsa\ (Suc\ i))\ )$
using *5* **by** (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *206*: $(\forall\ \ i< nlength\ lsa.$
$(\forall\ \ j \le (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i).$
$(nnth\ ls\ (nnth\ lsa\ i)) \le (nnth\ ls\ ((nnth\ lsa\ i)\ +j))$

)
)
using *2041 3 2060* **by** (*simp add*: *nfinite-conv-nlength-enat nidx-less-eq*)
have *207*: (∀ *i*< *nlength lsa*.
(∀ *j* ≤ (*nnth lsa* (*Suc i*)) − (*nnth lsa i*).
(*nnth* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *j*) =
(*nnth ls* ((*nnth lsa i*) +*j*)) − (*nnth ls* (*nnth lsa i*))
))
using *204 205* **by** *auto*
have *208*: (∀ *i*< *nlength lsa*.
(*nnth* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *0*) = *0*)
by (*simp add*: *207*)
have *209*: (∀ *i*< *nlength lsa*.
nlast (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*)))
(*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) =
(*nnth ls* (*nnth lsa* (*Suc i*))) − (*nnth ls* (*nnth lsa i*))
)
using *204 207 5 2060* **by** (*simp add*: *nsubn-def1 ntaken-ndropn-nlast*)
have *210*: (∀ *i*< *nlength lsa*.
nidx (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) ∧
nnth (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *0* = *0*
)
using *3 5 7* *nidx-expand nidx-shiftm nidx-nsubn* **by** (*simp add*: *2041 Suc-ile-eq order-less-imp-le*)
have *20*: *powerinterval* (*LIFT*(*f fproj g*)) σ (*pfilt ls lsa*)
proof −
have *201*: *powerinterval* (*LIFT*(*f fproj g*)) σ (*pfilt ls lsa*) =
(∀ *i*< *nlength lsa*. (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ⊨ *f* *fproj* *g*)
unfolding *powerinterval-def* **by** (*auto simp add*: *Suc-ile-eq pfilt-nlength pfilt-nnth*)
have *202*: (∀ *i*< *nlength lsa*. (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ⊨ *f* *fproj*
g) =
(∀ *i*< *nlength lsa*.
(∃ *ls1*. *nidx* *ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧
nfinite (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ∧
nlast ls1 = *the-enat*(*nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))) ∧
powerinterval f (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1* ∧
((*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1*) ⊨ *g*)
))
by (*simp add*: *fprojection-d-def*)
have *fg*: (∀ *i*< *nlength lsa*.
(∃ *ls1*. *nidx* *ls1* ∧ *nnth ls1 0* = *0* ∧ *nfinite ls1* ∧
nfinite (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ∧
nlast ls1 = *the-enat*(*nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))) ∧
powerinterval f (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1* ∧
((*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1*) ⊨ *g*)
))
proof *auto*
fix *i*::*nat*
assume *a*: *enat i* < *nlength lsa*

show ∃ *ls1*. *nidx ls1* ∧
nnth ls1 0 = *0* ∧
nfinite ls1 ∧
nfinite (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) ∧
nlast ls1 = *the-enat* (*nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))) ∧
powerinterval f (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1* ∧
g (*pfilt* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))) *ls1*)
proof −
have *fg1*: *nnth* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) *0* = *0*
using *210 a* **by** *blast*
have *fg2*: *nidx* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))
using *210 a* **by** *blast*
have *fg3*: *nfinite* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
using *2021 a* **by** *blast*
have *fg4*: *nlast* (*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))) =
the-enat (*nlength* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*)))))
using *203 209 a the-enat.simps* **by** *presburger*
have *fg5*: *powerinterval f* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nmap* (λ*x*. *x*− (*nnth ls* (*nnth lsa i*))) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))
proof (*auto simp add*: *powerinterval-def*)
fix *ia*::*nat*
assume *aa*: *enat ia* < *nlength* (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*)))
show *f* (*nsubn* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nnth* (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))) *ia* − *nnth ls* (*nnth lsa i*))
(*nnth* (*nmap* (λ*x*. *x* − *nnth ls* (*nnth lsa i*)) (*nsubn ls* (*nnth lsa i*) (*nnth lsa* (*Suc i*))))
(*Suc ia*)))
proof −
have *f1*: *ia* < (*nnth lsa* (*Suc i*)) − (*nnth lsa i*)
by (*metis 3 5 OPJ6help1 a aa enat-ord-simps*(*2*) *le-add1 le-add-same-cancel1*)
have *f2*: *nsubn* (*nsubn* σ (*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))))
(*nnth ls* (*nnth lsa i* + *ia*) − *nnth ls* (*nnth lsa i*))
(*nnth ls* (*nnth lsa i* + *Suc ia*) − *nnth ls* (*nnth lsa i*)) =
nsubn σ (*nnth ls* (*nnth lsa i* + *ia*)) (*nnth ls* (*nnth lsa i* + *Suc ia*))
proof −
have *f3*: (*nnth ls* (*nnth lsa i* + *ia*) − *nnth ls* (*nnth lsa i*) ≤
nnth ls (*nnth lsa i* + *Suc ia*) − *nnth ls* (*nnth lsa i*))
by (*metis 2041 205 2060 3 Suc-leI a aa diff-le-mono eSuc-enat f1 ileI1
le-add2 nidx-less-eq nidx-nsubn order-less-imp-le plus-1-eq-Suc*)
have *f4*: *nnth ls* (*nnth lsa i*) ≤ *nnth ls* (*nnth lsa* (*Suc i*))
using *115 a dual-order.order-iff-strict* **by** *blast*
have *f5*: *enat* (*nnth ls* (*nnth lsa* (*Suc i*))) ≤ *nlength* σ
by (*metis 2022 5 a eSuc-enat ileI1 nfinite-conv-nlength-enat order-le-imp-less-or-eq*)
have *f6*: *nnth ls* (*nnth lsa i* + *Suc ia*) − *nnth ls* (*nnth lsa i*) ≤
nnth ls (*nnth lsa* (*Suc i*)) − *nnth ls* (*nnth lsa i*)
by (*metis 2041 2060 3 Suc-leI a add-diff-cancel-left′ diff-le-mono f1 le-add1
le-diff-iff nidx-less-eq*)
show *?thesis* **using** *nsubn-nsubn-1*[*of* (*nnth ls* (*nnth lsa i* + *ia*) − *nnth ls* (*nnth lsa i*))
(*nnth ls* (*nnth lsa i* + *Suc ia*) − *nnth ls* (*nnth lsa i*))
(*nnth ls* (*nnth lsa i*)) (*nnth ls* (*nnth lsa* (*Suc i*))) σ]
by (*metis 206 Suc-leI a f1 f3 f4 f5 f6 le-add-diff-inverse order-less-imp-le*)

qed
have *f7*: $(nnth\ (nmap\ (\lambda x.\ x - nnth\ ls\ (nnth\ lsa\ i))$
$(nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))))\ (Suc\ ia)) =$
$(nnth\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))\ (Suc\ ia) - nnth\ ls\ (nnth\ lsa\ i))$
by (*metis aa eSuc-enat ileI1 nnth-nmap*)
have *f8*: $f\ (nsubn\ \sigma\ (nnth\ ls\ ((nnth\ lsa\ i)\ +ia))\ ((nnth\ ls\ ((nnth\ lsa\ i)+(Suc\ ia))))\ )$
using *2041 3* **unfolding** *powerinterval-def*
by (*metis a add.commute add-Suc-right enat-ord-simps*(*2*) *f1 less-diff-conv order-less-le-trans*)
show *?thesis* **using** *205 a f1 f2 f7 f8* **by** *fastforce*
qed
qed
have *fg6*: $g\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(nmap\ (\lambda x.\ x-\ (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ )))$
proof −
have *g1*: $(pfilt\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(nmap\ (\lambda x.\ x-\ (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ ))) =$
$nmap\ (nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))))$
$(nmap\ (\lambda x.\ x - nnth\ ls\ (nnth\ lsa\ i))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))))$
using *pfilt-nmap*[*of* $(nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(nmap\ (\lambda x.\ x-\ (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ ))$]
by *blast*
have *g2*: ... =
$nmap\ (((nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))))\ \circ$
$(\lambda x.\ x-\ (nnth\ ls\ (nnth\ lsa\ i))))$
$(nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))$
using *nellist.map-comp* **by** *blast*
have *g3*: ... =
$(nsubn\ (nmap\ (((nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))))\ \circ$
$(\lambda x.\ x-\ (nnth\ ls\ (nnth\ lsa\ i))))\ ls)$
$(nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))$
unfolding *nsubn-def1* **by** (*simp add*: *ndropn-nmap*)
have *g4*: $\bigwedge y.\ (((nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))))\ \circ$
$(\lambda x.\ x-\ (nnth\ ls\ (nnth\ lsa\ i))))\ y =$
$nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(y - nnth\ ls\ (nnth\ lsa\ i))$
by *simp*
have *g5*: $(nsubn\ (nmap\ (((nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))))\ \circ$
$(\lambda x.\ x-\ (nnth\ ls\ (nnth\ lsa\ i))))\ ls)$
$(nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))) =$
$(nsubn\ (nmap\ (\lambda y.\ nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(y - nnth\ ls\ (nnth\ lsa\ i)))\ ls)$
$(nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))$
using *g4* **by** *presburger*
have *g6*: $\bigwedge j.\ (nnth\ lsa\ i) \le j \wedge j \le (nnth\ lsa\ (Suc\ i)) \longrightarrow$
$nnth\ (nmap\ (\lambda y.\ nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(y - nnth\ ls\ (nnth\ lsa\ i)))\ ls)\ j =$
$nnth\ (nmap\ (nnth\ \sigma)\ ls)\ j$
proof
fix *j*::*nat*

assume *aaa*: $nnth\ lsa\ i \le j \wedge j \le nnth\ lsa\ (Suc\ i)$
show $nnth\ (nmap\ (\lambda y.\ nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(y - nnth\ ls\ (nnth\ lsa\ i)))\ ls)\ j = nnth\ (nmap\ (nnth\ \sigma)\ ls)\ j$
proof −
have *g8*: $nnth\ (nmap\ (nnth\ \sigma)\ ls)\ j = nnth\ \sigma\ (nnth\ ls\ j)$
by (*metis nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap*)
have *g9*: $nnth\ (nmap\ (\lambda y.\ nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(y - nnth\ ls\ (nnth\ lsa\ i)))\ ls)\ j =$
$(\lambda y.\ nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(y - nnth\ ls\ (nnth\ lsa\ i)))\ (nnth\ ls\ j)$
by (*metis* (*no-types*, *lifting*) *nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap*)
have *g10*: ... =
$nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(nnth\ ls\ j - nnth\ ls\ (nnth\ lsa\ i))$
by *auto*
have *g11*: ... $= nnth\ \sigma\ ((nnth\ ls\ (nnth\ lsa\ i)) + (nnth\ ls\ j - nnth\ ls\ (nnth\ lsa\ i)))$
using *nsubn-nnth*[*of* σ $(nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))$
$(nnth\ ls\ j - nnth\ ls\ (nnth\ lsa\ i))$]
by (*simp add*: *2041 3 a aaa diff-le-mono nidx-less-eq*)
have *g12*: ... $= nnth\ \sigma\ (nnth\ ls\ j)$
using *aaa*
by (*metis 206 a add-le-imp-le-diff diff-add le-add-diff-inverse*)
show *?thesis*
using *g11 g12 g8 g9* **by** *presburger*
qed
qed
have *g13*: $(pfilt\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
$(nmap\ (\lambda x.\ x-\ (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ ))) =$
$(nsubn\ (pfilt\ \sigma\ ls)\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))\ )$
by (*simp add*: *2041 2060 a g2 g3 g5 g6 nsubn-eq pfilt-nmap*)
show *?thesis* **using** *5 a* **by** (*simp add*: *g13 powerinterval-def*)
qed
show *?thesis*
using *203 2041 2060 209 210 a fg3 fg5 fg6 nsubn-nfinite* **by** *fastforce*
qed
qed
show *?thesis*
using *201 202 fg* **by** *blast*
qed
show *?thesis*
by (*metis 12 20 3 5 8 nfinite-nmap oprojection-d-def pfilt-nmap*)
qed

lemma *PJ7sem*:
$(\sigma \models f\ \ fproj\ \ (g\ \ fproj\ \ h) = (f\ \ fproj\ \ g)\ \ fproj\ \ h)$
proof −
have *1*: $nlength\ \sigma > 0 \longrightarrow (\sigma \models f\ \ fproj\ \ (g\ \ fproj\ \ h) = (f\ \ fproj\ \ g)\ \ fproj\ \ h)$
using *PJ7helpchain1a PJ7helpchain1b unl-lift2* **by** *blast*
have *2*: $nlength\ \sigma = 0 \longrightarrow (\sigma \models f\ \ fproj\ \ (g\ \ fproj\ \ h) = (f\ \ fproj\ \ g)\ \ fproj\ \ h)$
using *PJ7empty* **by** *blast*

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

lemma *OPJ7sem*:
$(\sigma \models f\ oproj\ (g\ oproj\ h) = (f\ \ fproj\ \ g)\ oproj\ h)$
using *OPJ7helpchain1a OPJ7helpchain1b unl-lift2* **by** *blast*

10.3.8 PJ8

lemma *PJ8semhelp*:
assumes *nidx ls*
nnth ls 0 = 0
nfinite ls
nfinite σ
nlast ls = the-enat(nlength σ*)*
$(\forall\ n\ na.\ na + n \le nlength\ \sigma \longrightarrow f\ (nsubn\ \sigma\ n\ (n+\ na)\ ) \longrightarrow g\ (nsubn\ \sigma\ n\ (n + na)\ ))$
shows
$(\forall\ i{<}nlength\ ls.\ f\ (nsubn\ \sigma\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )$
$\longrightarrow g\ (nsubn\ \sigma\ \ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )$
$)$
using *assms* **unfolding** *nidx-expand* **by** *auto*
(*metis assms*(*1*) *diff-add eSuc-enat enat-ord-simps*(*1*) *ileI1 le-add-diff-inverse less-imp-le-nat nfinite-nlength-enat nidx-all-le-nlast the-enat.simps*)

lemma *PJ8semhelp-alt*:
assumes *nidx ls*
nnth ls 0 = 0
$\neg$*nfinite ls*
$\neg$*nfinite* σ
$(\forall\ n\ na.\ na + n \le nlength\ \sigma \longrightarrow f\ (nsubn\ \sigma\ n\ (n+\ na)\ ) \longrightarrow g\ (nsubn\ \sigma\ n\ (n + na)\ ))$
shows
$(\forall\ i{<}nlength\ ls.\ f\ (nsubn\ \sigma\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )$
$\longrightarrow g\ (nsubn\ \sigma\ \ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )$
$)$
using *assms* **unfolding** *nidx-expand*
by *auto*
(*metis enat-ord-simps*(*3*) *enat-the-enat le-add-diff-inverse nlength-eq-enat-nfiniteD order-less-imp-le*)

lemma *PJ8sem*:
$(\sigma \models ba(f \longrightarrow g) \longrightarrow (f\ \ fproj\ \ h)\ \ \longrightarrow (g\ \ fproj\ \ h))$
using *PJ8semhelp*
by (*simp add*: *fprojection-d-def ba-defs powerinterval-def*)
(*metis eSuc-enat enat-ord-simps*(*1*) *ileI1 le-add-diff-inverse linorder-le-cases nfinite-nlength-enat nidx-expand nidx-less-eq nnth-nlast order-less-imp-le the-enat.simps*)

lemma *OPJ8sem*:
$(\sigma \models ba(f \longrightarrow g) \longrightarrow (f\ oproj\ h)\ \ \longrightarrow (g\ oproj\ h))$
using *PJ8semhelp-alt*
by (*simp add*: *oprojection-d-def ba-defs powerinterval-def*)
(*metis eSuc-enat enat-ord-code*(*4*) *enat-the-enat ileI1 le-add-diff-inverse nidx-expand*

nlength-eq-enat-nfiniteD order-less-imp-le order-less-imp-le)

10.3.9 PJ9

lemma *PJ9sem*:
($\sigma \models f$ *ufproj* ($g \longrightarrow h$) $\longrightarrow f$ *fproj* $g \longrightarrow f$ *fproj* h)
by (*simp add*: *ufprojection-d-def fprojection-d-def*)
(*metis less-numeral-extra*(*3*))

lemma *OPJ9sem*:
($\sigma \models f$ *uoproj* ($g \longrightarrow h$) $\longrightarrow f$ *oproj* $g \longrightarrow f$ *oproj* h)
by (*simp add*: *uoprojection-d-def oprojection-d-def*)
(*metis less-numeral-extra*(*3*))

10.4 Axioms

lemma *FBpGen*:
assumes $\vdash f$
shows $\vdash$ *fbp* f
using *assms*
by (*simp add*: *fbp-d-def ufprojection-d-def fprojection-d-def Valid-def*)

lemma *OBpGen*:
assumes $\vdash f$
shows $\vdash$ *obp* f
using *assms*
by (*simp add*: *obp-d-def uoprojection-d-def oprojection-d-def Valid-def*)

lemma *PJ00*:
$\vdash \neg$(f *fproj* *inf*)
using *PJ00sem Valid-def* **by** *blast*

lemma *OPJ00*:
$\vdash \neg$(f *oproj finite*)
using *OPJ00sem Valid-def* **by** *blast*

lemma *PJ01*:
$\vdash$ (f *fproj* g) $\longrightarrow$ *finite*
using *PJ01sem Valid-def* **by** *blast*

lemma *OPJ01*:
$\vdash$ (f *oproj* g) $\longrightarrow$ *inf*
using *OPJ01sem Valid-def* **by** *blast*

lemma *PJ02*:
$\vdash$ (f *fproj* g) = (($f \wedge$ *finite*) *fproj* g)
using *PJ02sem Valid-def* **by** *blast*

lemma *OPJ02*:
$\vdash$ (f *oproj* g) = (($f \wedge$ *finite*) *oproj* g)
using *OPJ02sem Valid-def* **by** *blast*

lemma *PJ03*:
⊢ $(f\ \ fproj\ \ g) = (f\ \ fproj\ \ (g \wedge finite))$
using *PJ03sem Valid-def* **by** *blast*

lemma *OPJ03*:
⊢ $(f\ oproj\ g) = (f\ oproj\ (g \wedge inf))$
using *OPJ03sem Valid-def* **by** *blast*

lemma *PJ1*:
⊢ $f\ \ fproj\ \ (\ g \vee h) = (f\ \ fproj\ \ g \vee f\ \ fproj\ \ h)$
using *PJ1sem Valid-def* **by** *blast*

lemma *OPJ1*:
⊢ $f\ oproj\ (\ g \vee h) = (f\ oproj\ g \vee f\ oproj\ h)$
using *OPJ1sem Valid-def* **by** *blast*

lemma *PJ2*:
⊢ $f\ \ fproj\ \ empty = empty$
using *PJ2sem Valid-def* **by** *blast*

lemma *OPJ2*:
⊢ $\neg(f\ oproj\ empty)$
using *OPJ2sem Valid-def* **by** *blast*

lemma *PJ3*:
⊢ $f\ \ fproj\ \ skip = (f \wedge more \wedge finite)$
using *PJ3sem Valid-def* **by** *blast*

lemma *OPJ3*:
⊢ $\neg(f\ oproj\ skip)$
using *OPJ3sem Valid-def* **by** *blast*

lemma *PJ4*:
⊢ $f\ \ fproj\ \ (g;h) = (f\ \ fproj\ \ g)\ ;\ (\ f\ \ fproj\ \ h)$
using *PJ4sem Valid-def* **by** *blast*

lemma *OPJ4*:
⊢ $f\ oproj\ ((g \wedge finite);h) = (f\ fproj\ g)\ ;\ (\ f\ oproj\ h)$
using *OPJ4sem Valid-def* **by** *blast*

lemma *PJ5*:
⊢ $f\ \ fproj\ \ init(g) \longrightarrow init(g)$
using *PJ5sem Valid-def* **by** *blast*

lemma *OPJ5*:
⊢ $f\ oproj\ init(g) \longrightarrow init(g)$
using *OPJ5sem Valid-def* **by** *blast*

lemma *PJ6*:
$\vdash$ *skip fproj g* = (*g* $\wedge$ *finite*)
using *PJ6sem Valid-def* **by** *blast*

lemma *OPJ6*:
$\vdash$ *skip oproj g* = (*g* $\wedge$ *inf*)
using *OPJ6sem Valid-def* **by** *blast*

lemma *PJ7*:
$\vdash$ *f fproj* (*g fproj h*) = (*f fproj g*) *fproj h*
using *PJ7sem Valid-def* **by** *blast*

lemma *OPJ7*:
$\vdash$ *f oproj* (*g oproj h*) = (*f fproj g*) *oproj h*
using *OPJ7sem Valid-def* **by** *blast*

lemma *PJ8*:
$\vdash$ *ba*(*f* $\longrightarrow$ *g*) $\longrightarrow$ (*f fproj h*) $\longrightarrow$ (*g fproj h*)
using *PJ8sem Valid-def* **by** *blast*

lemma *OPJ8*:
$\vdash$ *ba*(*f* $\longrightarrow$ *g*) $\longrightarrow$ (*f oproj h*) $\longrightarrow$ (*g oproj h*)
using *OPJ8sem Valid-def* **by** *blast*

lemma *PJ9*:
$\vdash$ *f ufproj* (*g* $\longrightarrow$ *h*) $\longrightarrow$ *f fproj g* $\longrightarrow$ *f fproj h*
using *PJ9sem Valid-def* **by** *blast*

lemma *OPJ9*:
$\vdash$ *f uoproj* (*g* $\longrightarrow$ *h*) $\longrightarrow$ *f oproj g* $\longrightarrow$ *f oproj h*
using *OPJ9sem Valid-def* **by** *blast*

10.5 Theorems

10.5.1 Projection

lemma *FPowerFProjLen*:
$\vdash$ *f fproj len n* = *fpower* (*f* $\wedge$ *more*) *n*
proof
(*induct n*)
case *0*
then show *?case*
by (*metis PJ2 fpower-d-def len-d-def wpow-0*)
next
case (*Suc n*)
then show *?case*
by (*metis AndMoreAndFiniteEqvAndFmore FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite PJ3 PJ4sem fpower-d-def intI inteq-reflection len-d-def lift-and-com wpow-Suc*)
qed

lemma *FProjLenExist*:
$\vdash$ *f fproj* ($\exists$ *n. len n*) = ($\exists$ *n. f fproj len n*)
by (*simp add*: *Valid-def fprojection-d-def*, *blast*)

lemma *FPowerFProjLenExist*:
$\vdash$ ($\exists$ *n. f fproj len n*) = ($\exists$ *n. fpower* (*f* $\wedge$ *more*) *n*)
using *FPowerFProjLen* **by** (*simp add*: *Valid-def FPowerFProjLen*, *blast*)

lemma *RightFProjImpFProj*:
assumes $\vdash$ *g1* $\longrightarrow$ *g2*
shows $\vdash$ *f fproj g1* $\longrightarrow$ *f fproj g2*
using *assms*
by (*simp add*: *Valid-def fprojection-d-def*, *blast*)

lemma *RightOProjImpOProj*:
assumes $\vdash$ *g1* $\longrightarrow$ *g2*
shows $\vdash$ *f oproj g1* $\longrightarrow$ *f oproj g2*
using *assms*
by (*simp add*: *Valid-def oprojection-d-def*, *blast*)

lemma *LeftFProjImpFProj*:
assumes $\vdash$ *f1* $\longrightarrow$ *f2*
shows $\vdash$ *f1 fproj g* $\longrightarrow$ *f2 fproj g*
using *assms*
by (*simp add*: *Valid-def fprojection-d-def powerinterval-def*, *blast*)

lemma *LeftOProjImpOProj*:
assumes $\vdash$ *f1* $\longrightarrow$ *f2*
shows $\vdash$ *f1 oproj g* $\longrightarrow$ *f2 oproj g*
using *assms*
by (*simp add*: *Valid-def oprojection-d-def powerinterval-def*, *blast*)

lemma *RightFProjEqvFProj*:
assumes $\vdash$ *g1* = *g2*
shows $\vdash$ *f fproj g1* = *f fproj g2*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *RightOProjEqvOProj*:
assumes $\vdash$ *g1* = *g2*
shows $\vdash$ *f oproj g1* = *f oproj g2*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *LeftFProjEqvFProj*:
assumes $\vdash$ *f1* = *f2*
shows $\vdash$ *f1 fproj g* = *f2 fproj g*
using *assms*

by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *LeftOProjEqvOProj*:
assumes $\vdash$ *f1* = *f2*
shows $\vdash$ *f1 oproj g* = *f2 oproj g*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *Valid-def inteq-reflection unl-lift2*)

lemma *FProjTrueEqvSChopstar*:
$\vdash$ *f fproj #True* = (*schopstar f*)
proof −
have *1*: $\vdash$ *finite* = (($\exists$ *n*. *len n*))
by (*simp add*: *Finite-exist-len*)
have *2*: $\vdash$ *f fproj #True* = *f fproj* ($\exists$ *n*. *len n*)
using *1*
by (*metis PJ03 Prop03 Prop10 int-simps*(*29*) *inteq-reflection lift-and-com*)
have *3*: $\vdash$ *f fproj* ($\exists$ *n*. *len n*) = ($\exists$ *n*. *fpower* (*f* $\wedge$ *more*) *n*)
using *FPowerFProjLenExist FProjLenExist* **by** *fastforce*
have *4*: $\vdash$ ($\exists$ *n*. *fpower* (*f* $\wedge$ *more*) *n*) = (*schopstar f*)
by (*metis FPowerstardef int-eq schopstar-d-def*)
show *?thesis* **using** *2 3 4* **by** *fastforce*
qed

lemma *OProjTrueEqvAOmega*:
$\vdash$ *f oproj #True* = (*aomega f*)
using *infinite-nidx-imp-infinite-interval*
by (*auto simp add*: *Valid-def oprojection-d-def aomega-d-def powerinterval-def*)
(*meson enat-ile linorder-le-cases nfinite-conv-nlength-enat not-le-imp-less*, *blast*)

lemma *OProjTrueEqvOmega*:
$\vdash$ *f oproj #True* = (*omega f*)
by (*metis OProjTrueEqvAOmega OmegaEqvAOmega int-eq*)

lemma *FProjSChopstarEqvSChopstarFProj*:
$\vdash$ *f fproj* (*schopstar g*) = (*schopstar* (*f fproj g*))
proof −
have *1*: $\vdash$ *f fproj* (*schopstar g*) = *f fproj* (*g fproj #True*)
by (*metis FProjTrueEqvSChopstar RightFProjEqvFProj inteq-reflection*)
have *2*: $\vdash$ *f fproj* (*g fproj #True*) = (*f fproj g*) *fproj #True*
by (*simp add*: *PJ7*)
have *3*: $\vdash$ (*f fproj g*) *fproj #True* = (*schopstar* (*f fproj g*))
by (*simp add*: *FProjTrueEqvSChopstar*)
show *?thesis* **using** *1 2 3* **by** *fastforce*
qed

lemma *OProjOmegaEqvOmegaFProj*:
$\vdash$ *f oproj* (*omega g*) = (*omega* (*f fproj g*))
proof −
have *1*: $\vdash$ *f oproj* (*omega g*) = *f oproj* (*g oproj #True*)
by (*metis OProjTrueEqvOmega RightOProjEqvOProj inteq-reflection*)

have *2*: ⊢ *f oproj* (*g oproj* #*True*) = (*f fproj g*) *oproj* #*True*
by (*simp add*: *OPJ7*)
have *3*: ⊢ (*f fproj g*) *oproj* #*True* = (*omega* (*f fproj g*))
by (*simp add*: *OProjTrueEqvOmega*)
show *?thesis* **using** *1 2 3* **by** *fastforce*
qed

lemma *OProjAndImp*:
⊢ *f oproj* (*g1* ∧ *g2*) ⟶ *f oproj g1* ∧ *f oproj g2*
by (*meson Prop12 RightOProjImpOProj int-iffD1 lift-and-com*)

lemma *FProjAndImp*:
⊢ *f fproj* (*g1* ∧ *g2*) ⟶ *f fproj g1* ∧ *f fproj g2*
by (*meson Prop12 RightFProjImpFProj int-iffD1 lift-and-com*)

lemma *FProjOrDist*:
⊢ #*True fproj* (*f* ∨ *g*) = (#*True fproj f* ∨ #*True fproj g*)
using *PJ1* **by** *blast*

lemma *OProjOrDist*:
⊢ #*True oproj* (*f* ∨ *g*) = (#*True oproj f* ∨ #*True oproj g*)
using *OPJ1* **by** *blast*

lemma *StateImportFProj*:
⊢ ((*init w*) ∧ *f fproj g*) = *f fproj* ((*init w*) ∧ *g*)
by (*auto simp add*: *Valid-def init-defs fprojection-d-def pfilt-nnth nidx-expand*)
(*metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le*,
metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le)

lemma *StateImportOProj*:
⊢ ((*init w*) ∧ *f oproj g*) = *f oproj* ((*init w*) ∧ *g*)
by (*auto simp add*: *Valid-def init-defs oprojection-d-def pfilt-nnth nidx-expand*)
(*metis ndropn-0 ndropn-nfirst pfilt-nlength pfilt-nnth zero-enat-def zero-le*,
metis ndropn-0 ndropn-nfirst pfilt-nlength pfilt-nnth zero-enat-def zero-le)

lemma *FProjStateAndNextEqvStateAndMoreChopFProj*:
⊢ *f fproj* ((*init w*) ∧ ○ *g*) = ((*init w*) ∧ (*f* ∧ *more* ∧ *finite*);(*f fproj g*))
proof −
have *2*: ⊢ (*f* ∧ *more* ∧ *finite*);(*f fproj g*) = *f fproj* ○ *g*
using *PJ3 PJ4* **unfolding** *next-d-def* **by** (*metis inteq-reflection*)
have *3*: ⊢ *f fproj* ((*init w*)) ⟶ *init w*
by (*simp add*: *PJ5*)
have *4*: ⊢ (*init w* ∧ *f fproj* ○ *g*) = *f fproj* ((*init w*) ∧ ○ *g*)
by (*simp add*: *StateImportFProj*)
have *5*: ⊢ *f fproj* ((*init w*) ∧ ○ *g*) ⟶ ((*init w*) ∧ (*f* ∧ *more* ∧ *finite*);(*f fproj g*))
using *2 3 FProjAndImp* **by** *fastforce*
from *5 4* **show** *?thesis* **using** *2* **by** *fastforce*
qed

lemma *OProjStateAndNextEqvStateAndMoreChopFProj*:
⊢ *f oproj* ((*init w*) ∧ ○ *g*) = ((*init w*) ∧ (*f* ∧ *more* ∧ *finite*);(*f oproj g*))
proof –
have *1*: ⊢ (*skip* ∧ *finite*) = *skip*
using *itl-defs*(*1*) *itl-defs*(*5*) *nlength-eq-enat-nfiniteD* **by** *fastforce*
have *2*: ⊢ (*f* ∧ *more* ∧ *finite*);(*f oproj g*) = *f oproj* ○ *g*
using *PJ3 OPJ4* **unfolding** *next-d-def* **by** (*metis 1 inteq-reflection*)
have *3*: ⊢ *f oproj* ((*init w*)) ⟶ *init w*
by (*simp add*: *OPJ5*)
have *4*: ⊢ (*init w* ∧ *f oproj* ○ *g*) = *f oproj* ((*init w*) ∧ ○ *g*)
by (*simp add*: *StateImportOProj*)
have *5*: ⊢ *f oproj* ((*init w*) ∧ ○ *g*) ⟶ ((*init w*) ∧ (*f* ∧ *more* ∧ *finite*);(*f oproj g*))
using *2 3 OProjAndImp* **by** *fastforce*
from *5 4* **show** *?thesis* **using** *2* **by** *fastforce*
qed

lemma *FProjNext*:
⊢ *f fproj* ○ *g* = (*f* ∧ *more* ∧ *finite*);(*f fproj g*)
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)

lemma *OProjNext*:
⊢ *f oproj* ○ *g* = (*f* ∧ *more* ∧ *finite*);(*f oproj g*)
by (*metis DiamondEmptyEqvFinite DiamondEqvEmptyOrNextDiamond FiniteChopEqvDiamond FiniteChopSkipEqvSkipChopFinite NowImpDiamond OPJ4 PJ3 Prop05 Prop10 SkipChopEqvNext inteq-reflection*)

lemma *FProjWnext*:
⊢ *f fproj* (*wnext g*) = (*empty* ∨ (*f* ∧ *more* ∧ *finite*);(*f fproj g*))
proof –
have *1*: ⊢ *f fproj* (*wnext g*) = *f fproj* (*empty* ∨ ○ *g*)
by (*simp add*: *RightFProjEqvFProj WnextEqvEmptyOrNext*)
have *2*: ⊢ *f fproj* (*empty* ∨ ○ *g*) = (*empty* ∨ *f fproj* (○ *g*))
using *PJ1 PJ2* **by** *fastforce*
have *3*: ⊢ *f fproj* (○ *g*) = (*f* ∧ *more* ∧ *finite*);(*f fproj g*)
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)
show *?thesis*
using *1 2 3* **by** *fastforce*
qed

lemma *OProjWnext*:
⊢ *f oproj* (*wnext g*) = ((*f* ∧ *more* ∧ *finite*);(*f oproj g*))
proof –
have *1*: ⊢ *f oproj* (*wnext g*) = *f oproj* (*empty* ∨ ○ *g*)
by (*simp add*: *RightOProjEqvOProj WnextEqvEmptyOrNext*)
have *2*: ⊢ *f oproj* (*empty* ∨ ○ *g*) = (*f oproj* (○ *g*))
using *OPJ2*[*of f*] *OPJ1*[*of f LIFT empty LIFT* ○ *g*] **by** *fastforce*
have *3*: ⊢ *f oproj* (○ *g*) = (*f* ∧ *more* ∧ *finite*);(*f oproj g*)
by (*simp add*: *OProjNext*)

show *?thesis*
using *1 2 3* **by** *fastforce*
qed

lemma *FProjIntro*:
assumes $\vdash$ f $\longrightarrow$ $((g \wedge$ *more*$) \wedge$ *finite*$)$; f
shows $\vdash f \wedge$ *finite*$\longrightarrow$ g *fproj* #*True*
using *assms SCSIntro*[*of f g*] *FProjTrueEqvSChopstar*[*of g*] **unfolding** *schop-d-def*
by *fastforce*

lemma *OProjIntro*:
assumes $\vdash$ f $\longrightarrow$ $((g \wedge$ *more*$) \wedge$ *finite*$)$; f
shows $\vdash f$ $\longrightarrow$ g *oproj* #*True*
using *assms OProjTrueEqvOmega*[*of g*] **by** (*metis OmegaWeakCoinduct int-eq*)

lemma *RightBoxStateImportFProj*:
$\vdash \Box($*init w*$) \wedge f$ *fproj* $g \longrightarrow f$ *fproj* $(\Box($*init w*$) \wedge g)$
by (*simp add*: *Valid-def always-defs init-defs fprojection-d-def*)
(*metis ndropn-all ndropn-nfirst nfinite-conv-nlength-enat nle-le pfilt-code*(*1*) *pfilt-nmap-pfilt*)

lemma *RightBoxStateImportOProj*:
$\vdash \Box($*init w*$) \wedge f$ *oproj* $g \longrightarrow f$ *oproj* $(\Box($*init w*$) \wedge g)$
by (*simp add*: *Valid-def always-defs init-defs oprojection-d-def*)
(*metis min-def ndropn-nfirst nfinite-conv-nlength-enat nfinite-ntaken ntaken-nlength pfilt-nnth*)

lemma *LeftBoxStateImportFProjhelp*:
$(\forall n.\ enat\ n \le nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa)))) \wedge$
$(\exists ls.\ (\forall i.\ enat\ (Suc\ i) \le nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
$nnth\ ls\ 0 = 0 \wedge$
$nfinite\ ls \wedge$
$nfinite\ wa \wedge$
$nlast\ ls = the\text{-}enat\ (nlength\ wa) \wedge$
$(\forall i.\ enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls)) \longrightarrow$
$(\exists ls.\ (\forall i.\ enat\ (Suc\ i) \le nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
$nnth\ ls\ 0 = 0 \wedge$
$nfinite\ ls \wedge$
$nfinite\ wa \wedge$
$nlast\ ls = the\text{-}enat\ (nlength\ wa) \wedge$
$(\forall i.\ enat\ i < nlength\ ls \longrightarrow$
$f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
$(\forall n.\ enat\ n \le nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
$w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))))) \wedge$
$g\ (pfilt\ wa\ ls))$
proof
assume *0*: $(\forall n.\ enat\ n \le nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa)))) \wedge$
$(\exists ls.\ (\forall i.\ enat\ (Suc\ i) \le nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
$nnth\ ls\ 0 = 0 \wedge$
$nfinite\ ls \wedge$
$nfinite\ wa \wedge$

nlast ls = *the-enat* (*nlength wa*) ∧
(∀ *i*. *enat i* < *nlength ls* ⟶ *f* (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*)))) ∧ *g* (*pfilt wa ls*))

show (∃ *ls*. (∀ *i*. *enat* (*Suc i*) ≤ *nlength ls* ⟶ *nnth ls i* < *nnth ls* (*Suc i*)) ∧
nnth ls 0 = *0* ∧
nfinite ls ∧
nfinite wa ∧
nlast ls = *the-enat* (*nlength wa*) ∧
(∀ *i*. *enat i* < *nlength ls* ⟶
f (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*))) ∧
(∀ *n*. *enat n* ≤ *nlength* (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*))) ⟶
w (*NNil* (*nfirst* (*ndropn n* (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*)))))))) ∧
g (*pfilt wa ls*))

proof −

have *1*: (∀ *n*. *enat n* ≤ *nlength wa* ⟶ *w* (*NNil* (*nfirst* (*ndropn n wa*))))
using *0* **by** *auto*

have *2*: (∃ *ls*. (∀ *i*. *enat* (*Suc i*) ≤ *nlength ls* ⟶ *nnth ls i* < *nnth ls* (*Suc i*)) ∧
nnth ls 0 = *0* ∧
nfinite ls ∧
nfinite wa ∧
nlast ls = *the-enat* (*nlength wa*) ∧
(∀ *i*. *enat i* < *nlength ls* ⟶
f (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*)))) ∧ *g* (*pfilt wa ls*))
using *0* **by** *auto*

obtain *ls* **where** *3*: (∀ *i*. *enat* (*Suc i*) ≤ *nlength ls* ⟶ *nnth ls i* < *nnth ls* (*Suc i*)) ∧
nnth ls 0 = *0* ∧
nfinite ls ∧
nfinite wa ∧
nlast ls = *the-enat* (*nlength wa*) ∧
(∀ *i*. *enat i* < *nlength ls* ⟶
f (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*)))) ∧ *g* (*pfilt wa ls*)
using *2* **by** *auto*

have *4*: *nnth ls 0* = *0*
using *3* **by** *auto*

have *5*: (∀ *i*. *enat* (*Suc i*) ≤ *nlength ls* ⟶ *nnth ls i* < *nnth ls* (*Suc i*))
using *3* **by** *auto*

have *6*: *nlast ls* = *the-enat*(*nlength wa*)
using *3* **by** *auto*

have *7*: (∀ *i*<*nlength ls*. *f* (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*))))
using *3* **by** *auto*

have *8*: *g* (*pfilt wa ls*)
using *3* **by** *auto*

have *9*: (∀ *i*<*nlength ls*.
f (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*))) ∧
(∀ *n*≤(*nnth ls* (*Suc i*)) −(*nnth ls i*).
w (*NNil* (*nnth wa* ((*nnth ls i*)+*n*)))))
using *1 7*
by (*metis 0 ndropn-eq-NNil ndropn-nfirst ndropn-nlast nfinite-conv-nlength-enat nnth-beyond not-le-imp-less the-enat.simps*)

have *10*: (∀ *i*<*nlength ls*.
nlength (*nsubn wa* (*nnth ls i*) (*nnth ls* (*Suc i*))) =

$(nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i)\ )$
by (*simp add*: *3 PJ6help1 Suc-ile-eq nidx-expand*)
have *11*: $(\forall\, i<nlength\ ls.$
$(\forall\, n\le(nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i).$
$nnth\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )\ n =$
$nnth\ wa\ ((nnth\ ls\ i)+n)\ ))$
using *3* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *12*: $(\forall\, i<nlength\ ls.$
$f\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )\ \wedge$
$(\forall\, n\le nlength\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ).$
$w\ (NNil\ (nnth\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i)))\ n))))$
using *9 10 11* **by** *simp*
have *13*: $(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow$
$f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ \wedge$
$(\forall\, n.\ enat\ n \le nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
$w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))))$
by (*simp add*: *12 ndropn-nfirst*)
show *?thesis*
using *13 3* **by** *blast*
qed
qed

lemma *LeftBoxStateImportOProjhelp*:
$(\forall\, n.\ enat\ n \le nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa))))\ \wedge$
$(\exists\, ls.\ (\forall\, i.\ enat\ (Suc\ i) \le nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i))\ \wedge$
$nnth\ ls\ 0 = 0\ \wedge$
$\neg\ nfinite\ ls\ \wedge\ \neg\ nfinite\ wa\ \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ \wedge\ g\ (pfilt\ wa\ ls)) \longrightarrow$
$(\exists\, ls.\ (\forall\, i.\ enat\ (Suc\ i) \le nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i))\ \wedge$
$nnth\ ls\ 0 = 0\ \wedge$
$\neg\ nfinite\ ls\ \wedge$
$\neg\ nfinite\ wa\ \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow$
$f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ \wedge$
$(\forall\, n.\ enat\ n \le nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
$w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))))\ \wedge$
$g\ (pfilt\ wa\ ls))$
proof
assume *0*: $(\forall\, n.\ enat\ n \le nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa))))\ \wedge$
$(\exists\, ls.\ (\forall\, i.\ enat\ (Suc\ i) \le nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i))\ \wedge$
$nnth\ ls\ 0 = 0\ \wedge$
$\neg\ nfinite\ ls\ \wedge\ \neg\ nfinite\ wa\ \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ \wedge\ g\ (pfilt\ wa\ ls))$
show $(\exists\, ls.\ (\forall\, i.\ enat\ (Suc\ i) \le nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i))\ \wedge$
$nnth\ ls\ 0 = 0\ \wedge$
$\neg\ nfinite\ ls\ \wedge$
$\neg\ nfinite\ wa\ \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow$
$f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ \wedge$
$(\forall\, n.\ enat\ n \le nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$

$w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))))) \wedge$
$g\ (pfilt\ wa\ ls))$

proof −
have *1*: $(\forall\, n.\ enat\ n \leq nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa))))$
using *0* **by** *auto*
have *2*: $(\exists\, ls.\ (\forall\, i.\ enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
$nnth\ ls\ 0 = 0 \wedge$
$\neg\ nfinite\ ls \wedge \neg\ nfinite\ wa \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls))$
using *0* **by** *auto*
obtain *ls* **where** *3*: $(\forall\, i.\ enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
$nnth\ ls\ 0 = 0 \wedge$
$\neg\ nfinite\ ls \wedge \neg\ nfinite\ wa \wedge$
$(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls)$
using *2* **by** *auto*
have *4*: $nnth\ ls\ 0 = 0$
using *3* **by** *auto*
have *5*: $(\forall\, i.\ enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i))$
using *3* **by** *auto*
have *7*: $(\forall\, i{<}nlength\ ls.\ f\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ))$
using *3* **by** *auto*
have *8*: $g\ (pfilt\ wa\ ls)$
using *3* **by** *auto*
have *9*: $(\forall\, i{<}nlength\ ls.$
$f\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))) \wedge$
$(\forall\, n{\leq}(nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i).$
$w\ (NNil\ (nnth\ wa\ ((nnth\ ls\ i){+}n)))))$
using *1 7*
by (*metis 0 linorder-le-cases ndropn-eq-NNil ndropn-nfirst nfinite-NNil nfinite-ndropn-b*)
have *10*: $(\forall\, i{<}nlength\ ls.$
$nlength\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ) =$
$(nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i)\ )$
by (*simp add*: *3 OPJ6help1 nidx-expand*)
have *11*: $(\forall\, i{<}nlength\ ls.$
$(\forall\, n{\leq}(nnth\ ls\ (Suc\ i))\ -(nnth\ ls\ i).$
$nnth\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ )\ n =$
$nnth\ wa\ ((nnth\ ls\ i){+}n)\ ))$
using *3* **by** (*simp add*: *nsubn-def1 ntaken-nnth*)
have *12*: $(\forall\, i{<}nlength\ ls.$
$f\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ) \wedge$
$(\forall\, n{\leq}nlength\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i))\ ).$
$w\ (NNil\ (nnth\ (nsubn\ wa\ (\ nnth\ ls\ i)\ (\ nnth\ ls\ (Suc\ i)))\ n))))$
using *9 10 11* **by** *simp*
have *13*: $(\forall\, i.\ enat\ i < nlength\ ls \longrightarrow$
$f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
$(\forall\, n.\ enat\ n \leq nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
$w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))))$
by (*simp add*: *12 ndropn-nfirst*)
show *?thesis*
using *13 3* **by** *blast*

qed
qed

lemma *LeftBoxStateImportFProj*:
⊢ □(*init w*) ∧ *f fproj g* ⟶ (*f* ∧ □ (*init w*)) *fproj g*
using *LeftBoxStateImportFProjhelp*[*of - w f g*]
by (*simp add*: *Valid-def always-defs init-defs fprojection-d-def powerinterval-def nidx-expand*)

lemma *LeftBoxStateImportOProj*:
⊢ □(*init w*) ∧ *f oproj g* ⟶ (*f* ∧ □ (*init w*)) *oproj g*
using *LeftBoxStateImportOProjhelp*[*of - w f g*]
by (*simp add*: *Valid-def always-defs init-defs oprojection-d-def powerinterval-def nidx-expand*)

10.5.2 fdp, fbp odp and obp

lemma *NotFDpEqvFBpNot*:
⊢ (¬(*fdp f*)) = *fbp* (¬ *f*)
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)

lemma *NotODpEqvOBpNot*:
⊢ (¬(*odp f*)) = *obp* (¬ *f*)
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)

lemma *NotFDBpEqvFDpNot*:
⊢ (¬(*fbp f*)) = *fdp*(¬ *f*)
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)

lemma *NotODBpEqvODpNot*:
⊢ (¬(*obp f*)) = *odp*(¬ *f*)
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)

lemma *NowImpFDp*:
⊢ *f* ∧ *finite*⟶ *fdp f*
proof −
 have *1*: ⊢ (*skip* ⟶ #*True*)
  by *simp*
 have *2*: ⊢ *ba*(*skip* ⟶ #*True*)
  using *1* **by** (*simp add*: *BaGen*)
 have *3*: ⊢ *ba*(*skip* ⟶ #*True*) ⟶ (*skip fproj f* ⟶ #*True fproj f*)
  using *PJ8* **by** *blast*
 have *4*: ⊢ (*skip fproj f* ⟶ #*True fproj f*)
  using *2 3 MP* **by** *blast*
 show *?thesis* **using** *4 PJ6*
 by (*metis 4 PJ6 fdp-d-def inteq-reflection*)
qed

lemma *NowImpODp*:
⊢ *f* ∧ *inf* ⟶ *odp f*
proof −
 have *1*: ⊢ (*skip* ⟶ #*True*)

by *simp*
have *2*: ⊢ *ba*(*skip* ⟶ *#True*)
using *1* **by** (*simp add*: *BaGen*)
have *3*: ⊢ *ba*(*skip* ⟶ *#True*) ⟶ (*skip oproj f* ⟶ *#True oproj f*)
using *OPJ8* **by** *blast*
have *4*: ⊢ (*skip oproj f* ⟶ *#True oproj f*)
using *2 3 MP* **by** *blast*
show *?thesis* **using** *4 OPJ6* **by** (*metis int-eq odp-d-def*)
qed

lemma *FBpElim*:
⊢ *fbp f* ∧ *finite*⟶ *f*
proof −
have *1*: ⊢ ¬ *f* ∧ *finite*⟶ *fdp* (¬ *f*)
by (*simp add*: *NowImpFDp*)
hence *2*: ⊢ ¬(*fdp* (¬ *f*)) ⟶ *f* ∨ *inf*
unfolding *finite-d-def* **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *Prop13 fbp-d-def fdp-d-def finite-d-def ufprojection-d-def*)
qed

lemma *OBpElim*:
⊢ *obp f* ∧ *inf* ⟶ *f*
proof −
have *1*: ⊢ ¬ *f* ∧ *inf* ⟶ *odp* (¬ *f*)
by (*simp add*: *NowImpODp*)
hence *2*: ⊢ ¬(*odp* (¬ *f*)) ⟶ *f* ∨ *finite*
unfolding *finite-d-def* **by** *auto*
from *2* **show** *?thesis*
by (*metis InfEqvNotFinite Prop13 inteq-reflection obp-d-def odp-d-def uoprojection-d-def*)
qed

lemma *FBpImpFDpImpFDp*:
⊢ *fbp* (*f* ⟶ *g*) ⟶ *fdp f* ⟶ *fdp g*
proof −
have *1*: ⊢ *fbp* (*f*⟶ *g*) ⟶ (*#True fproj f*) ⟶ (*#True fproj g*)
by (*simp add*: *PJ9 fbp-d-def*)
from *1* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *OBpImpODpImpODp*:
⊢ *obp* (*f* ⟶ *g*) ⟶ *odp f* ⟶ *odp g*
proof −
have *1*: ⊢ *obp* (*f*⟶ *g*) ⟶ (*#True oproj f*) ⟶ (*#True oproj g*)
by (*simp add*: *OPJ9 obp-d-def*)
from *1* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpContraPosImpDist*:

$\vdash$ *fbp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ (*fbp* *f*) $\longrightarrow$ (*fbp* *g*)
proof –
have *1*: $\vdash$ *fbp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ (*fdp* ($\neg$ *g*)) $\longrightarrow$ (*fdp* ($\neg$ *f*))
by (*rule FBpImpFDpImpFDp*)
hence *2*: $\vdash$ *fbp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ ($\neg$ (*fdp* ($\neg$ *f*))) $\longrightarrow$ ($\neg$ (*fdp* ($\neg$ *g*))) **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)
qed

lemma *OBpContraPosImpDist*:
$\vdash$ *obp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ (*obp* *f*) $\longrightarrow$ (*obp* *g*)
proof –
have *1*: $\vdash$ *obp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ (*odp* ($\neg$ *g*)) $\longrightarrow$ (*odp* ($\neg$ *f*))
by (*rule OBpImpODpImpODp*)
hence *2*: $\vdash$ *obp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ ($\neg$ (*odp* ($\neg$ *f*))) $\longrightarrow$ ($\neg$ (*odp* ($\neg$ *g*))) **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)
qed

lemma *FBpImpDist*:
$\vdash$ *fbp* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*fbp* *f*) $\longrightarrow$ (*fbp* *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*: $\vdash$ *fbp* ($\neg$ ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ $\neg$ (*f* $\longrightarrow$ *g*)) **by** (*rule FBpGen*)
have *4*: $\vdash$ *fbp* ($\neg$ ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ $\neg$ (*f* $\longrightarrow$ *g*))
$\longrightarrow$
fbp (*f* $\longrightarrow$ *g*) $\longrightarrow$ *fbp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) **by** (*rule FBpContraPosImpDist*)
have *5*: $\vdash$ *fbp* (*f* $\longrightarrow$ *g*) $\longrightarrow$ *fbp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) **using** *3 4 MP* **by** *blast*
have *6*: $\vdash$ *fbp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ (*fbp* *f*) $\longrightarrow$ (*fbp* *g*) **by** (*rule FBpContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *OBpImpDist*:
$\vdash$ *obp* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*obp* *f*) $\longrightarrow$ (*obp* *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*: $\vdash$ *obp* ($\neg$ ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ $\neg$ (*f* $\longrightarrow$ *g*)) **by** (*rule OBpGen*)
have *4*: $\vdash$ *obp* ($\neg$ ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ $\neg$ (*f* $\longrightarrow$ *g*))
$\longrightarrow$
obp (*f* $\longrightarrow$ *g*) $\longrightarrow$ *obp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) **by** (*rule OBpContraPosImpDist*)
have *5*: $\vdash$ *obp* (*f* $\longrightarrow$ *g*) $\longrightarrow$ *obp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) **using** *3 4 MP* **by** *blast*
have *6*: $\vdash$ *obp* ($\neg$ *g* $\longrightarrow$ $\neg$ *f*) $\longrightarrow$ (*obp* *f*) $\longrightarrow$ (*obp* *g*) **by** (*rule OBpContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FDpImpDpRule*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ *fdp* *f* $\longrightarrow$ *fdp* *g*

proof –
 have *1*: ⊢ $f \longrightarrow g$ **using** *assms* **by** *auto*
 hence *2*: ⊢ *#True fproj f* ⟶ *#True fproj g*
  by (*metis FBpGen MP PJ9 fbp-d-def*)
 from *2* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpImpODpRule*:
 assumes ⊢ $f \longrightarrow g$
 shows ⊢ *odp f* ⟶ *odp g*
proof –
 have *1*: ⊢ $f \longrightarrow g$ **using** *assms* **by** *auto*
 hence *2*: ⊢ *#True oproj f* ⟶ *#True oproj g*
  by (*metis OBpGen MP OPJ9 obp-d-def*)
 from *2* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpImpFBpRule*:
 assumes ⊢ $f \longrightarrow g$
 shows ⊢ *fbp f* ⟶ *fbp g*
proof –
 have *1*: ⊢ $f \longrightarrow g$ **using** *assms* **by** *auto*
 hence *2*: ⊢ ¬ *g* ⟶ ¬ *f* **by** *auto*
 hence *3*: ⊢ *fdp* (¬ *g*) ⟶ *fdp* (¬ *f*) **by** (*rule FDpImpDpRule*)
 hence *4*: ⊢ ¬ (*fdp* (¬ *f*)) ⟶ ¬ (*fdp* (¬ *g*)) **by** *auto*
 from *4* **show** *?thesis*
  by (*meson FBpGen FBpImpDist MP assms*)
qed

lemma *OBpImpOBpRule*:
 assumes ⊢ $f \longrightarrow g$
 shows ⊢ *obp f* ⟶ *obp g*
proof –
 have *1*: ⊢ $f \longrightarrow g$ **using** *assms* **by** *auto*
 hence *2*: ⊢ ¬ *g* ⟶ ¬ *f* **by** *auto*
 hence *3*: ⊢ *odp* (¬ *g*) ⟶ *odp* (¬ *f*) **by** (*rule ODpImpODpRule*)
 hence *4*: ⊢ ¬ (*odp* (¬ *f*)) ⟶ ¬ (*odp* (¬ *g*)) **by** *auto*
 from *4* **show** *?thesis*
  by (*meson OBpGen OBpImpDist MP assms*)
qed

lemma *FDpEqvFDpRule*:
 assumes ⊢ $f = g$
 shows ⊢ *fdp f* = *fdp g*
proof –
 have *1*: ⊢ $f = g$ **using** *assms* **by** *auto*
 hence *2*: ⊢ *#True fproj f* = *#True fproj g*
  using *RightFProjEqvFProj* **by** *blast*
 from *2* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpEqvODpRule*:
assumes $\vdash f = g$
shows $\vdash$ *odp* $f =$ *odp* g
proof –
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ #*True oproj* $f =$ #*True oproj* g
using *RightOProjEqvOProj* **by** *blast*
from *2* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpEqvFBpRule*:
assumes $\vdash f = g$
shows $\vdash$ *fbp* $f =$ *fbp* g
proof –
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ f) = (\neg\ g)$ **by** *auto*
hence *3*: $\vdash$ *fdp* $(\neg\ f) =$ *fdp* $(\neg\ g)$ **by** (*rule FDpEqvFDpRule*)
hence *4*: $\vdash (\neg\ ($*fdp* $(\neg\ f))) = (\neg\ ($ *fdp* $(\neg\ g)))$ **by** *auto*
from *4* **show** *?thesis*
by (*metis FBpImpFBpRule assms int-iffD1 int-iffI inteq-reflection*)
qed

lemma *OBpEqvOBpRule*:
assumes $\vdash f = g$
shows $\vdash$ *obp* $f =$ *obp* g
proof –
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg\ f) = (\neg\ g)$ **by** *auto*
hence *3*: $\vdash$ *odp* $(\neg\ f) =$ *odp* $(\neg\ g)$ **by** (*rule ODpEqvODpRule*)
hence *4*: $\vdash (\neg\ ($*odp* $(\neg\ f))) = (\neg\ ($ *odp* $(\neg\ g)))$ **by** *auto*
from *4* **show** *?thesis*
by (*metis OBpImpOBpRule assms int-iffD1 int-iffI inteq-reflection*)
qed

lemma *FDpState*:
$\vdash$ *fdp* (*init w*) = ((*init w*) $\wedge$ *finite*)
proof –
have *1*: $\vdash$ *init w* $\wedge$ *finite* $\longrightarrow$ *fdp* (*init w*)
using *NowImpFDp*[*of LIFT* (*init w*)] **by** *blast*
have *2*: $\vdash$*fdp* (*init w*) $\longrightarrow$ *init w*
unfolding *fdp-d-def* **by** (*simp add*: *PJ5*)
have *3*: $\vdash$*fdp* (*init w*) $\longrightarrow$ *finite*
unfolding *fdp-d-def* **by** (*simp add*: *PJ01*)
show *?thesis*
by (*simp add*: *1 2 3 Prop12 int-iffI*)
qed

lemma *ODpState*:
$\vdash$ *odp* (*init w*) = ((*init w*) $\wedge$ *inf*)

proof −
have *1*: ⊢ *init w* ∧ *inf* ⟶ *odp* (*init w*)
using *NowImpODp*[*of LIFT* (*init w*)] **by** *blast*
have *2*: ⊢*odp* (*init w*) ⟶ *init w*
unfolding *odp-d-def* **by** (*simp add*: *OPJ5*)
have *3*: ⊢*odp* (*init w*) ⟶ *inf*
unfolding *odp-d-def* **by** (*simp add*: *OPJ01*)
show *?thesis*
by (*simp add*: *1 2 3 Prop12 int-iffI*)
qed

lemma *StateEqvFBp*:
⊢ *finite* ⟶ (*init w*) = *fbp* (*init w*)
proof −
have *1*: ⊢ (*init w*) ⟶ *fbp* (*init w*)
by (*metis* (*no-types*, *lifting*) *DiEqvNotBiNot DiState Initprop*(*2*) *NotDiEqvBiNot PJ5 fbp-d-def inteq-reflection lift-imp-neg ufprojection-d-def*)
have *2*: ⊢ *fbp* (*init w*) ∧ *finite* ⟶ (*init w*) **using** *FBpElim* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *StateEqvOBp*:
⊢ *inf* ⟶ (*init w*) = *obp* (*init w*)
proof −
have *1*: ⊢ (*init w*) ⟶ *obp* (*init w*)
by (*metis* (*no-types*, *lifting*) *DiEqvNotBiNot DiState Initprop*(*2*) *NotDiEqvBiNot NotODpEqvOBpNot ODpState Prop11 Prop12 inteq-reflection lift-imp-neg*)
have *2*: ⊢ *obp* (*init w*) ∧ *inf* ⟶ (*init w*) **using** *OBpElim* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FDpFDpEqvFDp*:
⊢ *fdp* (*fdp f*) = *fdp f*
proof −
have *2*: ⊢ #*True fproj* (#*True fproj f*) = (#*True fproj* #*True*) *fproj f*
by (*simp add*: *PJ7*)
have *3*: ⊢ (#*True fproj* #*True*) = *finite*
by (*metis ChopEmpty EmptyImpFinite NowImpFDp PJ01 Prop10 TrueChopAndFiniteEqvAndFinite-ChopFinite*
fdp-d-def int-eq int-iffI)
have *4*: ⊢ *finite fproj f* = #*True fproj f*
by (*metis PJ02 Prop03 Prop10 finite-d-def int-simps*(*28*) *inteq-reflection lift-and-com*)
show *?thesis*
by (*metis 2 3 4 fdp-d-def int-eq*)
qed

lemma *ODpODpEqvODp*:
⊢ *odp* (*odp f*) = *odp f*
proof −
have *2*: ⊢ #*True oproj* (#*True oproj f*) = (#*True fproj* #*True*) *oproj f*

by (*simp add*: *OPJ7*)
have *3*: ⊢ (#*True* *fproj* #*True*) = *finite*
by (*metis ChopEmpty EmptyImpFinite NowImpFDp PJ01 Prop10 TrueChopAndFiniteEqvAndFinite-ChopFinite*
fdp-d-def int-eq int-iffI)
have *4*: ⊢ *finite oproj f* = #*True oproj f*
by (*metis OPJ02 Prop03 Prop10 finite-d-def int-simps*(*28*) *inteq-reflection lift-and-com*)
show *?thesis*
by (*metis 2 3 4 odp-d-def int-eq*)
qed

lemma *FBpFBpEqvFBp*:
⊢ *fbp* (*fbp f*) = *fbp f*
proof –
have *1*: ⊢ *fdp* (*fdp* (¬ *f*)) = *fdp* (¬ *f*)
using *FDpFDpEqvFDp* **by** *blast*
have *2*: ⊢ (¬ (*fdp* (*fdp* (¬ *f*)))) = (¬ (*fdp* (¬ *f*)))
using *1* **by** *auto*
have *3*: ⊢ (¬ (*fdp* (¬ *f*))) = *fbp f*
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)
have *4*: ⊢ (¬ (*fdp* (*fdp* (¬ *f*)))) = *fbp* (*fbp f*)
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)
from *2 3 4* **show** *?thesis*
by *fastforce*
qed

lemma *OBpOBpEqvOBp*:
⊢ *obp* (*obp f*) = *obp f*
proof –
have *1*: ⊢ *odp* (*odp* (¬ *f*)) = *odp* (¬ *f*)
using *ODpODpEqvODp* **by** *blast*
have *2*: ⊢ (¬ (*odp* (*odp* (¬ *f*)))) = (¬ (*odp* (¬ *f*)))
using *1* **by** *auto*
have *3*: ⊢ (¬ (*odp* (¬ *f*))) = *obp f*
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)
have *4*: ⊢ (¬ (*odp* (*odp* (¬ *f*)))) = *obp* (*obp f*)
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)
from *2 3 4* **show** *?thesis*
by *fastforce*
qed

lemma *FDpOrEqv*:
⊢ *fdp* (*f* ∨ *g*) = (*fdp f* ∨ *fdp g*)
proof –
have *1*: ⊢ #*True* *fproj* (*f*∨ *g*) = (#*True* *fproj* *f* ∨ #*True* *fproj* *g*)
using *FProjOrDist* **by** *auto*
from *1* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpOrEqv*:
$\vdash$ *odp* $(f \vee g) =$ $($*odp* $f \vee$ *odp* $g)$
proof –
have *1*: $\vdash$ *#True oproj* $(f \vee g) = ($ *#True oproj* $f \vee$ *#True oproj* $g)$
using *OProjOrDist* **by** *auto*
from *1* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpAndEqv*:
$\vdash$ *fbp*$(f \wedge g) = ($*fbp* $f \wedge$ *fbp* $g)$
proof –
have *1*: $\vdash$ *fdp* $((\neg f) \vee (\neg g)) =$ $($*fdp* $(\neg f) \vee$ *fdp* $(\neg g))$
using *FDpOrEqv* **by** *auto*
hence *2*: $\vdash (\neg$ $($*fdp* $((\neg f) \vee (\neg g)))) =$ $(\neg($*fdp* $(\neg f) \vee$ *fdp* $(\neg g)))$
by *auto*
have *3*: $\vdash (\neg$ $($*fdp* $((\neg f) \vee (\neg g)))) =$ *fbp* $(\neg((\neg f) \vee (\neg g)))$
using *NotFDpEqvFBpNot* **by** *blast*
have *4*: $\vdash (\neg((\neg f) \vee (\neg g))) =(f \wedge g)$
by *auto*
hence *5*: $\vdash$ *fbp*$(\neg((\neg f) \vee (\neg g))) =$ *fbp*$(f \wedge g)$
by (*simp add*: *FBpEqvFBpRule*)
have *6*: $\vdash (\neg($*fdp* $(\neg f) \vee$ *fdp* $(\neg g))) = ((\neg($*fdp* $(\neg f))) \wedge (\neg($*fdp* $(\neg g))))$
by *auto*
have *7*: $\vdash ((\neg($*fdp* $(\neg f))) \wedge (\neg($*fdp* $(\neg g)))) = ($*fbp* $f \wedge$ *fbp* $g)$
by (*simp add*: *fbp-d-def fdp-d-def ufprojection-d-def*)
show *?thesis*
by (*metis 2 3 4 6 7 inteq-reflection*)
qed

lemma *OBpAndEqv*:
$\vdash$ *obp*$(f \wedge g) = ($*obp* $f \wedge$ *obp* $g)$
proof –
have *1*: $\vdash$ *odp* $((\neg f) \vee (\neg g)) =$ $($*odp* $(\neg f) \vee$ *odp* $(\neg g))$
using *ODpOrEqv* **by** *auto*
hence *2*: $\vdash (\neg$ $($*odp* $((\neg f) \vee (\neg g)))) =$ $(\neg($*odp* $(\neg f) \vee$ *odp* $(\neg g)))$
by *auto*
have *3*: $\vdash (\neg$ $($*odp* $((\neg f) \vee (\neg g)))) =$ *obp* $(\neg((\neg f) \vee (\neg g)))$
using *NotODpEqvOBpNot* **by** *blast*
have *4*: $\vdash (\neg((\neg f) \vee (\neg g))) =(f \wedge g)$
by *auto*
hence *5*: $\vdash$ *obp*$(\neg((\neg f) \vee (\neg g))) =$ *obp*$(f \wedge g)$
by (*simp add*: *OBpEqvOBpRule*)
have *6*: $\vdash (\neg($*odp* $(\neg f) \vee$ *odp* $(\neg g))) = ((\neg($*odp* $(\neg f))) \wedge (\neg($*odp* $(\neg g))))$
by *auto*
have *7*: $\vdash ((\neg($*odp* $(\neg f))) \wedge (\neg($*odp* $(\neg g)))) = ($*obp* $f \wedge$ *obp* $g)$
by (*simp add*: *obp-d-def odp-d-def uoprojection-d-def*)
show *?thesis*
by (*metis 2 3 4 6 7 inteq-reflection*)
qed

lemma *FDpAndA*:
⊢ *fdp* (*f* ∧ *g*) ⟶ *fdp* *f*
proof –
have *1*: ⊢ #*True* *fproj* (*f* ∧ *g*) ⟶ #*True* *fproj* *f*
by (*meson Prop12 RightFProjImpFProj int-iffD1 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpAndA*:
⊢ *odp* (*f* ∧ *g*) ⟶ *odp* *f*
proof –
have *1*: ⊢ #*True oproj* (*f* ∧ *g*) ⟶ #*True oproj f*
by (*meson Prop12 RightOProjImpOProj int-iffD1 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FBpOrA*:
⊢ *fbp f* ⟶ *fbp*(*f* ∨ *g*)
by (*simp add*: *FBpImpFBpRule intI*)

lemma *OBpOrA*:
⊢ *obp f* ⟶ *obp*(*f* ∨ *g*)
by (*simp add*: *OBpImpOBpRule intI*)

lemma *FBpOrB*:
⊢ *fbp g* ⟶ *fbp*(*f* ∨ *g*)
by (*simp add*: *FBpImpFBpRule intI*)

lemma *OBpOrB*:
⊢ *obp g* ⟶ *obp*(*f* ∨ *g*)
by (*simp add*: *OBpImpOBpRule intI*)

lemma *FBpOrImpOr*:
⊢ *fbp f* ∨ *fbp g* ⟶ *fbp*(*f* ∨ *g*)
using *FBpOrA FBpOrB* **by** *fastforce*

lemma *OBpOrImpOr*:
⊢ *obp f* ∨ *obp g* ⟶ *obp*(*f* ∨ *g*)
using *OBpOrA OBpOrB* **by** *fastforce*

lemma *FDpAndB*:
⊢ *fdp* (*f* ∧ *g*) ⟶ *fdp* *g*
proof –
have *1*: ⊢ #*True* *fproj* (*f* ∧ *g*) ⟶ #*True* *fproj* *g*
by (*meson Prop12 RightFProjImpFProj int-iffD2 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *fdp-d-def*)
qed

lemma *ODpAndB*:
⊢ *odp* (*f* ∧ *g*) ⟶ *odp* *g*

proof −
have *1*: ⊢ *#True oproj* (*f* ∧ *g*) ⟶ *#True oproj g*
by (*meson Prop12 RightOProjImpOProj int-iffD2 lift-and-com*)
from *1* **show** *?thesis* **by** (*simp add*: *odp-d-def*)
qed

lemma *FDpAndImpAnd*:
⊢ *fdp* (*f* ∧ *g*) ⟶ *fdp f* ∧ *fdp g*
proof −
have *1*: ⊢ *fdp* (*f* ∧ *g*) ⟶ *fdp f* **by** (*rule FDpAndA*)
have *2*: ⊢ *fdp* (*f* ∧ *g*) ⟶ *fdp g* **by** (*rule FDpAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *ODpAndImpAnd*:
⊢ *odp* (*f* ∧ *g*) ⟶ *odp f* ∧ *odp g*
proof −
have *1*: ⊢ *odp* (*f* ∧ *g*) ⟶ *odp f* **by** (*rule ODpAndA*)
have *2*: ⊢ *odp* (*f* ∧ *g*) ⟶ *odp g* **by** (*rule ODpAndB*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FDpSkipEqvMore*:
⊢ *fdp skip* = (*more* ∧ *finite*)
proof −
have *1*: ⊢ *fdp skip* = *#True fproj skip*
by (*simp add*: *fdp-d-def*)
have *2*: ⊢ *#True fproj skip* = (*#True* ∧ *more* ∧ *finite*)
using *PJ3* **by** *blast*
have *3*: ⊢ (*#True* ∧ *more*) = *more*
by *auto*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FDpMoreEqvMore*:
⊢ *fdp more* = (*more* ∧ *finite*)
using *FDpFDpEqvFDp FDpSkipEqvMore*
by (*metis PJ03 fdp-d-def inteq-reflection*)

lemma *ODpMoreEqvInf*:
⊢ *odp more* = (*inf*)
by (*metis MoreAndInfEqvInf NowImpODp OPJ01 int-iffI inteq-reflection odp-d-def*)

lemma *FBpEmptyEqvEmpty*:
⊢ *fbp empty* = (*empty* ∨ *inf*)
proof −
have *1*: ⊢ *fbp empty* = (¬ (*fdp more*))
by (*metis NotFDpEqvFBpNot Prop11 empty-d-def*)
have *2*: ⊢ (¬ (*fdp more*)) = (¬ (*more* ∧ *finite*))

using *FDpMoreEqvMore* **by** *auto*
have *3*: ⊢ (¬ (*more* ∧ *finite*)) = (*empty* ∨ *inf*)
unfolding *finite-d-def empty-d-def* **by** *fastforce*
show *?thesis*
by (*metis 1 2 3 int-eq*)
qed

lemma *OBpEmptyEqvFinite*:
⊢ *obp empty* = (*finite*)
proof −
have *1*: ⊢ *obp empty* = (¬ (*odp more*))
by (*metis NotODpEqvOBpNot Prop11 empty-d-def*)
have *2*: ⊢ (¬ (*odp more*)) = (¬ (*inf*))
using *ODpMoreEqvInf* **by** *auto*
have *3*: ⊢ (¬ (*inf*)) = (*finite*)
unfolding *finite-d-def* **by** *fastforce*
show *?thesis*
by (*metis 1 2 3 int-eq*)
qed

lemma *FDpEmptyEqvEmpty*:
⊢ *fdp empty* = *empty*
proof −
have *1*: ⊢ *fdp empty* = #*True fproj empty*
by (*simp add*: *fdp-d-def*)
have *2*: ⊢ #*True fproj empty* = *empty*
by (*simp add*: *PJ2*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotODpEmpty*:
⊢ ¬(*odp empty*)
proof −
have *1*: ⊢ *odp empty* = #*True oproj empty*
by (*simp add*: *odp-d-def*)
show *?thesis*
by (*metis 1 OPJ2 int-eq*)
qed

lemma *FBpMoreEqvMore*:
⊢ *fbp more* = *more*
by (*metis NotFDBpEqvFDpNot PJ2 empty-d-def fdp-d-def int-eq int-simps*(*4*))

lemma *OBpMore*:
⊢ *obp more*
by (*metis OPJ2 empty-d-def obp-d-def uoprojection-d-def*)

lemma *NextFDpImpFDpNext*:

$\vdash\ \bigcirc\ (fdp\ f) \longrightarrow fdp\ (\bigcirc\ f)$
proof –
have *1*: $\vdash fdp(\bigcirc\ f) = \#True\ \ fproj\ \ (skip;f)$
by (*simp add*: *fdp-d-def next-d-def*)
have *2*: $\vdash \#True\ \ fproj\ \ (skip;f) = (\#True\ \ fproj\ \ skip);(\#True\ \ fproj\ \ f)$
by (*simp add*: *PJ4*)
have *3*: $\vdash (\#True\ \ fproj\ \ skip) = (\#True \wedge more \wedge finite)$
using *PJ3* **by** *blast*
have *4*: $\vdash (\#True \wedge more) = more$
by *auto*
have *40*: $\vdash skip \longrightarrow more$
by (*metis DiIntro DiSkipEqvMore int-eq*)
have *41*: $\vdash skip \longrightarrow finite$
by (*metis AndChopB EmptySChop FiniteChopSkipImpFinite Prop11 lift-imp-trans schop-d-def*)
have *5*: $\vdash skip;(\#True\ \ fproj\ \ f) \longrightarrow (more \wedge finite);(\#True\ \ fproj\ \ f)$
by (*simp add*: *40 41 LeftChopImpChop Prop12*)
show *?thesis* **by** (*metis 2 5 FDpSkipEqvMore fdp-d-def inteq-reflection next-d-def*)
qed

lemma *NextODpImpODpNext*:
$\vdash\ \bigcirc\ (odp\ f) \longrightarrow odp\ (\bigcirc\ f)$
proof –
have *1*: $\vdash odp(\bigcirc\ f) = \#True\ oproj\ (skip;f)$
by (*simp add*: *odp-d-def next-d-def*)
have *10*: $\vdash (skip \wedge finite) = skip$
by (*metis FiniteChopEqvDiamond FiniteChopSkipEqvSkipChopFinite NowImpDiamond Prop10 Prop11 SkipChopFiniteImpFinite lift-imp-trans*)
have *2*: $\vdash \#True\ oproj\ (skip;f) = (\#True\ \ fproj\ \ skip);(\#True\ oproj\ f)$
using *OPJ4* **by** (*metis 10 inteq-reflection*)
have *3*: $\vdash (\#True\ \ fproj\ \ skip) = (\#True \wedge more \wedge finite)$
using *PJ3* **by** *blast*
have *4*: $\vdash (\#True \wedge more) = more$
by *auto*
have *40*: $\vdash skip \longrightarrow more$
by (*metis DiIntro DiSkipEqvMore int-eq*)
have *41*: $\vdash skip \longrightarrow finite$
by (*metis AndChopB EmptySChop FiniteChopSkipImpFinite Prop11 lift-imp-trans schop-d-def*)
have *5*: $\vdash skip;(\#True\ oproj\ f) \longrightarrow (more \wedge finite);(\#True\ oproj\ f)$
by (*simp add*: *40 41 LeftChopImpChop Prop12*)
show *?thesis* **by** (*metis 2 5 FDpSkipEqvMore fdp-d-def inteq-reflection next-d-def odp-d-def*)
qed

lemma *BoxStateImportFBp*:
$\vdash \Box(init\ w)\ \longrightarrow fbp\ (\Box(init\ w))$
proof –
have *1*: $\vdash fbp\ (\Box(init\ w)) = (\neg(fdp\ (\Diamond\ (\neg(init\ w)))))$
by (*metis NotFDpEqvFBpNot always-d-def int-eq*)
have *2*: $\vdash\ (\Diamond\ (\neg(init\ w))) = (finite;(\neg(init\ w)))$
by (*simp add*: *sometimes-d-def*)
have *3*: $\vdash fdp\ (finite;(\neg(init\ w))) = (fdp\ finite);\ (fdp\ (\neg(init\ w)))$

by (*simp add*: *PJ4 fdp-d-def*)
have *4*: ⊢ (*fdp finite*) = *finite*
by (*metis EmptyOrMoreSplit FDpEmptyEqvEmpty FiniteAndEmptyEqvEmpty FiniteChopSkipEqvFinite-AndMore*
FiniteChopSkipImpFinite NowImpFDp PJ01 Prop02 Prop10 fdp-d-def int-iffI inteq-reflection)
have *5*: ⊢ (*fdp* (¬(*init w*))) = ((¬(*init w*)) ∧ *finite*)
by (*metis FDpState Initprop*(*2*) *inteq-reflection*)
have *6*: ⊢ *finite*;((¬(*init w*)) ∧ *finite*) ⟶ ◇(¬(*init w*))
by (*metis 2 ChopAndA inteq-reflection*)
show *?thesis*
by (*metis 1 2 3 4 5 6 always-d-def inteq-reflection lift-imp-neg*)
qed

lemma *BoxStateImportOBp*:
⊢ □(*init w*) ⟶ *obp* (□(*init w*))
proof −
have *1*: ⊢ *obp* (□(*init w*)) = (¬(*odp* (◇ (¬(*init w*)))))
by (*metis NotODpEqvOBpNot always-d-def int-eq*)
have *2*: ⊢ (◇ (¬(*init w*))) = (*finite*;(¬(*init w*)))
by (*simp add*: *sometimes-d-def*)
have *3*: ⊢ *odp* (*finite*;(¬(*init w*))) = (*fdp finite*); (*odp* (¬(*init w*)))
unfolding *odp-d-def fdp-d-def*
using *OPJ4*[*of LIFT #True LIFT finite LIFT* (¬(*init w*))]
by (*simp add*: *OPJ4 odp-d-def fdp-d-def*)
have *4*: ⊢ (*fdp finite*) = *finite*
by (*metis EmptyOrMoreSplit FDpEmptyEqvEmpty FiniteAndEmptyEqvEmpty FiniteChopSkipEqvFinite-AndMore*
FiniteChopSkipImpFinite NowImpFDp PJ01 Prop02 Prop10 fdp-d-def int-iffI inteq-reflection)
have *5*: ⊢ (*odp* (¬(*init w*))) = ((¬(*init w*)) ∧ *inf*)
by (*metis ODpState Initprop*(*2*) *inteq-reflection*)
have *6*: ⊢ *finite*;((¬(*init w*)) ∧ *inf*) ⟶ ◇(¬(*init w*))
by (*metis 2 ChopAndA inteq-reflection*)
show *?thesis* **by** (*metis 1 2 3 4 5 6 always-d-def int-eq lift-imp-neg*)
qed

lemma *BoxStateEqvFBpBoxState*:
⊢ *finite*⟶ □ (*init w*) = *fbp*(□ (*init w*))
proof −
have *1*: ⊢ *finite* ⟶ *fbp*(□ (*init w*)) ⟶ □ (*init w*)
by (*metis FBpElim Prop09 inteq-reflection lift-and-com*)
have *2*: ⊢ *fbp*(□ (*init w*)) = (¬(*#True fproj* (¬ □ (*init w*))))
by (*simp add*: *fbp-d-def ufprojection-d-def*)
have *2*: ⊢ □(*init w*) ∧ *finite*⟶ *fdp* (□(*init w*))
by (*metis NowImpFDp*)
have *2*: ⊢ □ (*init w*) ⟶ *fbp*(□ (*init w*))
using *BoxStateImportFBp* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateEqvOBpBoxState*:

$\vdash$ *inf* $\longrightarrow \Box$ (*init w*) = *obp*($\Box$ (*init w*))
proof –
 have *1*: $\vdash$ *inf* $\longrightarrow$ *obp*($\Box$ (*init w*)) $\longrightarrow \Box$ (*init w*)
  by (*metis OBpElim Prop09 inteq-reflection lift-and-com*)
 have *2*: $\vdash$ *obp*($\Box$ (*init w*)) = ($\neg$(#*True oproj* ($\neg$ $\Box$ (*init w*))))
  by (*simp add*: *obp-d-def uoprojection-d-def*)
 have *2*: $\vdash$ $\Box$(*init w*) $\wedge$ *inf* $\longrightarrow$ *odp* ($\Box$(*init w*))
  by (*metis NowImpODp*)
 have *2*: $\vdash$ $\Box$ (*init w*) $\longrightarrow$ *obp*($\Box$ (*init w*))
 using *BoxStateImportOBp* **by** *auto*
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

end

11 Infinite and Finite Interval Temporal Algebra

We have given an algebraic axiom system for Interval Temporal Logic: Interval Temporal Algebra. The axiom system is a combination of a variant of Kleene algebra and Omega algebra plus axioms for linearity and confluence. Kleene algebra and Omega algebra have been defined by Alasdair Armstrong, Georg Struth and Tjark Weber in [1].

theory *ITA*
imports *Semantics Chopstar Omega*
begin

11.1 Definition of Set of intervals and Operations on them

type-synonym *'a iintervals* = *'a nellist set*

definition *lan*:: (*'a*:: *world*) *formula* $\Rightarrow$ *'a iintervals*
where *lan f* = { σ . ($\sigma \models f$) }

definition *fusion* :: *'a iintervals* $\Rightarrow$ *'a iintervals* $\Rightarrow$ *'a iintervals* (**infixl** $\cdot$ *70*)
where $X \cdot Y$ = { σ .
  ($\exists$ $\sigma 1$ $\sigma 2$. σ = *nfuse* $\sigma 1$ $\sigma 2$ $\wedge$ *nfinite* $\sigma 1$ $\wedge$
   ($\sigma 1 \in X$) $\wedge$ ($\sigma 2 \in Y$) $\wedge$ (*nlast* $\sigma 1$ = *nfirst* $\sigma 2$))
  $\vee$ ($\neg$ *nfinite* σ $\wedge$ $\sigma \in X$) }

definition *sempty* :: *'a iintervals* (*SEmpty*)
where
 SEmpty $\equiv$ { σ . *nlength* σ = *0* }

definition *smore* :: *'a iintervals* (*SMore*)
where
 SMore $\equiv$ $-$ *SEmpty*

definition *sskip* :: *'a iintervals* (*SSkip*)
where
 SSkip $\equiv$ $-$(*SEmpty* $\cup$ (*SMore*$\cdot$*SMore*))

definition *sfalse* :: $'a$ *iintervals* (*SFalse*)
where
SFalse $\equiv$ {}

definition *strue* :: $'a$ *iintervals* (*STrue*)
where
STrue $\equiv -\{\}$

definition *sinit* :: $'a$ *iintervals* $\Rightarrow$ $'a$ *iintervals* ((*SInit* -) [*85*] *85*)
where
SInit X $\equiv$ (*X* $\cap$ *SEmpty*)$\cdot$*STrue*

definition *sinf* :: $'a$ *iintervals* (*SInf*)
where *SInf* $\equiv$ *STrue* $\cdot$ *SFalse*

definition *sfinite* :: $'a$ *iintervals* (*SFinite*)
where *SFinite* $\equiv -$ *SInf*

definition *sfmore* :: $'a$ *iintervals* (*SFMore*)
where *SFMore* $\equiv$ *SFinite* $\cap$ *SMore*

definition *sfin* :: $'a$ *iintervals* $\Rightarrow$ $'a$ *iintervals* ((*SFin* -) [*85*] *85*)
where
SFin X $\equiv$ *SFinite*$\cdot$(*X* $\cap$ *SEmpty*)

definition *ssometime* :: $'a$ *iintervals* $\Rightarrow$ $'a$ *iintervals* ((*SSometime* -) [*85*] *85*)
where
SSometime X $\equiv$ *SFinite*$\cdot$*X*

definition *salways* :: $'a$ *iintervals* $\Rightarrow$ $'a$ *iintervals* ((*SAlways* -) [*85*] *85*)
where
SAlways X $\equiv -$(*SSometime* ($-X$))

definition *sdi* :: $'a$ *iintervals* $\Rightarrow$ $'a$ *iintervals* ((*SDi* -) [*85*] *85*)
where
SDi X $\equiv$ *X*$\cdot$*STrue*

definition *sbi* :: $'a$ *iintervals* $\Rightarrow$ $'a$ *iintervals* ((*SBi* -) [*85*] *85*)
where
SBi X $\equiv -$(*SDi* ($-X$))

definition *sda* :: $'a$ *iintervals* $\Rightarrow$ $'a$ *iintervals* ((*SDa* -) [*85*] *85*)
where
SDa X $\equiv$ *SFinite*$\cdot$*X*$\cdot$*STrue*

definition *sba* :: $'a$ *iintervals* $\Rightarrow$ $'a$ *iintervals* ((*SBa* -) [*85*] *85*)
where
SBa X $\equiv -$(*SDa* ($-X$))

definition *snext* :: *'a iintervals* ⇒ *'a iintervals* ((*SNext* -) [*85*] *85*)
where
SNext X ≡ *SSkip*·*X*

definition *swnext* :: *'a iintervals* ⇒ *'a iintervals* ((*SWnext* -) [*85*] *85*)
where
SWnext X ≡ (−(*SSkip*·(−*X*)))

definition *sprev* :: *'a iintervals* ⇒ *'a iintervals* ((*SPrev* -) [*85*] *85*)
where
SPrev X ≡ *X*·*SSkip*

definition *swprev* :: *'a iintervals* ⇒ *'a iintervals* ((*SWprev* -) [*85*] *85*)
where
SWprev X ≡ (−((−*X*)·*SSkip*))

primrec *spower* :: *'a iintervals* ⇒ *nat* ⇒ *'a iintervals* ((*SPower* - -) [*88*,*88*] *87*)
where
pwr-0 : *SPower X 0* = *SEmpty*
| *pwr-Suc*: *SPower X* (*Suc n*) = ((*X* ∩ *SFinite*)·(*SPower X n*))

definition *sfpowerstar* :: *'a iintervals* ⇒ *'a iintervals* ((*SFPowerstar* -) [*85*] *85*)
where
SFPowerstar X ≡ (⋃ *n*. *SPower X n*)

definition *spowerstar* :: *'a iintervals* ⇒ *'a iintervals* ((*SPowerstar* -) [*85*] *85*)
where
SPowerstar X ≡ (*SFPowerstar X*)·(*SEmpty* ∪ (*X* ∩ *SInf*))

definition *sstar* :: *'a iintervals* ⇒ *'a iintervals* ((*SStar* -) [*85*] *85*)
where
SStar X ≡ *SPowerstar*(*X* ∩ *SMore*)

11.2 Simplification Lemmas

lemma *snot-elim* :
x ∈ −*X* ⟷ *x* ∉ *X*
by *simp*

lemma *sor-elim* :
x∈ (*X* ∪ *Y*) ⟷ (*x* ∈ *X* ∨ *x*∈*Y*)
by *simp*

lemma *sand-elim* :
x∈ (*X* ∩ *Y*) ⟷ (*x* ∈ *X* ∧ *x*∈*Y*)
by *simp*

lemma *sfalse-elim* :
σ ∉ *SFalse*
by (*simp add*: *sfalse-def*)

lemma *strue-elim* :
  $\sigma \in STrue$
by (*simp add*: *strue-def*)

lemma *sempty-elim* :
  $\sigma \in SEmpty \longleftrightarrow (\ nlength\ \sigma = 0\ )$
by (*simp add*: *sempty-def*)

lemma *smore-elim* :
  $\sigma \in SMore \longleftrightarrow$
  $(\ nlength\ \sigma > 0\ )$
by (*simp add*: *sempty-elim smore-def*)

lemma *fusion-iff*:
  $\sigma \in X{\cdot}Y \longleftrightarrow$
  (
  $(\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
  $(\sigma 1 \in X) \wedge (\sigma 2\ \in Y)\ \wedge (nlast\ \sigma 1 = nfirst\ \sigma 2))$
  $\vee (\neg\ nfinite\ \sigma \wedge \sigma \in X)\ )$

by (*unfold fusion-def*) *auto*

lemma *fusion-iff-1*:
  $\sigma \in X{\cdot}Y \longleftrightarrow$
  $(\ (\exists\, n \le nlength\ \sigma.\ \ (\ (ntaken\ n\ \sigma) \in X\ ) \wedge (\ (ndropn\ n\ \sigma) \in\ Y)\ )$
  $\vee (\neg\ nfinite\ \sigma \wedge \sigma \in X)$
)
using *fusion-iff*[*of* σ *X Y*] *nfuse-ntaken-ndropn*[*of* - σ]
by (*simp add*: *chop-nfuse-2*)

lemma *smore-fusion-smore* :
  $\sigma \in (SMore{\cdot}SMore) \longleftrightarrow (\ (enat\ 1) < nlength\ \sigma)$
using *fusion-iff-1*[*of* σ *SMore SMore*]
proof (*auto simp add*: *smore-elim*)
  show $\bigwedge n\ na.$
    $\sigma \in SMore \cdot SMore \Longrightarrow$
    $enat\ n \le nlength\ \sigma \Longrightarrow$
    $enat\ n \ne 0 \Longrightarrow$
    $nlength\ \sigma - enat\ n \ne 0 \Longrightarrow enat\ na \le nlength\ \sigma \Longrightarrow enat\ na \ne 0 \Longrightarrow$
    $nlength\ \sigma \ne 0 \Longrightarrow nlength\ \sigma - enat\ na \ne 0 \Longrightarrow$
    $enat\ (Suc\ 0) < nlength\ \sigma$
  by (*metis One-nat-def antisym-conv3 enat-ord-code*(*4*) *idiff-self ileI1 le-less-trans less-le not-iless0 one-eSuc one-enat-def*)
  show $\bigwedge n.\ \sigma \in SMore \cdot SMore \Longrightarrow$
    $enat\ n \le nlength\ \sigma \Longrightarrow$
    $enat\ n \ne 0 \Longrightarrow nlength\ \sigma - enat\ n \ne 0 \Longrightarrow$
    $\neg\ nfinite\ \sigma \Longrightarrow$
    $nlength\ \sigma \ne 0 \Longrightarrow$
    $enat\ (Suc\ 0) < nlength\ \sigma$

by (*metis Suc-ile-eq i0-less nlength-eq-enat-nfiniteD order.not-eq-order-implies-strict zero-enat-def*)
show $\bigwedge n.\ \sigma \in SMore \cdot SMore \Longrightarrow$
$\neg\ nfinite\ \sigma \Longrightarrow$
$enat\ n \le nlength\ \sigma \Longrightarrow$
$enat\ n \ne 0 \Longrightarrow$
$nlength\ \sigma \ne 0 \Longrightarrow$
$nlength\ \sigma - enat\ n \ne 0 \Longrightarrow$
$enat\ (Suc\ 0) < nlength\ \sigma$
by (*metis Suc-ile-eq i0-less nlength-eq-enat-nfiniteD order.not-eq-order-implies-strict zero-enat-def*)
show $\sigma \in SMore \cdot SMore \Longrightarrow \neg\ nfinite\ \sigma \Longrightarrow nlength\ \sigma \ne 0 \Longrightarrow enat\ (Suc\ 0) < nlength\ \sigma$
by (*metis One-nat-def ileI1 linorder-cases nlength-eq-enat-nfiniteD not-less-zero one-eSuc one-enat-def order.not-eq-order-implies-strict*)
show $\sigma \notin SMore \cdot SMore \Longrightarrow$
$enat\ (Suc\ 0) < nlength\ \sigma \Longrightarrow$
$\forall n.\ enat\ n \le nlength\ \sigma \longrightarrow enat\ n = 0 \lor nlength\ \sigma = 0 \lor nlength\ \sigma - enat\ n = 0 \Longrightarrow$
$nfinite\ \sigma \Longrightarrow$
False
proof −
assume *a1*: $enat\ (Suc\ 0) < nlength\ \sigma$
assume $nfinite\ \sigma$
assume *a2*: $\forall n.\ enat\ n \le nlength\ \sigma \longrightarrow enat\ n = 0 \lor nlength\ \sigma = 0 \lor nlength\ \sigma - enat\ n = 0$
have $enat\ (Suc\ 0) = 1$
using *One-nat-def one-enat-def* **by** *presburger*
then have *f3*: $enat\ 1 < nlength\ \sigma$
using *a1 one-enat-def* **by** *presburger*
have *f4*: $enat\ 1 \le enat\ 1$
by *blast*
have $enat\ 1 \ne \infty$
by *blast*
then show *False*
using *f4 f3 a2*
by (*metis* (*no-types*) *dual-order.strict-iff-order enat-diff-cancel-left gr-implies-not-zero idiff-self one-enat-def zero-neq-one*)
qed
qed

lemma *sskip-elim* :
$\sigma \in SSkip \longleftrightarrow$
$(nlength\ \sigma = 1)$
using *sskip-def smore-fusion-smore*
by (*metis One-nat-def Suc-ile-eq less-numeral-extra(4) one-enat-def order.not-eq-order-implies-strict sempty-elim smore-def smore-elim snot-elim sor-elim zero-enat-def zero-one-enat-neq(1)*)

lemma *sinfinite-elim*:
$\sigma \in SInf \longleftrightarrow$
$(\neg\ nfinite\ \sigma)$
by (*simp add*: *fusion-iff-1 sfalse-elim sinf-def strue-elim*)

lemma *sfinite-elim*:

$\sigma \in SFinite \longleftrightarrow$
$(nfinite\ \sigma)$
by (*simp add*: *sfinite-def sinfinite-elim*)

lemma *ffusion-iff*:
$\sigma \in (X \cap SFinite)\cdot Y \longleftrightarrow$
(
$(\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$(\sigma 1 \in X) \wedge (\sigma 2\ \in Y)\ \wedge (nlast\ \sigma 1 = nfirst\ \sigma 2))$
)
unfolding *fusion-def*
by (*simp add*: *sfinite-elim*) *blast*

lemma *ffusion-iff-1*:
$\sigma \in (X \cap SFinite)\cdot Y \longleftrightarrow$
$((\ (\exists n \le nlength\ \sigma.\ \ (\ (ntaken\ n\ \sigma) \in X\ ) \wedge (\ (ndropn\ n\ \sigma) \in Y))$
)
)
using *fusion-iff-1*[*of* σ $X \cap SFinite$ Y]
by (*meson nfinite-ntaken sand-elim sfinite-elim*)

lemma *sfmore-elim*:
$\sigma \in SFMore \longleftrightarrow$
$(nfinite\ \sigma \wedge 0 < nlength\ \sigma)$
by (*simp add*: *sfinite-elim sfmore-def smore-elim*)

lemma *spower-elim-zero* :
$\sigma \in SPower\ X\ 0 \longleftrightarrow \sigma \in SEmpty$
by *simp*

lemma *spower-elim-suc* :
$\sigma \in SPower\ X\ (Suc\ n) \longleftrightarrow \sigma \in (X \cap SFinite)\cdot(SPower\ X\ n)$
by *simp*

lemma *spower-elim-suc-1* :
$\sigma \in (X \cap SFinite)\cdot(SPower\ X\ n) \longleftrightarrow$
$(\ (\exists\ \sigma 1\ \sigma 2.\ \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
$\sigma 1 \in X\ \ \wedge \sigma 2\ \in (SPower\ X\ n)\ \ \wedge$
$nlast\ \sigma 1 = nfirst\ \sigma 2\ )$
)
by (*simp add*: *ffusion-iff*)

lemma *ffusionimp*:
assumes $Y0 \subseteq Y1$
shows $\sigma \in (X \cap SFinite)\cdot Y0\ \longrightarrow \sigma \in (X \cap SFinite)\cdot Y1$
using *assms* **unfolding** *ffusion-iff-1* **by** (*auto*)

lemma *spower-finite*:
$(SPower\ X\ n) \subseteq SFinite$
proof (*induct n*)

case *0*
then show *?case*
  by (*metis nlength-eq-enat-nfiniteD sempty-elim sfinite-elim spower.simps*(*1*) *subsetI zero-enat-def*)
next
case (*Suc n*)
then show *?case*
  proof −
   have *1*: (*SPower X* (*Suc n*)) = (*X* ∩ *SFinite*)·(*SPower X n*)
     by *simp*
   have *2*: *SPower X n* ⊆ *SFinite*
     using *Suc.hyps* **by** *blast*
   have *3*: (*X* ∩ *SFinite*)· *SFinite* ⊆ *SFinite*
     using *ffusion-iff-1*[*of - X SFinite*]
     by (*simp add*: *sfinite-elim*)
       (*metis ComplI sfinite-def sinfinite-elim subsetI*)
   have *4*: (*X* ∩ *SFinite*)·(*SPower X n*) ⊆ (*X* ∩ *SFinite*)· *SFinite*
     using *Suc.hyps ffusionimp* **by** *blast*
   show *?thesis*
     using *3 4* **by** *auto*
 qed
qed

lemma *sfpowerstar-elim*:
  σ ∈ *SFPowerstar X* ⟷ (∃ *n*. *σ* ∈ *SPower X n*)
by (*simp add*: *sfpowerstar-def*)

lemma *sfpowerstar-elim-1*:
  (∃ *n*. *σ* ∈ *SPower X n*) ⟷ (*σ* ∈ *SPower X 0* ∨ (∃ *n*. *σ* ∈ *SPower X* (*Suc n*)))
by (*metis not0-implies-Suc*)

lemma *sfpowerstar-suc*:
  (∃ *n*. *σ* ∈ *SPower X* (*Suc n*)) ⟷ (∃ *n*. *σ* ∈ (*X* ∩ *SFinite*)·(*SPower X n*))
by *simp*

lemma *sfpowerstar-suc-1*:
   (∃ *n*. *σ* ∈ (*X* ∩ *SFinite*)·(*SPower X n*)) ⟷ *σ* ∈ (*X* ∩ *SFinite*)·(*SFPowerstar X*)
unfolding *fusion-iff*
by (*cases σ*) (*auto simp add*: *sfpowerstar-elim*)

lemma *sfpowerstar-eqv-sem*:
 σ ∈ *SFPowerstar X* ⟷ (*σ* ∈ *SEmpty* ∨ *σ* ∈ (*X* ∩ *SFinite*)·(*SFPowerstar X*))
by (*simp add*: *sfpowerstar-elim sfpowerstar-elim-1 sfpowerstar-suc-1*)

lemma *sfpowerstar-eqv*:
 SFPowerstar X = *SEmpty* ∪ (*X* ∩ *SFinite*)·(*SFPowerstar X*)
using *sfpowerstar-eqv-sem* **by** *blast*

lemma *sfpowerstar-eqv-1*:
  (⋃ *n*. *SPower X n*) = *SEmpty* ∪ (*X* ∩ *SFinite*)·(⋃ *n*. *SPower X n*)
using *sfpowerstar-eqv* **by** (*simp add*: *sfpowerstar-def*)

11.3 Algebraic Laws

11.3.1 Commutative Additive Monoid

lemma *UnionCommute*:
$(X::'a\ iintervals) \cup Y = Y \cup X$
by (*simp add*: *Un-commute*)

lemma *UnionSFalse*:
$X \cup SFalse = X$
by (*simp add*: *sfalse-def*)

lemma *UnionAssoc*:
$(X::'a\ iintervals) \cup (Y \cup Z) = (X \cup Y) \cup Z$
by (*simp add*: *sup-assoc*)

11.3.2 Boolean algebra

lemma *Huntington*:
$(X::'a\ iintervals) = -(-X \cup -Y) \cup -(-X \cup Y)$
by *auto*

lemma *Morgan*:
$(X::'a\ iintervals) \cap Y = -(-X \cup -Y)$
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*)

11.3.3 multiplicative monoid

lemma *FusionSEmptyLsem*:
$\sigma \in SEmpty \cdot X \longleftrightarrow \sigma \in X$
using *fusion-iff-1*[*of* σ *SEmpty X*]
by (*auto simp add*: *fusion-iff-1 sempty-elim nlength-eq-enat-nfiniteD zero-enat-def*)
(*metis le-numeral-extra*(*3*) *min.bounded-iff min-enat-simps*(*3*) *ndropn-0 zero-enat-def*,
metis le-numeral-extra(*3*) *min.bounded-iff min-enat-simps*(*3*) *ndropn-0 zero-enat-def*)

lemma *FusionSEmptyL* :
$SEmpty \cdot X = X$
using *set-eqI*[*of SEmpty·X X*] *FusionSEmptyLsem*
by *auto*

lemma *FusionSEmptyRsem* :
$\sigma \in X \cdot SEmpty \longleftrightarrow \sigma \in X$

using *fusion-iff-1*[*of* σ *X SEmpty*]
by (*auto simp add: fusion-iff-1 sempty-elim*)
(*metis is-NNil-ndropn le-numeral-extra*(*3*) *ndropn-0 ndropn-nlength ntaken-all zero-enat-def*, *metis add.right-neutral le-iff-add ndropn-all ndropn-nlength nfinite-nlength-enat nlength-NNil ntaken-all*)

lemma *FusionSEmptyR* :
$X \cdot SEmpty = X$
using *set-eqI*[*of* $X \cdot SEmpty$ *X*] *FusionSEmptyRsem* **by** *auto*

lemma *FusionAssocA*:
assumes $x \in X \cdot (Y \cdot Z)$
shows $x \in (X \cdot Y) \cdot Z$
proof –
have *1*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2)$
$\vee (\neg\ nfinite\ x \wedge x \in X)$
using *assms fusion-iff*[*of x X Y* $\cdot$ *Z*] **by** *auto*
have *2*: $\neg\ nfinite\ x \wedge x \in X \Longrightarrow x \in (X \cdot Y) \cdot Z$
by (*simp add: fusion-iff*)
have *3*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2) \Longrightarrow$
$x \in (X \cdot Y) \cdot Z$
proof –
assume *a0*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2)$
show $x \in (X \cdot Y) \cdot Z$
proof –
obtain $\sigma 1$ $\sigma 2$ **where** *5*:
$x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2$
using *a0* **by** *auto*
have *6*: $(\exists \sigma 3\ \sigma 4.\ \sigma 2 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge nlast\ \sigma 3 = nfirst\ \sigma 4) \vee$
$(\neg\ nfinite\ \sigma 2 \wedge \sigma 2 \in Y)$
using *fusion-iff*[*of* $\sigma 2$ *Y Z*] *5* **by** *simp*
have *7*: $(\neg\ nfinite\ \sigma 2 \wedge \sigma 2 \in Y) \Longrightarrow x \in (X \cdot Y) \cdot Z$
by (*metis 5 fusion-iff ndropn-nfuse nfinite-ndropn*)
have *8*: $(\exists \sigma 3\ \sigma 4.\ \sigma 2 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge nlast\ \sigma 3 = nfirst\ \sigma 4) \Longrightarrow$
$x \in (X \cdot Y) \cdot Z$
proof –
assume *a1*: $(\exists \sigma 3\ \sigma 4.\ \sigma 2 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge nlast\ \sigma 3 = nfirst\ \sigma 4)$
show $x \in (X \cdot Y) \cdot Z$
proof –
obtain $\sigma 3$ $\sigma 4$ **where** *10*:
$\sigma 2 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge nlast\ \sigma 3 = nfirst\ \sigma 4$
using *a1* **by** *auto*
have *11*: $x = nfuse\ \sigma 1\ (nfuse\ \sigma 3\ \sigma 4)$
using *10 5* **by** *blast*
have *12*: $x = nfuse\ (nfuse\ \sigma 1\ \sigma 3)\ \sigma 4$
by (*simp add: 10 5 nfirst-nfuse nfuseassoc*)
have *13*: $(nfuse\ \sigma 1\ \sigma 3) \in X \cdot Y$
using *fusion-iff*[*of* (*nfuse* $\sigma 1$ $\sigma 3$) *X Y*]
using *10 5 nfirst-nfuse* **by** *fastforce*
show *?thesis* **using** *fusion-iff*[*of x* $X \cdot Y$ *Z*]

using *10 12 13 5*
by (*metis nfuse-nlength nfinite-nlength-enat nfirst-nfuse nlength-eq-enat-nfiniteD plus-enat-simps*(*1*) *nlast-nfuse*)
qed
qed
show *?thesis*
using *6 7 8* **by** *blast*
qed
qed
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *FusionAssocB*:
assumes $x \in (X \cdot Y) \cdot Z$
shows $x \in X \cdot (Y \cdot Z)$
proof –
have *1*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2) \vee (\neg\ nfinite\ x \wedge x \in X \cdot Y)$
using *assms fusion-iff*[*of x X·Y Z*] **by** *auto*
have *2*: $(\neg\ nfinite\ x \wedge x \in X \cdot Y) \Longrightarrow x \in X \cdot (Y \cdot Z)$
by (*metis fusion-iff-1 nfinite-ndropn-b*)
have *3*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2) \Longrightarrow x \in X \cdot (Y \cdot Z)$
proof –
assume *a0*: $(\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2)$
show $x \in X \cdot (Y \cdot Z)$
proof –
obtain $\sigma 1\ \sigma 2$ **where** *4*:
$x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge nlast\ \sigma 1 = nfirst\ \sigma 2$
using *a0* **by** *auto*
have *5*: $\exists \sigma 3\ \sigma 4.\ \sigma 1 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in X \wedge \sigma 4 \in Y \wedge nlast\ \sigma 3 = nfirst\ \sigma 4$
using *4 fusion-iff*[*of σ1 X Y*]
using *nfuse-nappend* **by** *fastforce*
obtain $\sigma 3\ \sigma 4$ **where** *6*:
$\sigma 1 = nfuse\ \sigma 3\ \sigma 4 \wedge nfinite\ \sigma 3 \wedge \sigma 3 \in X \wedge \sigma 4 \in Y \wedge nlast\ \sigma 3 = nfirst\ \sigma 4$
using *5* **by** *auto*
have *7*: $x = nfuse\ (nfuse\ \sigma 3\ \sigma 4)\ \sigma 2$
by (*simp add*: *4 6*)
have *8*: $x = nfuse\ \sigma 3\ (nfuse\ \sigma 4\ \sigma 2)$
using *4 6 nfuseassoc* **by** *metis*
have *9*: $(nfuse\ \sigma 4\ \sigma 2) \in Y \cdot Z$
using *fusion-iff*[*of* (*nfuse σ4 σ2*) *Y Z*]
by (*metis 4 6 nfuse-nappend nlast-nfuse*)
have *10*: $nlast\ \sigma 3 = nfirst\ (nfuse\ \sigma 4\ \sigma 2)$
using *4 6 nfirst-nfuse nlast-nfuse* **by** *fastforce*
show *?thesis*
using *fusion-iff*[*of x X Y·Z*]
using *10 6 8 9* **by** *auto*
qed

qed
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *FusionAssoc* :
$X{\cdot}(Y{\cdot}Z) = (X{\cdot}Y){\cdot}Z$
using *set-eqI*[*of* $X{\cdot}(Y{\cdot}Z)$ $(X{\cdot}Y){\cdot}Z$]
FusionAssocA FusionAssocB **by** *blast*

lemma *FusionAssoc1*:
$(X{\cdot}Y){\cdot}Z = X{\cdot}(Y{\cdot}Z)$
by (*simp add*: *FusionAssoc*)

— left and right distributivity

lemma *FusionUnionDistLsem*:
$x{\in}(X \cup Y){\cdot}Z \longleftrightarrow x \in (X{\cdot}Z) \cup (Y{\cdot}Z)$
by (*auto simp add*: *fusion-iff*)

lemma *FusionUnionDistL*:
$(X \cup Y){\cdot}Z = (X{\cdot}Z) \cup (Y{\cdot}Z)$
using *FusionUnionDistLsem*[*of* - X Y Z] **by** *fastforce*

lemma *FusionUnionDistRsem*:
$x{\in} X{\cdot}(Y \cup Z) \longleftrightarrow x \in (X{\cdot}Y) \cup (X{\cdot}Z)$
by (*auto simp add*: *fusion-iff*)

lemma *FusionUnionDistR*:
$X{\cdot}(Y \cup Z) = (X{\cdot}Y) \cup (X{\cdot}Z)$
using *FusionUnionDistRsem*[*of* - X Y Z] **by** *fastforce*

— left and right annihilation
lemma *SFalseFusion*:
SFalse $\cdot$ X = *SFalse*
by (*simp add*: *fusion-def sfalse-def*)

lemma *FusionSFalse*:
$X{\cdot}$*SFalse* = ($X \cap$ *SInf*)
by (*auto simp add*: *fusion-def sfalse-def sinfinite-elim*)

— idempotency
lemma *UnionIdem*:
$(X{::}'a\ iintervals) \cup X = X$
by *simp*

11.3.4 Subsumption order

lemma *Subsumption*:
$((X :: 'a\ iintervals) \subseteq Y) = (X \cup Y = Y)$
by *auto*

11.3.5 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*:
$(X \cap SFinite)\cdot(SPower\ X\ n) = (SPower\ X\ n)\cdot(X \cap SFinite)$
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*: $X \cdot (SPower\ X\ n \cdot Y) \subseteq Z$
     using *FusionRuleR Suc.hyps assms* **by** *blast*
   have $X \cap SFinite \subseteq X$
     by *blast*
   then show *?thesis*
     using *f1* **by** (*metis* (*no-types*) *FusionAssoc FusionRuleL order-trans pwr-Suc*)
  qed
qed

lemma *fusion-inductl-finite*:
assumes $Y \cup (X \cap SFinite)\cdot Z \subseteq Z$
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*: $SPower\ X\ n \cdot\ Y \subseteq Z$
    using *Suc.hyps assms* **by** *blast*
  then have $SPower\ X\ n \cdot\ Y \cup Z = Z$
    by *blast*
  then show *?thesis*
    using *f1*
    by (*metis FusionAssoc1 FusionUnionDistR assms le-supE pwr-Suc*)
 qed
qed

lemma *fusion-inductl-fmore*:
 assumes $Y \cup (X \cap SFMore) \cdot Z \subseteq Z$
 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 *1*: $SPower\ X\ (Suc\ n) \cdot\ Y = ((X \cap SFinite) \cdot (SPower\ X\ n)) \cdot Y$
    by *simp*
  have *2*: $((X \cap SFinite) \cdot (SPower\ X\ n)) \cdot\ Y = (X \cap SFinite) \cdot ((SPower\ X\ n) \cdot Y)$
    by (*simp add*: *FusionAssoc*)
  have *3*: $(X \cap SFinite) \cdot ((SPower\ X\ n) \cdot Y) \subseteq (X \cap SFinite) \cdot Z$
    using *FusionRuleR Suc.hyps assms* **by** *blast*
  have *31*: $X \cap SFinite = ((X \cap SFMore) \cup ((X \cap SEmpty) \cap SFinite))$
    by (*simp add*: *sfmore-def smore-def*) *blast*
  have *4*: $(X \cap SFinite) \cdot Z = ((X \cap SFMore) \cdot Z \cup ((X \cap SEmpty) \cap SFinite) \cdot Z)$
    by (*simp add*: *31 FusionUnionDistL*)
  have *5*: $(X \cap SFMore) \cdot Z \subseteq Z$
    using *assms* **by** *auto*
  have *6*: $((X \cap SEmpty) \cap SFinite) \cdot Z \subseteq Z$
    by (*metis FusionRuleL FusionSEmptyL inf-assoc inf-commute inf-le1*)
  show *?thesis*
    using *1 2 3 4 5 6* **by** *blast*
 qed
qed

lemma *fusion-inductl-inf*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $((SPower\ X\ n) \cdot (X \cap SInf)) \cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case*
by (*metis FusionAssoc FusionRuleL FusionRuleR FusionSEmptyL FusionSFalse le-supE order-trans pwr-0 sfalse-elim subsetI*)
next
case (*Suc n*)
then show *?case*
proof –
have *f2*: $Z = Z \cup (Y \cup X \cdot Z)$
using *assms* **by** *blast*
have $SPower\ X\ n \cdot (X \cap SInf) \cdot Y \subseteq Z$
using *Suc*(*1*) *assms* **by** *blast*
then have $Z = Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y$
by *blast*
then have *f3*: $(X \cup X \cap SFinite) \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y) = X \cdot Z$
by *force*
have $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y = X \cap SFinite \cdot (SPower\ X\ n \cdot (X \cap SInf) \cdot Y)$
by (*simp add*: *FusionAssoc*)
then have $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y \cup X \cap SFinite \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y) = X \cap SFinite \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y)$
using *FusionUnionDistR* **by** *blast*
then have $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y \cup X \cdot Z = X \cdot Z$
using *f3 FusionUnionDistL* **by** *blast*
then show *?thesis*
using *f2* **by** *blast*
qed
qed

lemma *fusion-inductl-infmore*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $((SPower\ X\ n) \cdot ((X \cap SMore) \cap SInf)) \cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case*
by (*metis* (*no-types*, *lifting*) *Diff-Compl Int-assoc Un-Diff-Int compl-inf double-compl fusion-inductl-inf inf.absorb-iff2 pwr-0 sfinite-def smore-def spower-finite sup-left-idem*)
next
case (*Suc n*)
then show *?case*
proof –
have *f1*: $\bigwedge n.\ SPower\ X\ n \cdot (X \cap SInf) \cdot Y \subseteq Z$
by (*meson assms fusion-inductl-inf*)
have $X \cap SMore \cap SInf \subseteq X \cap SInf$
by *blast*

then show *?thesis*
  using *f1* **by** (*metis* (*no-types*) *FusionRuleL FusionRuleR inf.orderE le-inf-iff*)
qed
qed

lemma *fusion-inductl-more*:
 assumes $Y \cup X \cdot Z \subseteq Z$
 shows $(SPower\ (X \cap SMore)\ \ n) \cdot Y \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyL*)
next
case (*Suc n*)
then show *?case*
by (*metis FusionUnionDistL fusion-inductl sup.boundedE sup.boundedI sup-inf-absorb*)
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*: $Z \cdot X \subseteq Z$
  using *assms* **by** *auto*
have $\forall S.\ Y \cdot (SPower\ X\ n \cdot S) \subseteq Z \cdot S$
  using *FusionAssoc FusionRuleL Suc.hyps assms* **by** *blast*
then show *?thesis*
  using *f1* **by** (*metis* (*no-types*) *FusionRuleR Int-iff order-trans pwr-Suc spower-commutes subsetI*)
qed
qed

lemma *SInfSFinite*:
 $X = (X \cap SFinite) \cup (X \cap SInf)$
using *sfinite-def* **by** *auto*

lemma *fusion-inductr-inf*:
assumes $Y \cup Z \cdot X \subseteq Z$
 shows $Y \cdot ((SPower\ X\ n) \cdot (X \cap SInf)) \subseteq Z$
proof −
 have *1*: $Y \cdot (SPower\ X\ n) \subseteq Z$
   using *assms fusion-inductr* **by** *blast*
 show *?thesis*
 proof −

have *f4*: $Y \cdot SPower\ X\ n = Y \cdot SPower\ X\ n \cap Z$
  using *1* **by** *blast*
have $Y \cdot SPower\ X\ n \cup Z = Z$
  by (*metis 1 subset-Un-eq*)
then have *f5*: $Z \cdot (X \cap SInf) = Z \cdot (X \cap SInf) \cup Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf)$
  using *f4* **by** (*metis* (*no-types*) *FusionUnionDistL sup.commute*)
have $Y \cdot SPower\ X\ n = Y \cdot SPower\ X\ n \cap Z$
  using *f4* **by** *auto*
then have $Y \cdot (SPower\ X\ n \cdot (X \cap SInf)) = Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf)$
  by (*simp add*: *FusionAssoc1*)
then show *?thesis*
  proof −
  have *f1*: $Y \cup Z \cdot X \cup Z = Z$
    by (*meson Subsumption assms*)
  have *f2*: $Y \cdot SPower\ X\ n \cap Z \cdot X \cup Z \cdot X = Z \cdot X$
    by (*metis* (*no-types*) *FusionUnionDistL* ‹$Y \cdot SPower\ X\ n \cup Z = Z$› *f4*)
  have $Y \cdot SPower\ X\ n \cap Z \cdot X \cup Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf) = Y \cdot SPower\ X\ n \cap Z \cdot X$
    by (*metis* (*no-types*) *FusionUnionDistR sup-inf-absorb*)
  then show *?thesis*
    using *f2 f1* ‹$Y \cdot (SPower\ X\ n \cdot (X \cap SInf)) = Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf)$› **by** *auto*
  qed
qed
qed

lemma *fusion-inductr-more*:
assumes $Y \cup Z{\cdot}X \subseteq Z$
shows $Y{\cdot}(SPower\ (X \cap SMore)\ \ n) \subseteq Z$
using *assms*
proof (*induct n*)
case *0*
then show *?case* **by** (*simp add*: *FusionSEmptyR*)
next
case (*Suc n*)
then show *?case*
by (*metis FusionUnionDistR fusion-inductr le-supE sup.boundedI sup-inf-absorb*)
qed

lemma *FusionUnionSPowersem*:
$\sigma \in Y \cdot (\bigcup n.\ \ (SPower\ X\ n){\cdot}Z) \longleftrightarrow \sigma \in (\bigcup n.\ Y \cdot ((SPower\ X\ n){\cdot}Z))$
by (*simp add*: *fusion-iff*) *blast*

lemma *FusionUnionSPower*:
$Y \cdot (\bigcup n.\ \ (SPower\ X\ n){\cdot}Z) = (\bigcup n.\ Y \cdot ((SPower\ X\ n){\cdot}Z))$
using *FusionUnionSPowersem* **by** *blast*

lemma *FusionUnionSPower1*:
$Y \cdot (\bigcup n.\ \ (SPower\ X\ n)) = (\bigcup n.\ Y \cdot ((SPower\ X\ n)))$
using *FusionUnionSPower*[*of Y X SEmpty*]
by (*metis* (*no-types*, *lifting*) *FusionSEmptyR Sup.SUP-cong*)

lemma *UnionFusionSPowersem*:
$\sigma \in (\bigcup n.\ Z{\cdot}(SPower\ X\ n)){\cdot}Y \longleftrightarrow \sigma \in (\bigcup n.\ (Z{\cdot}(SPower\ X\ n)){\cdot}Y)$
by (*simp add*: *fusion-iff*) *blast*

lemma *UnionFusionSPower*:
$(\bigcup n.\ Z{\cdot}(SPower\ X\ n)){\cdot}Y = (\bigcup n.\ (Z{\cdot}(SPower\ X\ n)){\cdot}Y)$
using *UnionFusionSPowersem* **by** *blast*

lemma *UnionFusionSPower1*:
$(\bigcup n.\ (SPower\ X\ n)){\cdot}Y = (\bigcup n.\ ((SPower\ X\ n)){\cdot}Y)$
using *UnionFusionSPower*[*of SEmpty X Y*]
by (*metis* (*no-types*, *lifting*) *FusionSEmptyL Sup.SUP-cong*)

lemma *spowercommute*:
$(X \cap SFinite){\cdot}(\bigcup n.\ (SPower\ X\ n)) = (\bigcup n.\ (SPower\ X\ n){\cdot}(X \cap SFinite))$
by (*metis* (*no-types*, *lifting*) *FusionUnionSPower1 Sup.SUP-cong spower-commutes*)

lemma *sstar-contl*:
$Y \cdot (SStar\ X) = (\bigcup n.\ Y \cdot (SPower\ (X \cap SMore)\ n){\cdot}(SEmpty \cup X \cap SMore \cap SInf))$
proof –
have *1*: $Y \cdot (SStar\ X) = Y \cdot ((\bigcup n.\ (\ (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf))$
by (*simp add*: *sstar-def spowerstar-def sfpowerstar-def*)
have *2*: $Y \cdot ((\bigcup n.\ (\ (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)) =$
$(Y \cdot (\bigcup n.\ (SPower\ (X \cap SMore)\ n))){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf)$
using *FusionAssoc* **by** *blast*
have *3*: $(Y \cdot (\bigcup n.\ (SPower\ (X \cap SMore)\ n))) = (\bigcup n.\ Y{\cdot}\ (SPower\ (X \cap SMore)\ n))$
using *FusionUnionSPower1* **by** *blast*
have *4*: $(Y \cdot (\bigcup n.\ (SPower\ (X \cap SMore)\ n))){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf) =$
$(\bigcup n.\ Y{\cdot}\ (SPower\ (X \cap SMore)\ n)){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf)$
by (*simp add*: *3*)
have *5*: $(\bigcup n.\ Y{\cdot}\ (SPower\ (X \cap SMore)\ n)){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf) =$
$(\bigcup n.\ (Y{\cdot}\ (SPower\ (X \cap SMore)\ n)){\cdot}\ (SEmpty \cup X \cap SMore \cap SInf))$
using *UnionFusionSPower*[*of Y X* $\cap$ *SMore* (*SEmpty* $\cup$ *X* $\cap$ *SMore* $\cap$ *SInf*)]
by *auto*
show *?thesis*
by (*simp add*: *1 2 4 5*)
qed

lemma *sstar-contr*:
$(SStar\ X){\cdot}Y = (\bigcup n.\ ((SPower\ (X \cap SMore)\ n){\cdot}(SEmpty \cup X \cap SMore \cap SInf)){\cdot}Y)$
proof –
have *1*: $(SStar\ X){\cdot}Y = ((\bigcup n.\ ((SPower\ (X \cap SMore)\ n))){\cdot}(SEmpty \cup X \cap SMore \cap SInf)) \cdot Y$
by (*simp add*: *sstar-def spowerstar-def sfpowerstar-def*)
have *2*: $((\bigcup n.\ ((SPower\ (X \cap SMore)\ n))){\cdot}(SEmpty \cup X \cap SMore \cap SInf)) \cdot Y =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n))){\cdot}((SEmpty \cup X \cap SMore \cap SInf) \cdot Y)$
using *FusionAssoc* **by** *blast*
have *3*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n))){\cdot}((SEmpty \cup X \cap SMore \cap SInf) \cdot Y) =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)){\cdot}((SEmpty \cup X \cap SMore \cap SInf) \cdot Y))$
using *UnionFusionSPower1*[*of X* $\cap$ *SMore* ((*SEmpty* $\cup$ *X* $\cap$ *SMore* $\cap$ *SInf*) $\cdot$ *Y*)]
by *auto*

have *4*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot\ Y)) =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup X \cap SMore \cap SInf)) \cdot Y)$
using *FusionAssoc* **by** *blast*
show *?thesis*
by (*simp add*: *1 2 3 4*)
qed

lemma *FPowerstarInductL*:
assumes $Y \cup (X \cap SFinite) \cdot Z \subseteq Z$
shows $(SFPowerstar\ X) \cdot\ Y \subseteq Z$
proof –
have *1*: $(SFPowerstar\ X) \cdot\ Y = (\bigcup n.\ (SPower\ X\ n)) \cdot Y$
by (*simp add*: *sfpowerstar-def*)
have *2*: $(\bigcup n.\ (SPower\ X\ n)) \cdot Y = (\bigcup n.\ (SPower\ X\ n) \cdot Y)$
using *UnionFusionSPower1* **by** *fastforce*
have *3*: $\bigwedge n.\ (SPower\ X\ n) \cdot Y \subseteq Z$
using *assms fusion-inductl-finite* **by** *blast*
have *4*: $(\bigcup n.\ (SPower\ X\ n) \cdot Y) \subseteq Z$
using *3* **by** *blast*
show *?thesis*
using *1 2 4* **by** *blast*
qed

lemma *FPowerstarInductMoreL*:
assumes $Y \cup ((X \cap SMore) \cap SFinite) \cdot Z \subseteq Z$
shows $(SFPowerstar\ X) \cdot\ Y \subseteq Z$
proof –
have *1*: $(SFPowerstar\ X) \cdot\ Y = (\bigcup n.\ (SPower\ X\ n)) \cdot Y$
by (*simp add*: *sfpowerstar-def*)
have *2*: $(\bigcup n.\ (SPower\ X\ n)) \cdot Y = (\bigcup n.\ (SPower\ X\ n) \cdot Y)$
using *UnionFusionSPower1* **by** *fastforce*
have *3*: $\bigwedge n.\ (SPower\ X\ n) \cdot Y \subseteq Z$
by (*metis Int-assoc Int-commute assms fusion-inductl-fmore sfmore-def*)
have *4*: $(\bigcup n.\ (SPower\ X\ n) \cdot Y) \subseteq Z$
using *3* **by** *blast*
show *?thesis*
using *1 2 4* **by** *blast*
qed

lemma *PowerstarInductL*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $(SPowerstar\ X) \cdot Y \subseteq Z$
proof –
have *1*: $(SPowerstar\ X) \cdot\ Y = ((\bigcup n.\ (SPower\ X\ n)) \cdot (SEmpty \cup X \cap SInf))\ \cdot Y$
by (*simp add*: *spowerstar-def sfpowerstar-def*)
have *2*: $((\bigcup n.\ (SPower\ X\ n)) \cdot (SEmpty \cup X \cap SInf))\ \cdot Y =$
$(\bigcup n.\ ((SPower\ X\ n))) \cdot ((SEmpty \cup X \cap SInf)\ \cdot Y)$
using *FusionAssoc1*[*of* $(\bigcup n.\ ((SPower\ X\ n)))\ (SEmpty \cup X \cap SInf)\ Y$]
by *blast*
have *3*: $(\bigcup n.\ ((SPower\ X\ n))) \cdot ((SEmpty \cup X \cap SInf)\ \cdot Y) =$

$(\bigcup n.\ ((SPower\ X\ n))\cdot((SEmpty \cup X \cap SInf)\ \cdot Y))$
using *UnionFusionSPower1*[*of* X $((SEmpty \cup X \cap SInf)\ \cdot Y)$]
by *auto*
have *4*: $\bigwedge n.\ (SPower\ X\ n)\cdot Y \subseteq Z$
using *assms fusion-inductl* **by** *blast*
have *5*: $\bigwedge n.\ ((SPower\ X\ n)\cdot(X \cap SInf))\cdot Y \subseteq Z$
using *assms fusion-inductl-inf* **by** *blast*
have *6*: $\bigwedge n.\ (SPower\ X\ n)\cdot Y \cup ((SPower\ X\ n)\cdot(X \cap SInf))\cdot Y =$
$((SPower\ X\ n)\cdot(SEmpty \cup (X \cap SInf)))\cdot Y$
by (*simp add*: *FusionSEmptyR FusionUnionDistL FusionUnionDistR*)
have *7*: $\bigwedge n.\ ((SPower\ X\ n)\cdot(SEmpty \cup (X \cap SInf)))\cdot Y \subseteq Z$
using *4 5 6* **by** *blast*
have *8*: $(\bigcup n.\ ((SPower\ X\ n)\cdot(SEmpty \cup (X \cap SInf)))\ \cdot Y) \subseteq Z$
using *7* **by** *blast*
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 3 8 FusionAssoc Sup.SUP-cong*)
qed

lemma *SStarInductMoreL*:
assumes $Y \cup (X \cap SMore)\cdot Z \subseteq Z$
shows $(SStar\ X)\cdot Y \subseteq Z$
proof −
have *1*: $(SStar\ X)\cdot\ Y = ((\bigcup n.\ (SPower\ (X \cap SMore)\ n))\cdot(SEmpty \cup (X \cap SMore) \cap SInf))\ \cdot Y$
by (*simp add*: *sstar-def spowerstar-def sfpowerstar-def*)
have *2*: $((\bigcup n.\ (SPower\ (X \cap SMore)\ \ n))\cdot(SEmpty \cup (X \cap SMore) \cap SInf))\ \cdot Y =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)))\cdot((SEmpty \cup (X \cap SMore) \cap SInf)\ \cdot Y)$
using *FusionAssoc1*[*of* $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)))$ $(SEmpty \cup (X \cap SMore) \cap SInf)$ Y]
by *blast*
have *3*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)))\cdot((SEmpty \cup (X \cap SMore) \cap SInf)\ \cdot Y) =$
$(\bigcup n.\ ((SPower\ (X \cap SMore)\ n))\cdot((SEmpty \cup (X \cap SMore) \cap SInf)\ \cdot Y))$
using *UnionFusionSPower1*[*of* $X \cap SMore$ $((SEmpty \cup (X \cap SMore) \cap SInf)\ \cdot Y)$]
by *auto*
have *4*: $\bigwedge n.\ (SPower\ (X \cap SMore)\ n)\cdot Y \subseteq Z$
using *assms fusion-inductl* **by** *blast*
have *5*: $\bigwedge n.\ ((SPower\ (X \cap SMore)\ n)\cdot((X \cap SMore) \cap SInf))\cdot Y \subseteq Z$
using *assms fusion-inductl-inf* **by** *blast*
have *6*: $\bigwedge n.\ (SPower\ (X \cap SMore)\ n)\cdot Y \cup ((SPower\ (X \cap SMore)\ n)\cdot((X \cap SMore) \cap SInf))\cdot Y =$
$((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)))\cdot Y$
by (*simp add*: *FusionSEmptyR FusionUnionDistL FusionUnionDistR*)
have *7*: $\bigwedge n.\ ((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup (\ (X\cap SMore) \cap SInf)))\cdot Y \subseteq Z$
using *4 5 6* **by** *blast*
have *8*: $(\bigcup n.\ ((SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)))\ \cdot Y) \subseteq Z$
using *7* **by** *blast*
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 3 8 FusionAssoc Sup.SUP-cong*)
qed

lemma *SFPowerstarInductR*:
assumes $Y \cup Z\cdot X \subseteq Z$
shows $Y\cdot(SFPowerstar\ X) \subseteq Z$

proof –
have *1*: $Y \cdot (SFPowerstar\ X) = Y \cdot (\bigcup n.\ (SPower\ X\ n))$
by (*simp add*: *sfpowerstar-def*)
have *2*: $Y \cdot (\bigcup n.\ (SPower\ X\ n)) = (\bigcup n.\ Y \cdot (SPower\ X\ n))$
using *FusionUnionSPower1* **by** *blast*
have *3*: $\bigwedge n.\ Y \cdot (SPower\ X\ n) \subseteq Z$
using *assms fusion-inductr* **by** *blast*
have *4*: $(\bigcup n.\ Y \cdot (SPower\ X\ n)) \subseteq Z$
using *3* **by** *blast*
show *?thesis*
by (*simp add*: *1 2 4*)
qed

lemma *SPowerstarInductR*:
assumes $Y \cup Z \cdot X \subseteq Z$
shows $Y \cdot (SPowerstar\ X) \subseteq Z$
proof –
have *1*: $Y \cdot (SPowerstar\ X) = Y \cdot ((\bigcup n.\ (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf)))$
by (*simp add*: *spowerstar-def sfpowerstar-def*)
have *11*: $Y \cdot ((\bigcup n.\ (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf))) =$
$Y \cdot ((\bigcup n.\ (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf))))$
by (*metis UnionFusionSPower1*)
have *12*: $Y \cdot ((\bigcup n.\ (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf)))) =$
$((\bigcup n.\ Y \cdot (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf))))$
using *FusionUnionSPower*[*of Y X* (*SEmpty* ∪ *X* ∩ *SInf*)]
by (*metis* (*no-types*, *lifting*) *FusionAssoc Sup.SUP-cong*)
have *3*: $\bigwedge n.\ Y \cdot (SPower\ X\ n) \subseteq Z$
using *assms fusion-inductr* **by** *blast*
have *31*: $\bigwedge n.\ Y \cdot (SPower\ X\ n) \cdot (X \cap SInf) \subseteq Z$
using *FusionAssoc assms fusion-inductr-inf* **by** *blast*
have *32*: $\bigwedge n.\ Y \cdot (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf)) \subseteq Z$
by (*simp add*: *3 31 FusionSEmptyR FusionUnionDistR*)
have *4*: $(\bigcup n.\ Y \cdot (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf))) \subseteq Z$
using *32* **by** *blast*
show *?thesis*
by (*simp add*: *1 11 12 4*)
qed

lemma *SStarInductMoreR*:
assumes $Y \cup Z \cdot (X \cap SMore) \subseteq Z$
shows $Y \cdot (SStar\ X) \subseteq Z$
proof –
have *1*: $Y \cdot (SStar\ X) = Y \cdot ((\bigcup n.\ (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)))$
by (*simp add*: *spowerstar-def sfpowerstar-def sstar-def*)
have *11*: $Y \cdot ((\bigcup n.\ (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) =$
$Y \cdot ((\bigcup n.\ (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))))$
by (*metis UnionFusionSPower1*)
have *12*: $Y \cdot ((\bigcup n.\ (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)))) =$
$((\bigcup n.\ Y \cdot (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))))$
using *FusionUnionSPower*[*of Y* (*X* ∩ *SMore*) (*SEmpty* ∪ ((*X* ∩ *SMore*) ∩ *SInf*))]

by (*metis* (*no-types*, *lifting*) *FusionAssoc Sup.SUP-cong*)
have *3*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n) \subseteq Z$
using *assms fusion-inductr* **by** *blast*
have *31*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot((X \cap SMore) \cap SInf) \subseteq Z$
using *assms FusionAssoc fusion-inductr-inf* **by** *blast*
have *32*: $\bigwedge n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf)) \subseteq Z$
by (*simp add*: *3 31 FusionSEmptyR FusionUnionDistR*)
have *4*: $(\bigcup n.\ Y\cdot(SPower\ (X \cap SMore)\ n)\cdot(SEmpty \cup ((X \cap SMore) \cap SInf))) \subseteq Z$
using *32* **by** *blast*
show *?thesis*
by (*simp add*: *1 11 12 4*)
qed

lemma *Powerstarhelp2*:
$(X \cap SInf) = (X \cap SInf)\cdot\ Z$
by (*metis FusionAssoc FusionSFalse SFalseFusion*)

lemma *Powerstarhelp3*:
$((X \cap SInf)\cdot Y \cup (X \cap SFinite)\cdot Y) = X\cdot Y$
by (*metis Diff-Compl FusionUnionDistL Un-Diff-Int sfinite-def*)

lemma *Powerstareqv*:
$(SPowerstar\ X) = SEmpty \cup X\cdot(SPowerstar\ X)$
proof −
have *1*: $(SPowerstar\ X) = (\bigcup n.\ (SPower\ X\ n))\cdot(SEmpty \cup (X \cap SInf))$
by (*simp add*: *sfpowerstar-def spowerstar-def*)
have *2*: $(\bigcup n.\ (SPower\ X\ n))\cdot(SEmpty \cup (X \cap SInf)) =$
$(SEmpty \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n)))\cdot\ (SEmpty \cup (X \cap SInf))$
by (*metis sfpowerstar-eqv-1*)
have *3*: $(SEmpty \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n)))\cdot\ (SEmpty \cup (X \cap SInf)) =$
$SEmpty\cdot(SEmpty \cup (X \cap SInf)) \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf))$
by (*simp add*: *FusionUnionDistL*)
have *4*: $SEmpty\cdot(SEmpty \cup (X \cap SInf)) \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf))$
$=$
$SEmpty \cup (X \cap SInf) \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf))$
by (*simp add*: *FusionSEmptyL*)
have *5*: $SEmpty \cup (X \cap SInf) \cup (X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)) =$
$SEmpty \cup (X \cap SInf)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)) \cup$
$(X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf))$
by (*metis Powerstarhelp2*)
have *6*: $SEmpty \cup (X \cap SInf)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)) \cup$
$(X \cap SFinite)\cdot(\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)) =$
$SEmpty \cup X\ \cdot((\bigcup n.\ (SPower\ X\ n))\cdot\ (SEmpty \cup (X \cap SInf)))$
by (*metis FusionAssoc Powerstarhelp3 UnionAssoc*)
show *?thesis*
using *1 2 3 4 5 6* **by** *presburger*
qed

lemma *SStareqv*:
$SStar\ X = SEmpty \cup (X \cap SMore)\cdot(SStar\ X)$

by (*metis Powerstareqv sstar-def*)

lemma *SSkipSFinite*:
SSkip ∩ *SFinite* = *SSkip*
proof −
have *1*: *SSkip* ∩ *SFinite* ⊆ *SSkip*
by *simp*
have *2*: *SInf* ⊆ *SEmpty* ∪ (− *SEmpty* · − *SEmpty*)
by (*metis Diff-Compl Diff-subset-conv Powerstarhelp2 Powerstarhelp3 double-compl inf-commute sup-ge1*)

have *3*: *SSkip* ⊆ *SSkip* ∩ *SFinite*
using *2* **by** (*simp add*: *sskip-def smore-def sfinite-def*) *blast*
show *?thesis*
using *3* **by** *blast*
qed

lemma *spower-sskip-elim* :
(σ ∈ *SPower SSkip n*) ⟷
(*nlength* σ = *n*)
proof (*induct n arbitrary*: σ)
case *0*
then show *?case*
by (*simp add*: *sempty-elim zero-enat-def*)
next
case (*Suc n*)
then show *?case*
proof −
have *1*: (σ ∈ *SPower SSkip* (*Suc n*)) ⟷ σ ∈ (*SFinite* ∩ *SSkip*)· (*SPower SSkip n*)
by (*simp add*: *inf-commute*)
have *2*: σ ∈ (*SFinite* ∩ *SSkip*) ⟷ σ ∈ *SSkip*
using *SSkipSFinite* **by** *auto*
have *3*: σ ∈ (*SFinite* ∩ *SSkip*)· (*SPower SSkip n*) ⟷
(∃ σ*1* σ*2*. σ = *nfuse* σ*1* σ*2* ∧ *nfinite* σ*1* ∧ σ*1* ∈ *SSkip* ∧ σ*2* ∈ *SPower SSkip n* ∧ *nlast* σ*1* = *nfirst* σ*2*)
using *ffusion-iff*[*of* σ *SSkip SPower SSkip n*] **by** (*simp add*: *inf-commute*)
have *4*: (∃ σ*1* σ*2*. σ = *nfuse* σ*1* σ*2* ∧ *nfinite* σ*1* ∧ σ*1* ∈ *SSkip* ∧ σ*2* ∈ *SPower SSkip n* ∧ *nlast* σ*1* = *nfirst* σ*2*) ⟷
(*nlength* σ = *enat* (*Suc n*))
by (*auto simp add*: *Suc.hyps nfuse-nlength one-enat-def sskip-elim*)
(*metis One-nat-def add.commute eSuc-enat enat.simps*(*3*) *enat-add-sub-same enat-le-plus-same*(*2*)
min.orderE ndropn-nfirst ndropn-nlength nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast
ntaken-nlength one-enat-def plus-1-eSuc(*2*))
show *?thesis*
using *1 3 4* **by** *presburger*
qed
qed

11.3.6 Kleene Algebra

lemma *UnfoldL*:

SEmpty ∪ *X*·(*SStar X*) = (*SStar X*)
proof –
have *1*: (*SStar X*) = *SEmpty* ∪ (*X* ∩ *SMore*)·(*SStar X*)
by (*meson Un-iff set-eqI SStareqv*)
have *2*: (*X* ∩ *SMore*)·(*SStar X*) ⊆ *X*·(*SStar X*)
by (*simp add*: *FusionRuleL*)
have *3*: (*SStar X*) ⊆ *SEmpty* ∪ *X*·(*SStar X*)
using *1 2* **by** *blast*
have *4*: *SEmpty* ⊆ (*SStar X*)
using *1* **by** *auto*
have *5*: *X* ⊆ *SEmpty* ∪ (*X* ∩ *SMore*)
by (*simp add*: *Un-Int-distrib smore-def*)
have *6*: *X*·(*SStar X*) ⊆ (*SStar X*) ∪ (*X* ∩ *SMore*)·(*SStar X*)
using *5* **by** (*metis FusionRuleL FusionUnionDistL FusionSEmptyL*)
have *7*: (*SStar X*) ⊆ *SEmpty* ∪ (*X* ∩ *SMore*)·(*SStar X*)
using *1* **by** *auto*
have *8*: *X*·(*SStar X*) ⊆ *SEmpty* ∪ (*X* ∩ *SMore*)·(*SStar X*)
using *6 7* **by** *blast*
hence *9*: *X*·(*SStar X*) ⊆ (*SStar X*)
using *1* **by** *auto*
have *10*: *SEmpty* ∪ *X*·(*SStar X*) ⊆ (*SStar X*)
using *9 4* **by** *simp*
from *3 10* **show** *?thesis* **by** *auto*
qed

— Left induction

lemma *SStarInductL*:
assumes *Y* ∪ *X*·*Z* ⊆ *Z*
shows (*SStar X*)·*Y* ⊆ *Z*
proof –
have *1*: (*SStar X*)· *Y* = ((⋃*n*. (*SPower* (*X* ∩ *SMore*) *n*))·(*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*)) ·*Y*
by (*simp add*: *sstar-def spowerstar-def sfpowerstar-def*)
have *2*: ((⋃*n*. (*SPower* (*X* ∩ *SMore*) *n*))·(*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*)) ·*Y* =
(⋃*n*. ((*SPower* (*X* ∩ *SMore*) *n*)))·((*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) ·*Y*)
using *FusionAssoc1*[*of* (⋃*n*. ((*SPower* (*X* ∩ *SMore*) *n*))) (*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) *Y*]
by *blast*
have *3*: (⋃*n*. ((*SPower* (*X* ∩ *SMore*) *n*)))·((*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) ·*Y*) =
(⋃*n*. ((*SPower* (*X* ∩ *SMore*) *n*))·((*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) ·*Y*))
using *UnionFusionSPower1*[*of X* ∩ *SMore* ((*SEmpty* ∪ (*X* ∩ *SMore*) ∩ *SInf*) ·*Y*)]
by *auto*
have *31*: *Y* ∪ (*X* ∩ *SMore*)· *Z* ⊆ *Z*
by (*meson FusionRuleL Un-mono assms dual-order.trans inf.cobounded1 subset-refl*)
have *4*: ⋀*n*. (*SPower* (*X* ∩ *SMore*) *n*)·*Y* ⊆ *Z*
using *assms fusion-inductl-more* **by** *blast*
have *5*: ⋀*n*. ((*SPower* (*X* ∩ *SMore*) *n*)·((*X* ∩ *SMore*) ∩ *SInf*))·*Y* ⊆ *Z*
using *31 fusion-inductl-inf* **by** *blast*
have *6*: ⋀*n*. (*SPower* (*X* ∩ *SMore*) *n*)·*Y* ∪ ((*SPower* (*X* ∩ *SMore*) *n*)·((*X* ∩ *SMore*) ∩ *SInf*))·*Y* =
((*SPower* (*X* ∩ *SMore*) *n*)·(*SEmpty* ∪ ((*X* ∩ *SMore*) ∩ *SInf*)))·*Y*

by (*simp add*: *FusionSEmptyR FusionUnionDistL FusionUnionDistR*)
have *7*: $\bigwedge n.$ ((*SPower* ($X \cap$ *SMore*) *n*)·(*SEmpty* $\cup$ (($X\cap$ *SMore*) $\cap$ *SInf*)))·$Y \subseteq Z$
using *4 5 6* **by** *blast*
have *8*: ($\bigcup n.$ ((*SPower* ($X \cap$ *SMore*) *n*)·(*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*))) ·Y) $\subseteq Z$
using *7* **by** *blast*
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 3 8 FusionAssoc Sup.SUP-cong*)
qed

— Right induction
lemma *SStarInductR*:
assumes $Y \cup Z{\cdot}X \subseteq Z$
shows $Y{\cdot}$(*SStar* X) $\subseteq Z$
proof –
have *1*: $Y{\cdot}$(*SStar* X) = $Y{\cdot}$(($\bigcup n.$ (*SPower* ($X \cap$ *SMore*) *n*))· (*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*)))
by (*simp add*: *spowerstar-def sfpowerstar-def sstar-def*)
have *11*: $Y{\cdot}$(($\bigcup n.$ (*SPower* ($X \cap$ *SMore*) *n*))· (*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*))) =
$Y{\cdot}$(($\bigcup n.$ (*SPower* ($X \cap$ *SMore*) *n*)· (*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*))))
by (*metis UnionFusionSPower1*)
have *12*: $Y{\cdot}$(($\bigcup n.$ (*SPower* ($X \cap$ *SMore*) *n*)· (*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*)))) =
(($\bigcup n.$ $Y{\cdot}$(*SPower* ($X \cap$ *SMore*) *n*)· (*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*))))
using *FusionUnionSPower*[*of Y* ($X \cap$ *SMore*) (*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*))]
by (*metis* (*no-types*, *lifting*) *FusionAssoc Sup.SUP-cong*)
have *21*: $Y \cup Z{\cdot}(X \cap$ *SMore*) $\subseteq Z$
by (*meson FusionRuleR Un-mono assms dual-order.trans inf.cobounded1 subset-refl*)
have *3*: $\bigwedge n.$ $Y{\cdot}$(*SPower* ($X \cap$ *SMore*) *n*) $\subseteq Z$
using *21 fusion-inductr* **by** *blast*
have *31*: $\bigwedge n.$ $Y{\cdot}$(*SPower* ($X \cap$ *SMore*) *n*)·(($X \cap$ *SMore*) $\cap$ *SInf*) $\subseteq Z$
using *21 FusionAssoc fusion-inductr-inf* **by** *blast*
have *32*: $\bigwedge n.$ $Y{\cdot}$(*SPower* ($X \cap$ *SMore*) *n*)·(*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*)) $\subseteq Z$
by (*simp add*: *3 31 FusionSEmptyR FusionUnionDistR*)
have *4*: ($\bigcup n.$ $Y{\cdot}$(*SPower* ($X \cap$ *SMore*) *n*)·(*SEmpty* $\cup$ (($X \cap$ *SMore*) $\cap$ *SInf*))) $\subseteq Z$
using *32* **by** *blast*
show *?thesis*
by (*simp add*: *1 11 12 4*)
qed

11.3.7 Omega Algebra

lemma *SFMoresem* [*mono*]:
$x \in$ (((($F \cap$ *SMore*) $\cap$ *SFinite*)·G)) $\longleftrightarrow$
(($\exists \sigma 1\ \sigma 2.$ x = *nfuse* $\sigma 1\ \sigma 2$ $\wedge$ *nfinite* $\sigma 1$ $\wedge$ $\sigma 1 \in F$ $\wedge$ $0<$(*nlength* $\sigma 1$) $\wedge$ $\sigma 2 \in G$ $\wedge$ *nlast* $\sigma 1$ = *nfirst* $\sigma 2$))

by (*simp add*: *fusion-iff sinfinite-elim smore-elim sfinite-elim*) *blast*

lemma *monoomega*[*mono*]:
mono ($\lambda p\ x.$ $x \in$ (($F \cap$ *SMore*) $\cap$ *SFinite* · p))
by (*auto simp add*: *mono-def sinfinite-elim fusion-iff sfmore-elim*)

coinductive-set *somega* :: *'a iintervals* ⇒ *'a iintervals*
  for *F* **where**
  $((\exists \sigma 1\ \sigma 2.\ x{=}nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in F \wedge 0{<}(nlength\ \sigma 1) \wedge$
  $\sigma 2 \in (somega\ F) \wedge nlast\ \sigma 1 = nfirst\ \sigma 2))$
  $\Longrightarrow x \in (somega\ F)$

lemma *SOmegaIntros*:
  $(((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq (somega\ F)$
using *somega.intros*[*of* - *F*] *SFMoresem*[*of* - *F* (*somega F*)]
by (*meson subsetI*)

lemma *SOmegaCases*:
assumes $x \in somega\ F$
  $x \in (\ (((F \cap SMore) \cap SFinite) \cdot (somega\ F))) \Longrightarrow P$
shows *P*
using *assms somega.cases*[*of x F P*] *SFMoresem* **by** *blast*

lemma *SOmegaUnrollsem*:
  $x \in (somega\ F) \longleftrightarrow x \in (((F \cap SMore) \cap SFinite) \cdot (somega\ F))$
proof *auto*
  show $x \in somega\ F \Longrightarrow x \in (F \cap SMore) \cap SFinite \cdot somega\ F$
    using *SOmegaCases* **by** *blast*
  show $x \in (F \cap SMore) \cap SFinite \cdot somega\ F \Longrightarrow x \in somega\ F$
   by (*metis* (*no-types*, *lifting*) *SOmegaIntros inf-commute subset-eq*)
qed

lemma *SOmegaUnroll*:
  $(somega\ F) = (((F \cap SMore) \cap SFinite) \cdot (somega\ F))$
using *SOmegaUnrollsem* **by** *blast*

lemma *SOmegaCoinductsem*:
assumes $\bigwedge x.\ x \in X \Longrightarrow x \in (\ (((F \cap SMore) \cap SFinite) \cdot (X \cup (somega\ F))))$
shows $x \in X \Longrightarrow x \in (somega\ F)$
using *assms somega.coinduct*[*of* $\lambda x.\ x \in X$ *x F*] *SFMoresem*[*of* - *F* ($X \cup$ *somega F*)] **by** *auto*

lemma *SOmegaCoinduct*:
assumes $X \subseteq (\ (((F \cap SMore) \cap SFinite) \cdot (X \cup (somega\ F))))$
shows $X \subseteq (somega\ F)$
using *assms SOmegaCoinductsem*[*of X F*] **by** *blast*

lemma *SOmegaWeakCoinductsem*:
assumes $\bigwedge x.\ x \in X \Longrightarrow x \in (\ (((F \cap SMore) \cap SFinite) \cdot X\ ))$
shows $x \in X \Longrightarrow x \in (somega\ F)$
using *assms somega.coinduct*[*of* $\lambda x.\ x \in X$ *x F*]
by (*metis SFMoresem*)

lemma *SOmegaWeakCoinduct*:
assumes $X \subseteq (\ (((F \cap SMore) \cap SFinite) \cdot X\ ))$
shows $X \subseteq (somega\ F)$
using *assms SOmegaWeakCoinductsem*[*of X F*] **by** *blast*

11.3.8 ITL specific Laws

lemma *PwrFusionInterLsem*:
$x \in ((((SPower\ SSkip\ n) \cap X) \cdot V) \cap (((SPower\ SSkip\ n) \cap Y) \cdot W)) \longleftrightarrow$
$x \in (((SPower\ SSkip\ n) \cap X \cap Y) \cdot (V \cap W))$
by (*auto simp add*: *spower-sskip-elim fusion-iff-1 min-absorb1 nlength-eq-enat-nfiniteD*)

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 *PwrFusionInterLsem* **by** *blast*

lemma *PwrFusionInterRsem* :
$x \in ((V \cdot ((SPower\ SSkip\ n) \cap X)) \cap ((W \cdot ((SPower\ SSkip\ n) \cap Y)))) \longleftrightarrow$
$x \in ((V \cap W) \cdot ((SPower\ SSkip\ n) \cap X \cap Y))$
by (*simp add*: *spower-sskip-elim fusion-iff-1*)
(*rule*,
metis enat.simps(*3*) *enat-add2-eq enat-add-sub-same le-iff-add ndropn-nlength nfinite-ndropn nlength-eq-enat-nfiniteD the-enat.simps*,
metis enat.simps(*3*) *enat-add-sub-same le-iff-add ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD*)

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 *PwrFusionInterRsem* **by** *blast*

lemma *SSkipFusionImpSMore*:
$SSkip \cdot STrue \subseteq SMore$
using *subsetI*[*of SSkip·STrue SMore*]
by (*auto simp add*: *fusion-iff-1 sskip-elim smore-elim strue-elim*)

lemma *SMoreImpSSkipFusion*:
$SMore \subseteq SSkip \cdot STrue$
using *subsetI*[*of SMore SSkip·STrue*]
by (*auto simp add*: *fusion-iff-1 sskip-elim smore-elim strue-elim*)
(*metis i0-less ileI1 one-eSuc one-enat-def*,
metis i0-less ileI1 one-eSuc one-enat-def)

lemma *SSkipSMore*:
$SSkip \cap SMore = SSkip$
by (*simp add*: *inf.absorb1 smore-def sskip-def*)

lemma *SPowerSkip*:
$(\bigcup n.\ (SPower\ SSkip\ n)) = SFinite$
proof −
have *1*: $(\bigcup n.\ (SPower\ SSkip\ n)) \subseteq SFinite$
by (*simp add*: *SUP-least spower-finite*)
have *2*: $\bigwedge x.\ \ x \in SFinite \Longrightarrow x \in (\bigcup n.\ (SPower\ SSkip\ n))$
by (*auto simp add*: *sfinite-elim spower-sskip-elim nfinite-nlength-enat*)
have *3*: $SFinite \subseteq (\bigcup n.\ (SPower\ SSkip\ n))$

using *2* **by** *blast*
from *1 3* **show** *?thesis* **by** *blast*
qed

lemma *SStarSkip*:
$(SStar\ SSkip) = SFinite$
by (*simp add*: *sstar-def SSkipSMore spowerstar-def sfpowerstar-def SPowerSkip*)
(*metis Compl-disjoint2 FusionSEmptyR SSkipSFinite UnionSFalse inf.assoc inf-bot-right sfalse-def sfinite-def*)

11.4 Derived Laws

11.4.1 Helper Lemmas

lemma *B01*:
assumes $(X{::}\ 'a\ iintervals) \subseteq Y$
shows $-Y \subseteq -X$
using *assms* **by** *auto*

lemma *B04*:
$((X{::}\ 'a\ iintervals) = Y) \longleftrightarrow (X \subseteq Y) \wedge (Y \subseteq X)$
by *auto*

lemma *B09*:
assumes $-X \cup Y = STrue$
shows $(X{::}\ 'a\ iintervals) \subseteq Y$
using *assms* **using** *strue-def* **by** *auto*

lemma *B20*:
$(X{::}\ 'a\ iintervals) \subseteq Y \cup Z \longleftrightarrow X \cap -Y \subseteq Z$
by *auto*

lemma *B28*:
$((X{::}\ 'a\ iintervals) \cap Y) \cup (X \cap Z) = X \cap (Y \cup Z)$
by *auto*

lemma *CH01*:
$STrue{\cdot}STrue = STrue$
by (*metis FusionSEmptyR FusionUnionDistR Int-commute STrueTop 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 SSkipSFinite*)

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 SSkipSFinite*)

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*:
$(SStar\ (X \cap SEmpty)) = SEmpty$
using *SStareqv*[*of* $(X \cap SEmpty)$]
by (*metis Compl-disjoint2 Powerstarhelp2 SFalseBottom UnionSFalse inf-assoc inf-bot-right sfalse-def smore-def*)

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*:
$(SInit\ (-X)) \subseteq -(SInit\ X)$
unfolding *sinit-def*
by (*metis SFalseBottom SFalseFusion ST24 disjoint-eq-subset-Compl inf.commute inf-compl-bot-left2 sinit-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*:
$(SStar\ X) \subseteq (SStar\ (SStar\ X))$
proof –
have $SStar\ X \cdot SEmpty \cup SStar\ X \cdot (SStar\ X \cdot SStar\ SStar\ X) = SStar\ X \cdot SStar\ SStar\ X$
by (*metis* (*full-types*) *FusionUnionDistR UnfoldL*)
then have $SStar\ X \cdot SEmpty \cup SStar\ X \cdot (SStar\ X \cdot SStar\ SStar\ X) \subseteq SStar\ SStar\ X$
using *UnfoldL* **by** *auto*
then show *?thesis*
by (*simp add*: *FusionSEmptyR*)
qed

lemma *SStar08*:
$(SStar\ X) = (SStar\ (SStar\ X))$

by (*meson B04 SStar05 SStar06 SStar07*)

lemma *SStar15*:
  SEmpty ⊆ (*SStar SSkip*)
using *UnfoldL* **by** *fastforce*

lemma *SStar16*:
  SSkip ⊆ (*SStar SSkip*)
using *SSkipSFinite SStarSkip* **by** *blast*

lemma *SStar22*:
  (*SEmpty* ∩ *X*)·(*SStar* (*SEmpty* ∩ *X*)) = (*SEmpty* ∩ *X*)
by (*metis ST15 FusionSEmptyR inf-commute*)

lemma *SStar23*:
  (*SStar* (*SEmpty* ∩ *X*)) = *SEmpty*
using *SStar22 UnfoldL* **by** *auto*

lemma *SStar25*:
  (*SStar STrue*) = *STrue*
by (*metis FusionRuleR FusionSEmptyR Un-absorb2 UnfoldL UnionCommute compl-bot-eq strue-def sup.right-idem sup-ge2 sup-top-right*)

lemma *SStar28*:
  (*SStar X*)·*X* ⊆ *X*·(*SStar X*)
by (*metis B04 FusionUnionDistR FusionSEmptyR UnfoldL SStarInductL*)

lemma *SStar29*:
  X·(*SStar X*) ⊆ (*SStar X*)·*X*
by (*metis B04 SStar28 SStarInductR UnfoldL FusionRuleL ST47 sup.mono*)

lemma *SStar17*:
  (*SStar SSkip*)·*SSkip* ⊆ *SSkip*·(*SStar SSkip*)
by (*simp add*: *SStar28*)

lemma *SStar18*:
  SSkip·(*SStar SSkip*) ⊆ (*SStar SSkip*)·*SSkip*
by (*simp add*: *SStar29*)

lemma *SStar19*:
  (*SStar SSkip*)·*SSkip* = *SSkip*·(*SStar SSkip*)
using *SStar17 SStar18* **by** *auto*

lemma *SStar30*:
  X·(*SStar X*) = (*SStar X*)·*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*)
 ⊆ (−*X*) ∪ (*SStar SSkip*)·(*X* ∩ (*SSkip*·(−*X*)))
using *FusionUnionDistR*[*of SSkip*] *UnfoldL*[*of SSkip*]
proof −
have *f1*: − *X* ∪ (*SEmpty* ∪ *SSkip* · *SStar SSkip*) · (*X* ∩ (*SSkip* · − *X*)) ⊆
 − *X* ∪ *SStar SSkip* · (*X* ∩ (*SSkip* · − *X*))

by (*simp add*: ‹$SEmpty \cup SSkip \cdot SStar\ SSkip = SStar\ SSkip$›)
have *f2*: $SSkip \cdot -\ X \subseteq -\ X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -\ X))$
by (*metis B20 Morgan ST47 UnionIdem* ‹$SEmpty \cup SSkip \cdot SStar\ SSkip = SStar\ SSkip$›
inf-commute inf-idem sup.cobounded1)
have $SSkip \cdot (SStar\ SSkip \cdot (X \cap (SSkip \cdot -\ X))) \subseteq -\ X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -\ X))$
using *f1* **by** (*metis* (*no-types*) *FusionAssoc ST47 Un-subset-iff*)
then show *?thesis*
using *f2* **by** (*simp add*: ‹$\bigwedge Z\ Y.\ SSkip \cdot (Y \cup Z) = SSkip \cdot Y \cup SSkip \cdot Z$›)
qed

lemma *SStar52*:
$(SStar\ X) \subseteq SEmpty \cup (X \cap SMore)\cdot(SStar\ X)$
by (*metis B04 SStar47 UnfoldL*)

lemma *SStar53*:
$SEmpty \cup (X \cap SMore)\cdot(SStar\ X) \subseteq (SStar\ X)$
by (*metis SStar12 SStar47*)

lemma *BD45*:
$(SBi\ ((-X) \cup X1)) \cap (X\cdot Y) \subseteq X1\cdot Y$
proof −
have *1*: $(SBi\ ((-X) \cup X1)) = -(-((-X) \cup X1)\cdot(Y \cup (-Y)))$
by (*metis sbi-def sdi-def STrueTop*)
have *2*: $-(-((-X) \cup X1)\cdot(Y \cup (-Y))) \cap (X\cdot Y) \subseteq$
$-(-((-X) \cup X1)\cdot(Y)) \cap (X\cdot Y)$
using *FusionUnionDistR* **by** *fastforce*
have *3*: $-(-((-X) \cup X1)\cdot(Y)) \cap (X\cdot Y) \subseteq\ (((-X) \cup X1) \cap X)\cdot Y$
by (*metis* (*no-types*, *opaque-lifting*) *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*: $(SAlways\ ((-Y) \cup Y1)) = -((SFinite \cap (X1 \cup -X1))\cdot (-((-Y) \cup Y1)))$
by (*simp add*: *salways-def ssometime-def*)
have *2*: $-((SFinite \cap (X1 \cup -X1))\cdot(-((-Y) \cup Y1))) =$
$-(((SFinite \cap X1) \cup (SFinite \cap -X1))\cdot(-((-Y) \cup Y1)))$
by (*simp add*: *B28*)
have *3*: $-(((SFinite \cap X1) \cup (SFinite \cap -X1))\cdot(-((-Y) \cup Y1))) =$
$-((SFinite \cap X1)\cdot(-((-Y) \cup Y1)) \cup (SFinite \cap -X1)\cdot(-((-Y) \cup Y1)))$
by (*simp add*: *FusionUnionDistL*)
have *4*: $-((SFinite \cap X1)\cdot(-((-Y) \cup Y1)) \cup (SFinite \cap -X1)\cdot(-((-Y) \cup Y1))) =$
$-((SFinite \cap X1)\cdot(-((-Y) \cup Y1))) \cap -((SFinite \cap -X1)\cdot(-((-Y) \cup Y1)))$
by *blast*
have *5*: $(X1\cdot Y) = ((SFinite \cap X1)\cdot Y) \cup (SInf \cap X1)$
by (*metis Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute*)

have *6*: $-((SFinite \cap X1)\cdot(-((-Y) \cup Y1)))$ =
$-((SFinite \cap X1)\cdot(Y \cap -Y1))$
by *simp*
have *7*: $-((SFinite \cap X1)\cdot(-((-Y) \cup Y1))) \cap ((SFinite \cap X1)\cdot Y) \subseteq$
$(SFinite \cap X1)\cdot(((-Y) \cup Y1) \cap Y)$
by (*metis* (*no-types*) *B20 FusionRuleR FusionUnionDistR double-compl inf-sup-aci*(*1*) *order-refl*)
have *8*: $(SFinite \cap X1)\cdot(((-Y) \cup Y1) \cap Y) \subseteq (SFinite \cap X1)\cdot Y1$
by (*metis B20 FusionRuleR double-compl inf.cobounded1 inf.right-idem*)
have *9*: $(SFinite \cap X1)\cdot Y1 \subseteq X1 \cdot Y1$
by (*simp add*: *FusionRuleL*)
have *10*: $-((SFinite \cap X1)\cdot(-((-Y) \cup Y1))) \cap (SInf \cap X1) \subseteq ((SFinite \cap X1)\cdot Y1) \cup (SInf \cap X1)$
by *blast*
have *11*: $((SFinite \cap X1)\cdot Y1) \cup (SInf \cap X1) \subseteq X1 \cdot Y1$
by (*metis B04 Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute*)
have *12*: $-((SFinite \cap (X1 \cup -X1))\cdot(-((-Y) \cup Y1))) \cap (X1\cdot Y) =$
$(-((SFinite \cap (X1 \cup -X1))\cdot(-((-Y) \cup Y1))) \cap ((SFinite \cap X1)\cdot Y)) \cup$
$(-((SFinite \cap (X1 \cup -X1))\cdot(-((-Y) \cup Y1))) \cap (SInf \cap X1))$
by (*simp add*: *5 B28*)
have *13*: $(-((SFinite \cap (X1 \cup -X1))\cdot(-((-Y) \cup Y1))) \cap ((SFinite \cap X1)\cdot Y)) \subseteq X1 \cdot Y1$
using *7 8 9 2 3* **by** *blast*
have *14*: $(-((SFinite \cap (X1 \cup -X1))\cdot(-((-Y) \cup Y1))) \cap (SInf \cap X1)) \subseteq X1 \cdot Y1$
using *10 11* **by** *blast*
have *15*: $(-((SFinite \cap (X1 \cup -X1))\cdot(-((-Y) \cup Y1))) \cap ((SFinite \cap X1)\cdot Y)) \cup$
$(-((SFinite \cap (X1 \cup -X1))\cdot(-((-Y) \cup Y1))) \cap (SInf \cap X1)) \subseteq X1 \cdot Y1$
using *13 14* **by** *blast*
show *?thesis*
using *1 12 15* **by** *blast*
qed

11.4.2 ITL Axioms derived

lemma *SBoxGen*:
assumes $X=STrue$
shows $(SAlways\ X) = STrue$
using *assms*
by (*metis Compl-disjoint2 FusionSFalse double-compl salways-def sfalse-def sfinite-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* **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*: $SSkip \cdot SFalse = SFalse$
using *FusionSFalse SStar16 SStarSkip sfalse-def sfinite-def* **by** *fastforce*
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*

11.5 Extra Laws

11.5.1 Boolean Laws

lemma *B02*:
assumes $-Y \subseteq -X$
shows $(X{::}\ 'a\ iintervals) \subseteq Y$
using *assms* **by** *auto*

lemma *B03*:
$((X{::}\ 'a\ iintervals) = Y\ ) \longleftrightarrow (-X = -Y)$
by *auto*

lemma *B05*:
assumes $(X{::}\ 'a\ iintervals) \cup Y \subseteq Z$
shows $X \subseteq Z \wedge Y \subseteq Z$
using *assms* **by** *auto*

lemma *B06*:
assumes $X \subseteq Z \wedge Y \subseteq Z$
shows $(X{::}\ 'a\ iintervals) \cup Y \subseteq Z$
using *assms* **by** *auto*

lemma *B07*:
$(X{::}\ 'a\ iintervals) \cup Y \subseteq Z \longleftrightarrow X \subseteq Z \wedge Y \subseteq Z$
by *auto*

lemma *B08*:
assumes $(X{::}\ 'a\ iintervals) \subseteq Y$
shows $-X \cup Y = STrue$
using *assms*
using *strue-def* **by** *auto*

lemma *B10*:
$(X{::}\ 'a\ iintervals) \subseteq Y \longleftrightarrow -X \cup Y = STrue$
using *strue-def* **by** *auto*

lemma *B11*:
assumes $(X{::}\ 'a\ iintervals) \subseteq Y$
shows $X \cap -Y = SFalse$

using *assms sfalse-def* **by** *auto*

lemma *B12*:
 assumes $X \cap -Y = \mathit{SFalse}$
 shows $(X:: 'a\ \mathit{iintervals}) \subseteq Y$
using *assms sfalse-def* **by** *auto*

lemma *B13*:
 $(X:: 'a\ \mathit{iintervals}) \subseteq Y \longleftrightarrow X \cap -Y = \mathit{SFalse}$
using *sfalse-def* **by** *auto*

lemma *B14*:
 assumes $(X:: 'a\ \mathit{iintervals}) \subseteq Y$
 shows $X \cap Y = X$
using *assms* **by** *auto*

lemma *B15*:
 assumes $(X:: 'a\ \mathit{iintervals}) \subseteq Y \cap Z$
 shows $X \subseteq Y \wedge X \subseteq Z$
using *assms* **by** *auto*

lemma *B16*:
 assumes $X \subseteq Y \wedge X \subseteq Z$
 shows $(X:: 'a\ \mathit{iintervals}) \subseteq Y \cap Z$
using *assms* **by** *auto*

lemma *B17*:
 $(X:: 'a\ \mathit{iintervals}) \subseteq Y \cap Z \longleftrightarrow X \subseteq Y \wedge X \subseteq Z$
by *auto*

lemma *B18*:
 assumes $(X:: 'a\ \mathit{iintervals}) \subseteq Y \cup Z$
 shows $X \cap -Y \subseteq Z$
using *assms* **by** *auto*

lemma *B19*:
 assumes $X \cap -Y \subseteq Z$
 shows $(X:: 'a\ \mathit{iintervals}) \subseteq Y \cup Z$
using *assms* **by** *auto*

lemma *B21*:
 $(X:: 'a\ \mathit{iintervals}) \cup (Y \cap Z) \subseteq (X \cup Y) \cap (X \cup Z) \longleftrightarrow$
  $X \cup (Y \cap Z) \subseteq (X \cup Y) \wedge X \cup (Y \cap Z) \subseteq (X \cup Z)$
by *auto*

lemma *B22*:
   $(X:: 'a\ \mathit{iintervals}) \cup (Y \cap Z) \subseteq X \cup Y$
by *auto*

lemma *B23*:

$(X:: 'a\ iintervals) \cup (Y \cap Z) \subseteq X \cup Z$
by *auto*

lemma *B24*:
$((X:: 'a\ iintervals) \cup Y) \cap (X \cup Z) \subseteq X \cup (Y \cap Z) \longleftrightarrow$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z$
by *auto*

lemma *B25*:
$(((X:: 'a\ iintervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z \longleftrightarrow$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \wedge$
$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$
by *auto*

lemma *B26*:
$(((X:: 'a\ iintervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y$
by *auto*

lemma *B27*:
$(((X:: 'a\ iintervals) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$
by *auto*

lemma *B29*:
$(X:: 'a\ iintervals) \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
by *auto*

11.5.2 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

11.5.3 Next

lemma *N01*:
(*SNext SEmpty*) = *SSkip*
by (*simp add*: *FusionSEmptyR snext-def*)

lemma *N02*:
(*SNext SFalse*) = *SFalse*
by (*metis FusionSFalse SStar16 SStarSkip disjoint-eq-subset-Compl sfalse-def sfinite-def snext-def*)

lemma *N03*:
$(SNext\ X) \cdot Y = (SNext\ (X \cdot Y))$
by (*simp add*: *snext-def FusionAssoc*)

lemma *N04*:
$(SNext\ (X \cup Y)) = (SNext\ X) \cup (SNext\ Y)$
by (*simp add*: *FusionUnionDistR snext-def*)

lemma *N05*:
$(SNext\ (X \cap Y)) = (SNext\ X) \cap (SNext\ Y)$
by (*metis CH07 SStar16 inf.orderE snext-def*)

lemma *N06*:
assumes $X \subseteq Y$
shows $(SNext\ X) \subseteq (SNext\ Y)$
using *assms*
by (*metis FusionUnionDistR Subsumption snext-def*)

lemma *N07*:
$(SNext\ ((-X) \cup Y)) = (SNext\ (-X)) \cup (SNext\ Y)$
by (*simp add*: *N04*)

lemma *N08*:
$SMore \subseteq SSkip \cdot STrue$
using *SMoreImpSSkipFusion* **by** *blast*

lemma *N23*:
$(SWprev\ X) \subseteq (SEmpty \cup (SPrev\ X))$
proof –
have *1*: $(X \cap SFinite) \cdot SSkip \cup (-\ X \cap SFinite) \cdot SSkip = SStar\ SSkip \cdot SSkip$
by (*metis B28 FusionUnionDistL SStarSkip STrueTop boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)
have *2*: $(SEmpty) \cup STrue \cdot SSkip = STrue$
by (*metis FusionSFalse FusionUnionDistL Powerstarhelp2 SStar19 SStarSkip UnfoldL UnionAssoc compl-bot-eq compl-sup-top sfinite-def sinf-def strue-def*)
have *3*: $-\ SWprev\ X \cup (SEmpty \cup SPrev\ X) = STrue$
unfolding *swprev-def sprev-def*
by (*metis 2 FusionUnionDistL compl-bot-eq compl-sup-top double-compl inf-sup-aci*(*7*) *strue-def*)
then show *?thesis* **by** (*meson B09*)
qed

lemma *N24*:
$(SEmpty) \subseteq (SWprev\ X)$
proof –
have *1*: $(-\ SEmpty \cap -\ (-\ SEmpty \cdot -\ SEmpty)) \subseteq SMore$
by (*simp add*: *smore-def*)
have *2*: $-X \cdot SMore \subseteq SMore$
by (*metis CH01 FusionAssoc1 FusionUnionDistL SMoreImpSSkipFusion SSkipFusionImpSMore SStar30 SStar48 STrueTop subset-antisym sup.orderI sup.right-idem*)
have *3*: $-\ X \cdot (-\ SEmpty \cap -\ (-\ SEmpty \cdot -\ SEmpty)) \subseteq SMore$
using *1 2 FusionRuleR* **by** *blast*
show *?thesis*
by (*metis 3 compl-le-swap1 compl-sup smore-def sskip-def swprev-def*)
qed

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*: $((SPrev\ X) \cap (SPrev\ (-X))) = (X \cap (-X)) \cdot SSkip$
by (*metis CH08 SStar16 inf.orderE sprev-def*)
have *3*: $(X \cap (-X)) \cdot SSkip = SFalse \cdot SSkip$
by (*simp add*: *sfalse-def*)
have *4*: $SFalse \cdot SSkip = SFalse$
by (*simp add*: *SFalseFusion*)
from *1 2 3 4* **show** *?thesis* **by** *auto*

qed

lemma *N26*:
$(SWprev\ X) = \ (SEmpty \cup \ (SPrev\ X))$
using *N23 N24 N25* **by** *blast*

lemma *N09*:
SSkip $\cup$ *SMore·SSkip* $\subseteq$ *SMore*
proof −
have *1*: *SSkip* $\subseteq$ *SMore*
by (*simp add*: *smore-def sskip-def*)
have *2*: *SMore·SSkip* $\subseteq$ *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* $\cup$ *SMore·SSkip* $\subseteq$ *SMore*
shows *SSkip*·(*SStar SSkip*) $\subseteq$ *SMore*
using *assms SStarInductR N09* **by** *blast*

lemma *N11*:
SSkip · *STrue* $\subseteq$ *SMore*
by (*simp add*: *SSkipFusionImpSMore*)

lemma *N12*:
$(SNext\ X) = -(SWnext\ (-X))$
by (*simp add*: *snext-def swnext-def*)

lemma *N13*:
SMore·STrue = *SMore*
by (*metis CH01 FusionAssoc1 SMoreImpSSkipFusion SSkipFusionImpSMore subset-antisym*)

lemma *N14*:
STrue·SSkip $\subseteq$ *SMore*
by (*metis N09 ST47 STrueTop smore-def*)

lemma *N15*:
SMore $\subseteq$ *STrue·SSkip*
proof −
have *1*: *SMore* $\subseteq$ *SSkip* · *STrue*
by (*simp add*: *SMoreImpSSkipFusion*)
have *2*: *SSkip* · *STrue* $\subseteq$ *STrue·SSkip*
proof −
have *f3*: *SSkip* · *SFinite* $\subseteq$ *STrue* · *SSkip*
by (*metis B19 FusionRuleL SStar19 SStarSkip STrueTop inf-sup-ord*(*2*))
have *SSkip* · *SInf* $\subseteq$ *STrue* · *SSkip*
by (*metis* (*no-types*, *opaque-lifting*) *B04 Compl-disjoint2 FusionRuleR SChopAssoc SMoreImpSSkip-Fusion*
SSkipFusionImpSMore ST47 boolean-algebra-cancel.inf0 compl-inf double-compl

inf-commute inf-le1 sfalse-def sinf-def strue-def)
then show *?thesis*
using *f3* **by** (*metis* (*no-types*) *FusionUnionDistR STrueTop le-sup-iff sfinite-def*)
qed
show *?thesis*
using *2 SMoreImpSSkipFusion* **by** *blast*
qed

lemma *N19*:
$(SWnext\ X) \subseteq (SEmpty \cup (SNext\ X))$
proof −
have $STrue \subseteq SSkip \cdot (-\ X) \cup (SEmpty \cup SSkip \cdot X)$
by (*metis B04 FusionUnionDistR N13 SMoreImpSSkipFusion SSkipFusionImpSMore SStar48 UnfoldL compl-bot-eq compl-sup-top inf-sup-aci*(*7*) *strue-def*)
then show *?thesis*
by (*simp add*: *B13 snext-def strue-def swnext-def*)
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*: $(SNext\ (-X)) \subseteq SSkip \cdot STrue$
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 CH07 SSkipSFinite compl-inf*)

lemma *N27*:
$(SNext\ ((-X) \cup Y)) \subseteq (-(SNext\ X) \cup (SNext\ Y))$
using *N04 SNextImpNotNextNot* **by** *blast*

lemma *N28*:
$(SPrev\ ((-X) \cup Y)) \subseteq (\ -(SPrev\ X) \cup (SPrev\ Y))$
unfolding *sprev-def*
proof –
have $\bigwedge I.\ (SEmpty) \cup I \cdot SSkip = -\ (-\ I \cdot SSkip)$
using *N26 sprev-def swprev-def* **by** *blast*
then have $(Y \cup -\ X) \cdot SSkip \subseteq Y \cdot SSkip \cup -\ (X \cdot SSkip)$
using *FusionUnionDistL* **by** *blast*
then show $(-\ X \cup Y) \cdot SSkip \subseteq -\ (X \cdot SSkip) \cup Y \cdot SSkip$
by (*simp add*: *UnionCommute*)
qed

lemma *N29*:
$(SPrev\ X) = -(SWprev\ (-X)\ )$
by (*simp add*: *sprev-def swprev-def*)

11.5.4 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*:
$(X \cap SEmpty) \subseteq (X \cap SEmpty) \cdot (X \cap SEmpty)$
proof –
have $\forall\ S\ Sa\ Sb\ Sc.\ (Sc) \cdot (Sb \cap SEmpty) \cap (Sa \cdot (S \cap SEmpty)) = Sc \cap Sa \cdot (Sb \cap (S \cap SEmpty))$
by (*simp add*: *CH10 inf-assoc*)
then have $\forall\ S\ Sa.\ (Sa) \cdot (S \cap SEmpty) \cdot (S \cap SEmpty) = Sa \cdot (S \cap SEmpty)$
by (*metis FusionSEmptyR inf.idem inf-commute*)
then show *?thesis*
by (*metis FusionSEmptyL subset-refl*)
qed

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 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*:
$(SInit\ X) \cap (SDi\ Y) = (SDi\ (\ (SInit\ X) \cap Y))$
unfolding *sdi-def sinit-def strue-def*
by (*metis CH09 FusionAssoc FusionSEmptyL compl-bot-eq inf-top.left-neutral*)

lemma *ST31*:
$(X \cdot (STrue \cap SEmpty)) \cap (STrue \cdot (Y \cap SEmpty)) = X \cdot (Y \cap SEmpty)$
by (*metis Int-commute compl-bot-eq inf-top.right-neutral CH10 strue-def*)

lemma *ST32*:
$(STrue \cap SEmpty) \cdot SEmpty \cap (SInit\ X) = (X \cap SEmpty)$
proof –
have $\forall S\ Sa.\ (Sa) \cap (S \cap Sa) = S \cap Sa$
by *fastforce*
then have *f1*: $\forall S\ Sa.\ (Sa) \cap (S \cap SEmpty) \subseteq Sa \cap SEmpty \cdot STrue$
by (*metis* (*no-types*) *ST07 inf-assoc*)
have *f4*: $\forall S.\ -(-\ (S::'a\ iintervals)) = S$
by *blast*
have *f6*: $\forall S.\ (SEmpty) \cap (S \cap (S \cap SEmpty \cdot STrue)) = S \cap SEmpty$
using *f1* **by** *auto*
have *f7*: $\forall S.\ (SEmpty) \cap (SEmpty \cap S \cdot STrue) \subseteq S$
using *f4* **by** (*metis CH11 FusionSEmptyR inf-aci*(*1*) *inf-le2 pwr-0*)
have $\forall S.\ (SEmpty) \cap (S \cap (SEmpty \cap S \cdot STrue)) = SEmpty \cap S$
using *f6* **by** (*simp add*: *inf-commute*)
then have $SEmpty \cap (SEmpty \cap X \cdot STrue) = SEmpty \cap X$
using *f7* **by** *auto*
then show *?thesis*
by (*metis* (*no-types*) *ST33 inf-commute sinit-def*)
qed

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*)

11.5.5 SStar

lemma *SStar02*:
assumes $X \subseteq Y$
shows $X \cdot (SStar\ Y) \cup SEmpty \subseteq (SStar\ Y)$

using *assms UnfoldL*[*of Y*]
by (*metis B05 B06 FusionRuleL Subsumption sup.cobounded2*)

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*:
$(SStar\ SSkip) = SFinite$
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*:
$X \subseteq (SStar\ X)$
using *UnfoldL*[*of X*]
by (*metis B05 FusionSEmptyR FusionUnionDistR SStar10*)

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*:
$(SStar\ (X \cdot Y)) \cdot X \subseteq X \cdot (SStar\ (Y \cdot X))$
by (*meson SStar32 SStarInductL*)

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*

11.5.6 Box and Diamond

lemma *BD01*:
 $(SSometime\ SEmpty) = SFinite$
by (*simp add*: *ssometime-def FusionSEmptyR*)

lemma *BD02*:
 $X \subseteq (SSometime\ X)$
unfolding *ssometime-def*
by (*metis SStar15 SStarSkip ST47 subset-Un-eq*)

lemma *BD03*:
 $(SNext\ (SSometime\ X)) \subseteq (SSometime\ X)$
by (*metis FusionUnionDistL N03 SStar16 SStar03 SStar04 SStarSkip snext-def ssometime-def sup.absorb-iff2 sup.orderE*)
lemma *BD04*:
 $(SSometime\ (SNext\ X)) \subseteq (SSometime\ X)$
by (*metis BD03 FusionAssoc SStar28 SStar29 SStarSkip inf-sup-aci*(5) *snext-def ssometime-def sup.orderE*)

lemma *BD05*:
 $(SSometime\ X) \cup (SSometime\ Y) = (SSometime\ (X \cup Y))$
by (*simp add*: *FusionUnionDistR ssometime-def*)

lemma *BD06*:
 $(SSometime\ STrue) = STrue$
using *BD02 SMP* **by** *blast*

lemma *BD07*:

(*SSometime* (*X* ∩ *Y*)) ⊆ (*SSometime* *X*) ∩ (*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*) ⊆ (*SSometime X*)
by (*metis B02 BD02 BD09 set-rev-mp subsetI*)

lemma *BD11*:
(*SSometime* (*SSometime X*)) = (*SSometime X*)
by (*metis FusionAssoc SStar03 SStar04 SStarSkip ssometime-def subset-antisym*)

lemma *BD12*:
(*SAlways X*) ⊆ *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* ∪ *Y*)) = (*SDi X*) ∪ (*SDi Y*)
by (*simp add*: *FusionUnionDistL sdi-def*)

lemma *BD17*:
assumes *X* ⊆ *Y*
shows (*SDi X*) ⊆ (*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 (*metis BD01 BD06 sda-def ssometime-def*)

lemma *BD20*:
$(SDa\ STrue) = STrue$
by (*metis BD06 CH01 sda-def ssometime-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 B08 B12 FusionRuleR FusionSEmptyR SFalseBottom double-compl inf.absorb-iff2 inf-le1 sda-def ssometime-def sup-inf-absorb*)

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*:
$(SAlways\ X) \cap (SAlways\ Y) = (SAlways\ (X \cap Y))$
proof −
have *1*: $(SAlways\ X) \cap (SAlways\ Y) = -\ SSometime\ (-\ X) \cap -\ SSometime\ (-\ Y)$
by (*simp add* : *salways-def*)
have *2*: $SSometime\ (-\ X) \cup\ SSometime\ (-\ Y) = SSometime(-X \cup -Y)$

using *BD05* **by** *blast*
have *3*: $-$ *SSometime*$(-X \cup -Y) = ($*SAlways* $(X \cap Y))$
by (*simp add* : *salways-def*)
show *?thesis* **using** *1 2 3* **by** *auto*
qed

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 B08 BD08 BD12 FusionSEmptyR N20 SBoxInduct ST33 sup.absorb-iff2 sup.orderE*)

lemma *BD36*:
(*SBi SEmpty*) = *SEmpty*
proof $-$
have *1*: $- (-$ *SEmpty* $\cdot$ *STrue*$) \subseteq$ *SEmpty*
by (*metis B04 N13 double-compl smore-def*)
have *2*: *SEmpty* $\subseteq - (-$ *SEmpty* $\cdot$ *STrue*$)$
using *N13 smore-def* **by** *fastforce*
show *?thesis*
by (*simp add*: *1 2 B04 sbi-def sdi-def*)
qed

lemma *BD37*:
(*SBa SEmpty*) = *SEmpty*
by (*metis BD09 BD35 BD36 sba-def sbi-def sda-def sdi-def ssometime-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 BD11 CH01 FusionAssoc sda-def ssometime-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 (*simp add*: *FusionAssoc1 FusionRuleR sdi-def sprev-def strue-elim subsetI*)

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*:
  $(SAlways\ (SMore \cup X)) = STrue\cdot(X \cap SEmpty)$
proof –
 have *1*: $((SFinite\cdot(X \cap SEmpty)) \cup SInf) = (STrue\cdot(X \cap SEmpty))$
  by (*metis Powerstarhelp2 Powerstarhelp3 UnionCommute boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)
 have *11*: $SInf \cup SFinite \cdot (SEmpty \cap X \cup SEmpty \cap -\ X) = STrue$
  by (*simp add*: *B28 FusionSEmptyR sfinite-def strue-def*)
 have *2*: $(SAlways\ (SMore \cup X)) \cap SFinite \subseteq SFinite\cdot(X \cap SEmpty)$
  using *11*
  by (*simp add*: *B09 BD09 FusionUnionDistR UnionCommute inf-commute inf-sup-aci*(*7*) *smore-def ssometime-def sfinite-def*)
 have *3*: $(SAlways\ (SMore \cup X)) \subseteq ((SFinite\cdot(X \cap SEmpty)) \cup SInf)$
  using *2 sfinite-def* **by** *fastforce*
 have *4*: $(SAlways\ (SMore \cup X)) \subseteq\ STrue\cdot(X \cap SEmpty)$
  using *1 3* **by** *blast*
 have *5*: $SFinite\cdot(X \cap SEmpty) \subseteq (SAlways\ (SMore \cup X))$
  unfolding *salways-def ssometime-def smore-def*
  using *CH10*[*of SFinite X SFinite*] *FusionSFalse*
  by (*metis* (*no-types*, *opaque-lifting*) *SFalseBottom compl-sup disjoint-eq-subset-Compl double-compl inf-commute inf-le2 sfinite-def*)
 have *6*: $SInf \subseteq (SAlways\ (SMore \cup X))$
  by (*metis B05 BD30 FusionSEmptyR double-compl salways-def sfinite-def smore-def ssometime-def*)
 show *?thesis*
  using *1 3 5 6* **by** *auto*
qed

lemma *BD57*:
  $(SAlways\ H) \subseteq SAlways\ (-G \cup ((SAlways\ H) \cap\ G))$
proof –
 have *1*: $(SAlways\ H) \subseteq (-G \cup ((SAlways\ H) \cap\ G))$

by *blast*
have *2*: *SAlways* (*SAlways* *H*) ⊆ *SAlways* (−*G* ∪ ((*SAlways* *H*)∩ *G*))
by (*simp add*: *1 BD38*)
have *3*: *SAlways* (*SAlways* *H*) = (*SAlways* *H*)
by (*simp add*: *BD42*)
show *?thesis* **using** *2 3* **by** *blast*
qed

lemma *BD58*:
((*SAlways* *H*) ∩ (*F*·*G*)) ⊆ (*F*·((*SAlways* *H*) ∩ *G*))
proof −
have *1*: *SAlways* (−*G* ∪ ((*SAlways* *H*)∩ *G*)) ∩ (*F*·*G*) ⊆ (*F*·((*SAlways* *H*) ∩ *G*))
using *BD46* **by** *blast*
show *?thesis* **using** *1 BD57* **by** *blast*
qed

11.5.7 Finite and Infinite

lemma *FI01*:
SFinite· *SFinite* = *SFinite*
by (*metis BD01 BD11 ssometime-def*)

lemma *FI02*:
(*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*) ⊆ (*X*·*Y*) ∩ *SFinite*
by (*metis B16 CH06 FI01 inf.cobounded1 inf-le2*)

lemma *FI03*:
(*X*·*Y*) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
proof −
have *1*: (*X*·*Y*) = (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*) ∪ (*X* ∩ *SFinite*)·(*Y* ∩ *SInf*)
∪ (*X* ∩ *SInf*)·(*Y* ∩ *SFinite*) ∪ (*X* ∩ *SInf*)·(*Y* ∩ *SInf*)
by (*metis FusionUnionDistR Powerstarhelp2 Powerstarhelp3 SInfSFinite UnionCommute sup.right-idem*)
have *2*: ((*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
by *auto*
have *3*: (*X* ∩ *SFinite*)·(*Y* ∩ *SInf*) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
by (*metis* (*no-types*, *lifting*) *B20 FusionAssoc FusionSFalse inf-le2 sfinite-def subset-trans sup-ge1*)
have *4*: (*X* ∩ *SInf*)·(*Y* ∩ *SFinite*) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
using *Powerstarhelp2 sfinite-def* **by** *fastforce*
have *5*: (*X* ∩ *SInf*)·(*Y* ∩ *SInf*) ∩ *SFinite* ⊆ (*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*)
by (*metis 4 Powerstarhelp2*)
have *6*: (*X*·*Y*) ∩ *SFinite* = ((*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*) ∩ *SFinite*) ∪
((*X* ∩ *SFinite*)·(*Y* ∩ *SInf*) ∩ *SFinite*) ∪
((*X* ∩ *SInf*)·(*Y* ∩ *SFinite*) ∩ *SFinite*) ∪
((*X* ∩ *SInf*)·(*Y* ∩ *SInf*) ∩ *SFinite*)
using *1* **by** *auto*
from *6 2 3 4 5* **show** *?thesis* **by** *auto*
qed

lemma *FI04*:
(*X* ∩ *SFinite*)·(*Y* ∩ *SFinite*) = (*X*·*Y*) ∩ *SFinite*

using *FI03 FI02* **by** *fastforce*

lemma *FI05*:
SAlways SMore ⊆ *SInf*
by (*metis B04 BD56 Compl-disjoint2 sfalse-def sinf-def smore-def sup.orderE*)

lemma *FI06*:
SEmpty ⊆ *SFinite*
using *BD01 BD02* **by** *auto*

lemma *FI07*:
SSkip·SFinite ⊆ *SFinite*
using *SStarSkip UnfoldL* **by** *fastforce*

lemma *FI08*:
SFinite·SSkip ⊆ *SFinite*
by (*metis FI07 SStar19 SStarSkip*)

lemma *FI09*:
SFinite·SSkip = *SFinite* ∩ *SMore*
proof −
have *1*: *SFinite·SSkip* ⊆ *SFinite* ∩ *SMore*
by (*metis B16 FI07 N09 N10 SStar19 SStarSkip*)
have *2*: *SFinite* ∩ *SMore* ⊆ *SFinite·SSkip*
by (*metis B20 FI01 FI02 SStar19 SStarSkip UnfoldL inf.idem smore-def*)
show *?thesis* **using** *1 2* **by** *blast*
qed

lemma *FI10*:
SFinite· SSkip = *SSkip ·SFinite*
by (*metis SStar19 SStarSkip*)

lemma *FI11*:
SFinite ∩ *SEmpty* = *SEmpty*
by (*simp add*: *FI06 Int-absorb2 inf-sup-aci*(*1*))

lemma *FI12*:
SInf·SInf = *SInf*
by (*metis FusionSFalse Powerstarhelp2 sinf-def*)

lemma *FI13*:
SFinite·SInf = *SInf*
by (*metis BD06 FusionAssoc1 sinf-def ssometime-def*)

lemma *FI14*:
SInf · *SFinite* = *SInf*
by (*metis FusionSFalse Powerstarhelp2 sinf-def*)

lemma *FI15*:
SInf = − *SFinite*

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

lemma *FI16*:
SFinite = − *SInf*
by (*simp add*: *sfinite-def*)

lemma *FI17*:
((*F*·*STrue*) ∩ *SFinite*) = (*F* ∩ *SFinite*)·*SFinite*
by (*metis FI04 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)

lemma *FI18*:
((*STrue*·*F*) ∩ *SFinite*) = *SFinite*·(*F* ∩ *SFinite*)
by (*metis BD19 FI01 FI04 FI17 FusionSEmptyR inf.idem sda-def*)

lemma *FI19*:
SFinite· *SMore* = *SMore*
by (*metis BD06 FusionAssoc1 N14 N15 ssometime-def subset-antisym*)

lemma *FI20*:
SInf ∪ *SFinite* = *STrue*
using *STrueTop sfinite-def* **by** *auto*

lemma *FI21*:
SFMore = *SSkip*·*SFinite*
using *FI09 FI10 sfmore-def* **by** *auto*

lemma *FI22*:
(*F* ∩ *SFinite* ⊆ *G*) = ((*F* ∩ *SFinite*) ⊆ (*G* ∩ *SFinite*))
by *simp*

lemma *FI23*:
(*F*·*G*) ∩ *SInf* = *F* · (*G* ∩ *SInf*)
by (*metis FusionAssoc FusionSFalse*)

lemma *FI24*:
((*F*·*G*) ∩ *SInf*) = ((*F* ∩ *SInf*) ∪ ((*F* ∩ *SFinite*)·(*G* ∩ *SInf*)))
using *FI23*[*of F G*] *Powerstarhelp2 Powerstarhelp3* **by** *fastforce*

lemma *FI25*:
SMore ∩ *SInf* = *SInf*
using *SStar15 SStarSkip sfinite-def smore-def* **by** *fastforce*

lemma *FI26*:
(*F* ∩ *SInf*)·(*F* ∩ *SInf*) = (*F* ∩ *SInf*)
using *Powerstarhelp2* **by** *blast*

lemma *FI27*:
(*F* ∩ *SInf*)·*G* = (*F* ∩ *SInf*)
using *Powerstarhelp2* **by** *blast*

lemma *FI28*:
$((F \cap SMore) \cap SInf) = (F \cap SInf)$
using *FI25* **by** *blast*

lemma *FI29*:
$((F \cap SMore) \cap SFinite) = (F \cap SFMore)$
by (*metis inf-assoc inf-commute sfmore-def*)

lemma *FI30*:
$(SEmpty \cap SFMore) = SFalse$
by (*simp add*: *sfalse-def sfmore-def smore-def*)

lemma *FI31*:
$((F \cap SInf) \cap SFMore) = SFalse$
by (*simp add*: *sfalse-def sfinite-def sfmore-def*)

lemma *FI32*:
$(SEmpty \cap SInf) = SFalse$
using *FI25 sfalse-def smore-def* **by** *auto*

lemma *FI33*:
$(F \cap SInf) = F \cdot SFalse$
by (*simp add*: *FusionSFalse*)

lemma *FI34*:
$(F \cap SFinite) \cdot G \subseteq SSometime\ G$
by (*simp add*: *FusionRuleL ssometime-def*)

lemma *FI35*:
assumes $F \subseteq SNext\ F$
shows $SFinite \subseteq -F$
using *assms*
by (*metis B01 FusionSEmptyR N12 N22 SStarInductL SStarSkip double-compl snext-def*)

lemma *FI36*:
assumes $F \cap -G \subseteq SNext\ F$
shows $F \cap SFinite \subseteq SSometime\ G$
using *assms*
proof −
have *1*: $F \cap -G \subseteq SNext\ F$
using *assms* **by** *auto*
have *2*: $F \cap -G \cap SAlways\ (-G) \subseteq SNext\ F \cap SAlways\ (-G)$
using *assms* **by** *blast*
have *3*: $SAlways\ (-G) \subseteq -G$
by (*simp add*: *BD12*)
have *4*: $SAlways\ (-G) = SAlways\ (-G) \cap -G$
using *3* **by** *blast*
have *5*: $F \cap SAlways\ (-G) \subseteq SNext\ F \cap SAlways\ (-G)$
using *2 4* **by** *blast*
have *51*: $(SFinite \cdot G) = G \cup (SSkip \cdot (SFinite \cdot G))$

by (*metis FusionAssoc1 SStarSkip ST47 UnfoldL*)
have *6*: *SAlways* $(-G) = -G \cap$ *SWnext* (*SAlways* $(-G)$)
using *51* **by** (*simp add*: *salways-def ssometime-def swnext-def*) *blast*
have *7*: *SNext F* $\cap$ *SAlways* $(-G) \subseteq$ *SNext F* $\cap$ *SWnext* (*SAlways* $(-G)$)
using *6* **by** *blast*
have *8*: $F \cap$ *SAlways* $(-G) \subseteq$ *SNext F* $\cap$ *SWnext* (*SAlways* $(-G)$)
using *5 7* **by** *blast*
have *9*: $F \cap$ *SAlways* $(-G) \subseteq$ *SMore* $\cap$ *SWnext F* $\cap$ *SWnext* (*SAlways* $(-G)$)
using *8 N16* **by** *blast*
have *10*: $F \cap$ *SAlways* $(-G) \subseteq$ *SWnext F* $\cap$ *SWnext* (*SAlways* $(-G)$)
using *8 N16* **by** *fastforce*
have *11*: $F \cap$ *SAlways* $(-G) \subseteq$ *SWnext* ($F \cap$ *SAlways* $(-G)$)
using *10 N17* **by** *blast*
have *12*: *SAlways* ($F \cap$ *SAlways* $(-G)$) $\subseteq$ *SWnext* ($F \cap$ *SAlways* $(-G)$)
by (*metis 10 B15 BD12 N17 inf.absorb-iff2*)
have *13*: *SAlways* ($F \cap$ *SAlways* $(-G)$) $\subseteq$
SWnext ($F \cap$ *SAlways* $(-G)$) $\cap F \cap (-($*SAlways* $(-G)) \cup$ *SAlways*($F \cap$ *SAlways* $(-G)$)))
using *12 BD12* **by** *auto*
have *14*: ($F \cap$ *SAlways* $(-G)$) $\subseteq$ *SAlways*($F \cap$ *SAlways* $(-G)$)
using *12 13*
by (*metis 10 B08 BD08 N17 SBoxInduct boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)
have *15*: *SAlways*($F \cap$ *SAlways* $(-G)$) $\subseteq$ ($F \cap$ *SAlways* $(-G)$)
using *BD12* **by** *blast*
have *16*: *SAlways* ($F \cap$ *SAlways* $(-G)$) = ($F \cap$ *SAlways* $(-G)$)
using *14 15* **by** *blast*
have *17*: ($F \cap$ *SAlways* $(-G)$) $\subseteq$ *SMore*
using *9* **by** *blast*
have *18*: *SAlways* ($F \cap$ *SAlways* $(-G)$) $\subseteq$ *SAlways SMore*
by (*simp add*: *17 BD38*)
have *19*: *SFinite* = $-$(*SAlways SMore*)
by (*simp add*: *BD01 salways-def smore-def*)
have *20*: *SFinite* $\subseteq -(F \cap$ *SAlways* $(-G)$)
using *16 18 19* **by** *blast*
have *21*: *SFinite* $\subseteq -F \cup -($*SAlways* $(-G)$)
using *20* **by** *blast*
have *22*: $-($*SAlways* $(-G)$) = *SSometime G*
by (*simp add*: *BD09*)
show *?thesis*
using *20 22* **by** *blast*
qed

lemma *FI37*:
assumes $F \cap -G \subseteq$ *SNext* $(F \cap -G)$
shows $F \cap$ *SFinite* $\subseteq$ *SSometime G*
using *assms*
by (*metis B15 FI36 N05*)

lemma *FI38*:
assumes $F \cap -G \subseteq$ (*SNext F*) $\cap -$(*SNext G*)
shows $F \cap$ *SFinite* $\subseteq G$

proof –
 have *1*: $F \cap -G \subseteq SNext((F \cap -G))$
  using *N17*[*of F* $-G$]
  by (*metis* (*no-types*, *lifting*) *N12 N16 assms double-compl inf.semigroup-axioms semigroup.assoc*)
 have *2*: $SFinite \subseteq -((F \cap -G))$
  using *1 FI35* **by** *auto*
 show *?thesis*
  using *2* **by** *blast*
qed

lemma *FI39*:
 assumes $SWnext\ (SSometime\ F) \subseteq F$
 shows $SFinite \subseteq F$
proof –
 have *1*: $-F \subseteq SNext\ (-F)$
  by (*metis BD02 N12 N18 Subsumption assms compl-le-swap2 double-complement sup.coboundedI1*)
 from *1* **show** *?thesis* **using** *FI35* **by** *blast*
qed

lemma *FI40*:
 assumes $SEmpty \subseteq F$
   $SNext\ F \subseteq F$
 shows $SFinite \subseteq F$
proof –
 have *1*: $-F \subseteq SNext\ (-F)$
  using *N12 N19 assms*(*1*) *assms*(*2*) **by** *blast*
 from *1* **show** *?thesis* **using** *FI35* **by** *blast*
qed

lemma *FI41*:
 assumes $SEmpty \cap F \subseteq G$
   $SNext\ (-F \cup G) \cap F \subseteq G$
 shows $F \cap SFinite \subseteq G$
proof –
 have *1*: $(F \cap -G) \subseteq SNext\ (F \cap -G)$
  using *N19*[*of* $(-F \cup G)$]
  using *assms*(*1*) *assms*(*2*) *snext-def swnext-def* **by** *fastforce*
 have *2*: $SFinite \subseteq -\ (F \cap -G)$
  using *1 FI35* **by** *auto*
 from *2* **show** *?thesis* **by** *blast*
qed

lemma *FI42*:
 assumes $F \subseteq SFMore\cdot\ F$
 shows $SFinite \subseteq -F$
proof –
 have *1*: $SSometime\ F \subseteq SNext\ (SSometime\ F)$
  by (*simp add*: *ssometime-def snext-def*)
   (*metis FI21 SChopAssoc SStarSkip ST47 UnfoldL assms order-refl subset-Un-eq*)
 have *2*: $SFinite \subseteq -(SSometime\ F)$

by (*simp add*: *1 FI35*)
show *?thesis*
by (*metis 2 B01 FI21 SStarSkip ST47 UnfoldL assms le-iff-sup ssometime-def subset-trans*)
qed

lemma *FI43*:
assumes $F \cap -G \subseteq SFMore \cdot (F \cap -G)$
shows $F \cap SFinite \subseteq G$
proof –
have *1*: $SFinite \subseteq -(F \cap -G)$
using *FI42 assms* **by** *blast*
from *1* **show** *?thesis* **by** *blast*
qed

lemma *FI44*:
assumes $F \cap SFinite \subseteq SFMore \cdot F$
shows $SFinite \subseteq -F$
proof –
have *1*: $F \cap SFinite \subseteq SFMore \cdot (F \cap SFinite)$
by (*metis FI04 FI21 FI22 SSkipSFinite assms inf.idem*)
show *?thesis*
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma *FI45*:
assumes $F \cap SFinite \subseteq SMore \cdot F$
shows $SFinite \subseteq -F$
proof –
have *1*: $F \cap SFinite \subseteq SFMore \cdot (F \cap SFinite)$
by (*metis FI04 FI22 assms inf-commute sfmore-def*)
show *?thesis*
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma *FI46*:
assumes $F \subseteq SMore \cdot F$
shows $SFinite \subseteq -F$
proof –
have *1*: $F \cap SFinite \subseteq SFMore \cdot (F \cap SFinite)$
using *B11 FI45 assms* **by** *auto*
show *?thesis*
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma *FI47*:
assumes $(F \cap -G) \cap SFinite \subseteq SFMore \cdot (F \cap -G)$
shows $F \cap SFinite \subseteq G$
proof –
have *1*: $SFinite \subseteq -(F \cap -G)$
using *FI44 assms* **by** *blast*

from *1* **show** *?thesis* **by** *blast*
qed

lemma *FI48*:
assumes $(F \cap -G) \subseteq SMore \cdot (F \cap -G)$
shows $F \cap SFinite \subseteq G$
proof –
have *1*: $SFinite \subseteq -(F \cap -G)$
using *FI45 assms* **by** *blast*
from *1* **show** *?thesis* **by** *blast*
qed

lemma *FI49*:
assumes $F \subseteq G \cdot F$
$G \subseteq SFMore$
shows $SFinite \subseteq -F$
proof –
have *1*: $G{\cdot}F \subseteq SFMore \cdot F$
by (*simp add*: *FusionRuleL assms*(*2*))
have *2*: $F \subseteq SFMore \cdot F$
using *1 assms*(*1*) **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *FI42*)
qed

lemma *FI50*:
assumes $F \subseteq G \cdot F$
$G \subseteq SMore$
shows $SFinite \subseteq -F$
proof –
have *1*: $G{\cdot}F \subseteq SMore \cdot F$
by (*simp add*: *FusionRuleL assms*(*2*))
have *2*: $F \subseteq SMore \cdot F$
using *1 assms*(*1*) **by** *auto*
from *2* **show** *?thesis*
by (*simp add*: *FI46*)
qed

lemma *FI51*:
assumes $F \cap -G \subseteq (H{\cdot}F) \cap -(H{\cdot}G)$
$H \subseteq SFMore$
shows $F \cap SFinite \subseteq G$
proof –
have *1*: $H{\cdot}(F \cap -G) \subseteq SFMore \cdot (F \cap -G)$
by (*simp add*: *FusionRuleL assms*(*2*))
have *2*: $F \cap -G \subseteq SFMore \cdot (F \cap -G)$
using *1 CH02 assms*(*1*) **by** *blast*
from *2* **show** *?thesis*
by (*simp add*: *FI43*)
qed

lemma *FI52*:
assumes $F \cap -G \subseteq (H{\cdot}F) \cap -(H{\cdot}G)$
$H \subseteq SMore$
shows $F \cap SFinite \subseteq G$
proof –
have *1*: $H{\cdot}(F \cap -G) \subseteq SMore \cdot (F \cap -G)$
by (*simp add*: *FusionRuleL assms*(*2*))
have *2*: $F \cap -G \subseteq SMore \cdot (F \cap -G)$
using *1 CH02 assms*(*1*) **by** *blast*
from *2* **show** *?thesis*
by (*simp add*: *FI48*)
qed

lemma *FI53*:
$SAlways\ SInf = SInf$
by (*metis BD01 BD35 BD42 FI15 salways-def*)

11.5.8 Omega

lemma *OA01*:
$(somega\ SSkip) = SSkip{\cdot}(somega\ SSkip)$
by (*metis SOmegaUnroll SSkipSFinite SSkipSMore*)

lemma *OA02*:
$(somega\ SEmpty) = SFalse$
by (*metis FI30 SFalseFusion SOmegaUnroll inf-assoc inf-commute sfmore-def*)

lemma *OA03*:
$(somega\ SFalse) = SFalse$
by (*metis Compl-disjoint2 Powerstarhelp2 SOmegaUnroll inf-assoc inf-compl-bot-right sfalse-def*)

lemma *OA04*:
$(somega\ SInf) = SFalse$
by (*metis FI25 SFalseBottom SFalseFusion SOmegaUnroll inf-commute sfinite-def*)

lemma *BD59*:
$SInf \cap G \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite){\cdot}G)) \subseteq$
$((F \cap SMore) \cap SFinite){\cdot}(SInf \cap G \cap SAlways(-G \cup (((F \cap SMore) \cap SFinite){\cdot}G)))$
proof –
have *1*: $SInf \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite){\cdot}G)) \subseteq (-G \cup (((F \cap SMore) \cap SFinite){\cdot}G))$
using *BD12* **by** *blast*
have *2*: $SInf \cap G \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite){\cdot}G)) \subseteq$
$SInf \cap (((F \cap SMore) \cap SFinite){\cdot}G) \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite){\cdot}G))$
using *1* **by** *blast*
have *3*: $SInf \cap (((F \cap SMore) \cap SFinite){\cdot}G) \cap SAlways\ (-G \cup (((F \cap SMore) \cap SFinite){\cdot}G)) \subseteq$
$((F \cap SMore) \cap SFinite){\cdot}(SInf \cap G \cap SAlways(-G \cup (((F \cap SMore) \cap SFinite){\cdot}G)))$
by (*metis BD58 FI23 inf-commute*)
show *?thesis* **using** *2 3* **by** *blast*
qed

lemma *OA06*:
$SInf \cap G \cap SAlways\ (\ -G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq (somega\ F)$
by (*metis* (*no-types*, *lifting*) *BD59 SOmegaWeakCoinduct*)

lemma *FI54*:
$SAlways\ SFMore \subseteq SInf$
by (*metis BD38 BD56 Compl-disjoint2 SMoreImpSSkipFusion SSkipFusionImpSMore inf-le2 sfalse-def sfmore-def sinf-def smore-def subset-antisym sup.orderE*)

lemma *FI55*:
$SAlways\ SFinite = SFinite$
using *FI13 FI15 salways-def ssometime-def* **by** *fastforce*

lemma *FI56*:
$(SAlways\ SFMore) = SFalse$
using *BD12 FI31 FI54* **by** *blast*

lemma *OA07*:
$(somega\ ((SPower\ SSkip\ (Suc\ n)))) \subseteq SInf$
by (*metis FI15 FI42 FusionRuleL SOmegaUnroll boolean-algebra-cancel.inf0 compl-le-swap1 inf-commute inf-le2 inf-mono sfmore-def*)

lemma *OA08*:
$SInf \subseteq (somega\ ((SPower\ SSkip\ (Suc\ n))))$
proof −
have *1*: $SInf \cap STrue \cap SAlways\ (\ -STrue \cup ((((SPower\ SSkip\ (Suc\ n)) \cap SMore) \cap SFinite) \cdot STrue)) \subseteq (somega\ (SPower\ SSkip\ (Suc\ n)))$
by (*simp add*: *OA06*)
have *2*: $-\ STrue \cup (SPower\ SSkip\ (Suc\ n) \cap SMore) \cap SFinite \cdot STrue = (SPower\ SSkip\ (Suc\ n) \cap SMore) \cap SFinite \cdot STrue$
by (*simp add*: *strue-def*)
have *21*: $((((SPower\ SSkip\ (Suc\ n)) \cap SMore) \cap SFinite) \cdot STrue) \subseteq SMore$
by (*metis N13 N14 N15 SStar25 SStarInductR inf-le2 le-sup-iff subset-antisym sup-inf-absorb*)
have *3*: $SAlways\ (\ ((((SPower\ SSkip\ (Suc\ n)) \cap SMore) \cap SFinite) \cdot STrue)) \subseteq SAlways\ (SMore)$
using *BD38 21* **by** *auto*
have *4*: $SAlways\ (SMore) \subseteq SInf$
using *FI05* **by** *blast*
have *5*: $SAlways\ (\ -STrue \cup ((((SPower\ SSkip\ (Suc\ n)) \cap SMore) \cap SFinite) \cdot STrue)) \subseteq SInf$
by (*metis 2 3 FI05 subset-trans*)
have *6*: $SInf \subseteq SAlways\ (SMore)$
by (*simp add*: *BD01 FI15 salways-def smore-def*)
have *7*: $(SPower\ SSkip\ (Suc\ n)) \cap SMore \cap SFinite \subseteq SMore \cap SFinite$
using *FI07 FI21* **by** *blast*
have *8*: $SMore \subseteq (SMore \cap SFinite) \cdot STrue$
by (*metis FI19 FI21 ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusionImpSMore SStar19 SStarSkip inf-commute sfmore-def subset-antisym*)
have *9*: $SInf \cap ((((SPower\ SSkip\ (Suc\ n)) \cap SMore) \cap SFinite) \cdot STrue) = ((((SPower\ SSkip\ (Suc\ n)) \cap SMore) \cap SFinite) \cdot SInf)$
by (*metis FI23 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)

have *10*: *SInf* ∩ *SAlways* (((((*SPower SSkip* (*Suc n*)) ∩ *SMore*)∩*SFinite*)·*STrue*)) =
SAlways (((((*SPower SSkip* (*Suc n*)) ∩ *SMore*)∩*SFinite*)·*SInf*))
by (*metis 9 BD29 FI53*)
have *11*: ((((*SPower SSkip* (*Suc n*)) ∩ *SMore*)∩*SFinite*)·*SInf*) = *SInf*
proof (*induct n*)
case *0*
then show *?case*
proof −
have *f1*: (*SFinite*) ∩ *SSkip* = *SSkip*
using *SSkipSFinite* **by** *blast*
have *f2*: ∀ *S*. *S* · *SSkip* ⊆ (*SMore* ∩ *SFinite*) ∨ ¬ *S* ⊆ *SFinite*
using *FI10 FI21 FusionRuleL* **by** (*metis inf-commute sfmore-def*)
have *f3*: (*SEmpty*) · *SSkip* = *SSkip*
using *f1* **by** (*metis FusionSEmptyR inf-commute pwr-0 spower-commutes*)
have *f4*: (*SEmpty*) ⊆ *SFinite*
using *pwr-0 spower-finite* **by** *blast*
have *f5*: (*SSkip*) ⊆ *SFinite*
using *f1* **by** *blast*
have *f6*: (*SSkip*) · (*STrue* · (*SFalse* · *SFalse*)) = *SInf*
by (*metis FI25 FusionAssoc1 FusionSFalse Powerstarhelp2 SMoreImpSSkipFusion SSkipFusion-ImpSMore*
subset-antisym)
have *f7*: (*SSkip*) ∩ *SMore* ∩ *SFinite* = *SSkip*
using *f4 f3 f2* **by** *blast*
have (*SSkip*) · *SInf* = *SInf*
using *f6 f5 f4 f3 f2* **by** (*simp add*: *SFalseFusion sinf-def*)
then show *?thesis*
by (*simp add*: *f7 FusionSEmptyR SSkipSFinite*)
qed
next
case (*Suc n*)
then show *?case*
proof −
have *f1*: *SSkip* · *STrue* = *SMore*
using *SMoreImpSSkipFusion SSkipFusionImpSMore* **by** *blast*
have *f2*: ∀ *S*. *STrue* ∩ *S* = *S*
by (*simp add*: *strue-def*)
have *f3*: ∀ *S*. *SMore* ∩ (*SSkip* · *S*) = *SSkip* · *S*
using *f1*
by (*metis* (*no-types*) *B14 CH06 boolean-algebra-cancel.inf0 compl-bot-eq inf-aci*(*1*) *inf-le2 strue-def*)
then have *f0*: ∀ *S*. *SMore* ∩ *SFinite* ∩ (*SSkip* · *S*) = *SFinite* ∩ (*SSkip* · *S*)
by *blast*
then show *?thesis*
using *f3 f2 f1*
proof −
have *f4*: *SPower* (*SSkip*) (*Suc* (*Suc n*)) ∩ *SMore* ∩ *SFinite*= *SFinite* ∩ (*SSkip* ∩ *SFinite* · *SPower SSkip* (*Suc n*))
by (*metis SSkipSFinite f3 inf-sup-aci*(*1*) *pwr-Suc*)
then have *f5*: *SPower* (*SSkip*) (*Suc* (*Suc n*)) ∩ *SMore* ∩ *SFinite* = *SSkip* ∩ *SFinite* · *SPower SSkip* (*Suc n*)

proof –
have (*SSkip*) ∩ *SFinite* · *SPower SSkip* (*Suc n*) ⊆ *SFinite*
using *pwr-Suc spower-finite* **by** *blast*
then show *?thesis*
using *f4* **by** *blast*
qed
show *?thesis*
by (*metis B14 CH01 FI12 FI23 FI25 FusionAssoc1 Powerstarhelp3 SSkipSFinite Suc f1 f3 inf-commute pwr-Suc sinf-def spower-finite sup-inf-absorb*)
qed
qed
qed
have *12*: *SInf* ⊆ *SAlways* (((((*SPower SSkip* (*Suc n*)) ∩ *SMore*) ∩*SFinite*)·*SInf*))
by (*metis 11 B04 FI53*)
show *?thesis*
using *11 SOmegaWeakCoinduct* **by** (*metis 12 FI53*)
qed

lemma *OA09*:
(*somega* ((*SPower SSkip* (*Suc n*)))) = *SInf*
using *OA07 OA08* **by** *blast*

lemma *OA10*:
(*somega SSkip*) = *SInf*
by (*metis FusionSEmptyR OA09 SSkipSFinite pwr-0 spower.simps*(*2*))

lemma *OA11*:
(*somega STrue*) ⊆ *SInf*
by (*metis FI15 FI42 SOmegaUnroll boolean-algebra.compl-zero boolean-algebra-cancel.inf0 compl-le-swap1 dual-order.refl inf-commute sfmore-def strue-def*)

lemma *OA12*:
SInf ⊆ (*somega STrue*)
proof –
have *1*: (*SAlways* (*SFalse* ∪ ((*STrue* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ *SInf*
by (*metis B04 BD01 BD09 BD35 FI10 FI15 FI19 FI21 ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusionImpSMore boolean-algebra.compl-zero boolean-algebra.de-Morgan-disj boolean-algebra-cancel.inf0 inf-commute salways-def sfalse-def sfmore-def smore-def strue-def*)
have *2*: *SInf* ⊆ (*SAlways* (*SFalse* ∪ ((*STrue* ∩ *SMore*)∩*SFinite*)·*STrue*))
by (*metis 1 B15 BD01 BD09 BD12 BD35 FI10 FI15 FI19 FI21 FI56 ITA.SChopAssoc SMoreImpSSkip-Fusion SSkipFusionImpSMore boolean-algebra.compl-zero boolean-algebra-cancel.inf0 inf.absorb-iff2 inf-commute salways-def sfmore-def smore-def strue-def sup.absorb2*)
have *3*: *SInf* ∩ *STrue* ∩ (*SAlways* (*SFalse* ∪ ((*STrue* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ (*somega STrue*)
by (*metis OA06 compl-bot-eq compl-top-eq sfalse-def strue-def*)
show *?thesis*
using *2 3 strue-def* **by** *fastforce*
qed

lemma *OA13*:
(*somega STrue*) = *SInf*
by (*simp add*: *B04 OA11 OA12*)

lemma *OA14*:
(*somega SMore*) ⊆ *SInf*
by (*metis Int-absorb1 Int-lower2 OA13 SOmegaUnroll SOmegaWeakCoinduct compl-bot-eq inf-top-left strue-def*)

lemma *OA15*:
SInf ⊆ (*somega SMore*)
proof −
have *1*: (*SAlways* (*SFalse* ∪ ((*SMore* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ *SInf*
by (*metis FI05 FI10 FI19 FI21 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusion-ImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero boolean-algebra.de-Morgan-conj inf.idem inf-commute sfalse-def sfmore-def smore-def strue-def subset-antisym*)
have *2*: *SInf* ⊆ (*SAlways* (*SFalse* ∪ ((*SMore* ∩ *SMore*)∩*SFinite*)·*STrue*))
by (*metis BD38 FI10 FI19 FI21 FI25 FI53 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero boolean-algebra.de-Morgan-conj inf.idem inf-commute inf-le1 sfalse-def sfmore-def smore-def strue-def subset-antisym*)
have *3*: *SInf* ∩ *STrue* ∩ (*SAlways* (*SFalse* ∪ ((*SMore* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ (*somega STrue*)
using *OA13* **by** *blast*
show *?thesis*
by (*metis 1 2 Int-absorb1 OA06 boolean-algebra-cancel.inf0 compl-bot-eq compl-top-eq sfalse-def strue-def subset-antisym*)
qed

lemma *OA16*:
(*somega SMore*) = *SInf*
using *OA14 OA15* **by** *blast*

lemma *OA17*:
(*somega SFinite*) ⊆ *SInf*
by (*metis FI07 FI15 FI21 FI42 SOmegaUnroll compl-le-swap1 dual-order.refl inf.orderE sfmore-def*)

lemma *OA18*:
SInf ⊆ (*somega SFinite*)
proof −
have *1*: (*SAlways* (*SFalse* ∪ ((*SFinite* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ *SInf*
by (*metis B14 FI05 FI07 FI10 FI19 FI21 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero boolean-algebra.de-Morgan-conj sfalse-def sfmore-def smore-def strue-def subset-antisym*)
have *2*: *SInf* ⊆ (*SAlways* (*SFalse* ∪ ((*SFinite* ∩ *SMore*)∩*SFinite*)·*STrue*))
by (*metis B14 BD38 FI10 FI19 FI21 FI25 FI53 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero boolean-algebra.de-Morgan-conj inf-le1 sfalse-def sfmore-def smore-def strue-def subset-antisym*)
have *3*: *SInf* ∩ *STrue* ∩ (*SAlways* (*SFalse* ∪ ((*SFinite* ∩ *SMore*)∩*SFinite*)·*STrue*)) ⊆ (*somega STrue*)
using *OA13* **by** *blast*

show *?thesis*
by (*metis 1 2 Int-absorb1 OA06 boolean-algebra-cancel.inf0 compl-bot-eq compl-top-eq sfalse-def strue-def subset-antisym*)
qed

lemma *OA19*:
($somega\ SFinite$) = $SInf$
by (*simp add*: *B04 OA17 OA18*)

lemma *OA20*:
assumes $H \subseteq ((F \cap SMore) \cap SFinite) \cdot H$
shows $H \cap SInf \subseteq (somega\ F)$
using *assms*
using *SOmegaWeakCoinduct* **by** *blast*

lemma *OA21*:
assumes $F \subseteq G$
shows $(somega\ F) \cap SInf \subseteq (somega\ G)$
using *assms*
by (*metis FusionRuleL OA20 SOmegaUnroll inf-mono subset-refl*)

lemma *OA22*:
assumes $F = G$
shows $(somega\ F) = (somega\ G)$
using *assms* **by** *auto*

lemma *OA23*:
$(somega\ (F \cap G)) \cap SInf \subseteq (somega\ F)$
by (*simp add*: *OA21*)

lemma *OA24*:
$(somega\ (F \cap G)) \cap SInf \subseteq (somega\ G)$
by (*simp add*: *OA21*)

lemma *BD60*:
$(SBi\ F) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot G1$
proof −
have *1*: $F \subseteq (-G0 \cup (F \cap G0))$
by *blast*
have *2*: $SBi\ F \subseteq SBi(-G0 \cup (F \cap G0))$
by (*simp add*: *1 BD39*)
have *3*: $SBi(-G0 \cup (F \cap G0)) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot G1$
by (*simp add*: *BD45*)
show *?thesis*
using *2 3* **by** *blast*
qed

lemma *BD61*:
$(SAlways\ H) \cap (F \cdot G) \subseteq F \cdot (H \cap G)$
proof −

have *1*: $H \subseteq (-G \cup (H \cap G))$
  by *blast*
have *2*: *SAlways* $H \subseteq$ *SAlways* $(-G \cup (H \cap G))$
  by (*simp add*: *1 BD38*)
have *3*: *SAlways* $(-G \cup (H \cap G)) \cap (F \cdot G) \subseteq F \cdot (H \cap G)$
  using *BD46* **by** *blast*
show *?thesis*
  using *2 3* **by** *blast*
qed

lemma *BD62*:
  $(SBa\ F) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot (F \cap G1)$
proof –
have *1*: $(SBa\ F) \subseteq (SBi\ F)$
  by (*simp add*: *BD27*)
have *2*: $(SBi\ F) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot G1$
  by (*simp add*: *BD60*)
have *3*: $(SBa\ F) \subseteq (SAlways\ F)$
  by (*simp add*: *BD28*)
have *4*: $(SAlways\ F) \cap ((F \cap G0) \cdot G1) \subseteq (F \cap G0) \cdot (F \cap G1)$
  using *BD61* **by** *blast*
show *?thesis*
  using *1 2 3 4* **by** *blast*
qed

lemma *BD63*:
$(SBa\ F) \cap (G \cdot G1) \subseteq (F \cap G) \cdot ((SBa\ F) \cap G1)$
proof –
have *1*: $(SBa\ F) = (SBa\ (SBa\ F))$
  using *BD44* **by** *blast*
have *2*: $(SBa\ (SBa\ F)) \cap (G \cdot G1) \subseteq G \cdot ((SBa\ F) \cap G1)$
  using *BD28 BD61* **by** *blast*
have *3*: $(SBa\ F) \cap (G \cdot ((SBa\ F) \cap G1)) \subseteq (F \cap G) \cdot ((SBa\ F) \cap G1)$
  using *BD62 FusionRuleR* **by** *blast*
show *?thesis*
  using *1 2 3* **by** *blast*
qed

lemma *OA25*:
  $SBa\ (-F \cup G) \cap SInf \cap (somega\ F) \subseteq (somega\ G)$
proof –
have *1*: $SBa\ (-F \cup G) \cap (((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq$
    $((-F \cup G) \cap ((F \cap SMore) \cap SFinite)) \cdot ((-F \cup G) \cap (somega\ F))$
  by (*simp add*: *BD62*)
have *2*: $(-F \cup G) \cap ((F \cap SMore) \cap SFinite) \subseteq ((G \cap SMore) \cap SFinite)$
  by *auto*
have *3*: $(-F \cup G) \cap (somega\ F) \subseteq (somega\ F)$
  by *auto*
have *4*: $SBa\ (-F \cup G) \cap (((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq ((G \cap SMore) \cap SFinite) \cdot (somega\ F)$
  using *1 2 3 CH06* **by** *blast*

have *5*: $SBa\ (-F \cup G) \cap (((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq$
$((-F \cup G) \cap ((F \cap SMore) \cap SFinite)) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$
using *BD63* **by** *blast*
have *6*: $((-F \cup G) \cap ((F \cap SMore) \cap SFinite)) \cdot (SBa\ (-F \cup G) \cap (somega\ F)) \subseteq$
$((G \cap SMore) \cap SFinite) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$
using *2 FusionRuleL* **by** *blast*
have *7*: $(SBa\ (-F \cup G) \cap (somega\ F)) \subseteq ((G \cap SMore) \cap SFinite) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$
using *5 6 SOmegaUnroll* **by** *blast*
have *8*: $(SBa\ (-F \cup G) \cap (somega\ F)) \cap SInf \subseteq (somega\ G)$
by (*simp add*: *7 OA20*)
show *?thesis*
using *8* **by** *auto*
qed

lemma *OA26*:
$SBa\ ((-F \cup G) \cap (-G \cup F)) \cap SInf \subseteq (-(somega\ G) \cup (somega\ F)) \cap (-(somega\ F) \cup (somega\ G))$
proof −
have *1*: $SBa\ ((-F \cup G) \cap (-G \cup F)) = SBa\ (-F \cup G) \cap SBa\ (-G \cup F)$
by (*simp add*: *BD22 sba-def*)
have *2*: $SBa\ (-F \cup G) \cap SInf \subseteq (-(somega\ F) \cup (somega\ G))$
by (*simp add*: *B19 OA25*)
have *3*: $SBa\ (-G \cup F) \cap SInf \subseteq (-(somega\ G) \cup (somega\ F))$
by (*simp add*: *B19 OA25*)
show *?thesis*
using *1 2 3* **by** *blast*
qed

lemma *OA27*:
$SBa\ F \cap (somega\ G) \cap SInf \subseteq somega\ (F \cap G)$
by (*metis* (*no-types*, *lifting*) *BD63 OA20 SOmegaUnroll inf-assoc*)

lemma *FI57*:
$SInf \cap (((F \cap SMore) \cap SFinite) \cdot G) = ((F \cap SMore) \cap SFinite) \cdot (G \cap SInf)$
using *FI23* **by** *blast*

lemma *FI58*:
$SInf \cap (SAlways\ F) = SAlways\ (F \cap SInf)$
using *BD29 FI53* **by** *blast*

lemma *FI59*:
$SInf \cap (SAlways\ (-F \cup G)) = SAlways\ ((-F \cap SInf) \cup (G \cap SInf))$
by (*simp add*: *FI58 inf-sup-distrib2*)

11.6 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 (*auto simp add*: *fusion-iff chop-nfuse*)

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*: *itl-defs sempty-elim zero-enat-def*)

lemma *sempty-empty-1*:
$SEmpty = (lan\ (LIFT\ empty))$
using *sempty-empty lan-def* **by** *fastforce*

lemma *smore-more*:
$\sigma \in SMore \longleftrightarrow (\sigma \models more)$
using *zero-enat-def* **by** (*auto simp add*: *itl-defs sempty-elim smore-def*)

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*: *one-enat-def itl-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 *sfinite*:
$\sigma \in SFinite \longleftrightarrow (\sigma \models finite)$
by (*simp add*: *sfinite-def sinf-def finite-d-def infinite-d-def fusion-chop sfalse-false-1 strue-true-1*)

lemma *sfinite-1*:
$SFinite = lan(LIFT(finite))$
using *sfinite lan-def* **by** *fastforce*

lemma *and-inter-finite*:
$\sigma \in (((lan\ f) \cap SFinite)) \longleftrightarrow (\sigma \models (f \wedge finite)\ )$
using *sfinite inter-and* **by** *auto*

lemma *and-inter-finite-1*:
$(((lan\ f) \cap SFinite)) = lan(LIFT\ (f \wedge finite)\ )$
by (*simp add*: *inter-and-1 sfinite-1*)

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*:
$(SPower\ (lan\ f)\ n) = (lan\ (LIFT(fpower\ f\ n)))$
proof (*induct n*)
case *0*
then show *?case*
by (*simp add*: *sempty-empty-1 fpower-d-def*)
next
case (*Suc n*)
then show *?case*
by (*simp add*: *and-inter-finite-1 fusion-chop-1 fpower-d-def*)
qed

lemma *spowerstar*:
$\sigma \in SPowerstar\ (lan\ f) \longleftrightarrow \sigma \in SFPowerstar\ (lan\ (f)) \cdot (SEmpty \cup ((lan\ f) \cap SInf))$
by (*simp add*: *spowerstar-def*)

lemma *sstar-spowerstar*:
$\sigma \in SStar\ (lan\ f) \longleftrightarrow \sigma \in SPowerstar\ ((lan\ f) \cap SMore)$
by (*simp add*: *sstar-def*)

lemma *union-exists*:
$\sigma \in (\bigcup\ n.\ SPower\ (lan\ f)\ n) \longleftrightarrow \sigma \in lan(LIFT(\exists\, n.\ fpower\ f\ n))$
by (*simp add*: *spower-chop-power*)

lemma *union-exists-1*:
$(\bigcup\ n.\ SPower\ (lan\ f)\ n) = lan(LIFT(\exists\, n.\ fpower\ f\ n))$
using *union-exists lan-def* **by** *blast*

lemma *sstar-chopstar*:
$\sigma \in (SStar\ (lan\ f)) \longleftrightarrow \sigma \in (lan\ (LIFT(f^{\star})))$
proof –
have *1*: $\sigma \in (SStar\ (lan\ f)) \longleftrightarrow \sigma \in SPowerstar\ ((lan\ f) \cap SMore)$

using *sstar-spowerstar* **by** *blast*
have *2*: $\sigma \in$ *SPowerstar* ((*lan f*) $\cap$ *SMore*) $\longleftrightarrow$
$\sigma \in$ *SFPowerstar* (*lan* (*f*) $\cap$ *SMore*) $\cdot$ (*SEmpty* $\cup$ (((*lan f*) $\cap$ *SMore*) $\cap$ *SInf*))
by (*simp add*: *spowerstar-def*)
have *3*: *SFPowerstar* (*lan* (*f*) $\cap$ *SMore*) = *SFPowerstar*(*lan*(*LIFT*(*f* $\wedge$ *more*)))
by (*simp add*: *and-inter-more-2*)
have *31*: $\bigwedge$*n*. *SPower* (*lan*(*LIFT*(*f* $\wedge$ *more*))) *n* = *lan*(*LIFT*(*fpower* (*f* $\wedge$ *more*) *n*))
using *spower-chop-power* **by** *blast*
have *32*: *SFPowerstar*(*lan*(*LIFT*(*f* $\wedge$ *more*))) = *lan*(*LIFT*(*fpowerstar* (*f* $\wedge$ *more*)))
using *union-exists-1* **by** (*auto simp add*: *sfpowerstar-def fpowerstar-d-def*)
have *4*: (*SEmpty* $\cup$ (((*lan f*) $\cap$ *SMore*) $\cap$ *SInf*)) =
lan(*LIFT*(*empty* $\vee$ ((*f* $\wedge$ *more*) $\wedge$ *inf*)))
by (*metis Morgan and-inter-more-2 finite-d-def inter-and-1 not-negation-1 sempty-empty-1 sfinite-1 sfinite-def union-or-1*)
have *5*: *SFPowerstar* (*lan* (*f*) $\cap$ *SMore*) $\cdot$ (*SEmpty* $\cup$ (((*lan f*) $\cap$ *SMore*) $\cap$ *SInf*)) =
lan(*LIFT*(*powerstar* (*f* $\wedge$ *more*)))
by (*simp add*: *powerstar-d-def 3 32 4 fpowerstar-d-def fusion-chop-1*)
have *6*: *lan*(*LIFT*(*powerstar* (*f* $\wedge$ *more*))) =
lan(*LIFT*(*chopstar f*))
by (*simp add*: *chopstar-d-def*)
show *?thesis*
by (*simp add*: *1 2 5 6*)
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* (*no-types*, *lifting*) *SChopstarEqv chopstar-sstar-1 fusion-chop-1 inter-and-1 sempty-empty-1 smore-more-1 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 *sinf*:
$\sigma \in$ *SInf* $\longleftrightarrow$ ($\sigma \models$ *inf*)
by (*simp add*: *fusion-chop infinite-d-def sfalse-false-1 sinf-def strue-true-1*)

lemma *sinf-1*:
SInf = *lan*(*LIFT*(*inf*))
using *sinf* **by** *fastforce*

lemma *fmore*:
$\sigma \in$ *SFMore* $\longleftrightarrow$ ($\sigma \models$ *fmore*)
by (*metis fmore-d-def inf-commute inter-and-1 interval-lan sfinite-1 sfmore-def smore-more-1*)

lemma *fmore-1*:

SFMore = *lan*(*LIFT*(*fmore*))
using *fmore* **by** *fastforce*

lemma *omega-induct-sem*:
x ∈ − (*lan g*) ∪ (((((*lan f*) ∩ *SMore*)∩*SFinite*)·(*lan g*))) ⟷
(*x* ⊨ *g* ⟶ ((*f* ∧ *more*)∧*finite*);*g*)
by (*simp add*: *fusion-chop-1 inter-and-1 sfinite-1 smore-more-1*)

lemma *omega-induct*:
− (*lan g*) ∪ (((((*lan f*) ∩ *SMore*)∩*SFinite*)·(*lan g*))) =
lan(*LIFT*(*g* ⟶ ((*f* ∧ *more*)∧ *finite*);*g*))
using *omega-induct-sem*[*of - g f*] **by** *fastforce*

lemma *somega-omega-sem-1*:
assumes *x* ⊨ f^{ω}
shows *x* ∈ *somega* (*lan f*)
proof −
have *1*: *x*⊨ f^{ω} ⟶ ((*f* ∧ *more*)∧*finite*);f^{ω}
by (*metis OmegaUnroll intD int-iffD1 inteq-reflection*)
have *2*: *x*∈ −(*lan* (*LIFT*(f^{ω}))) ∪ ((((*lan f*) ∩ *SMore*)∩*SFinite*)·(*lan*(*LIFT*(f^{ω}))))
using *1 omega-induct-sem* **by** *blast*
have *3*: ⋀*x* . *x* ∈ (*lan* (*LIFT*(f^{ω}))) ⟹ *x* ∈ (((((*lan f*) ∩ *SMore*)∩*SFinite*)·(*lan* (*LIFT*(f^{ω})))))
by (*metis OmegaUnroll fusion-chop-1 inteq-reflection inter-and-1 sfinite-1 smore-more-1*)
show *?thesis* **using** *3 SOmegaWeakCoinductsem assms interval-lan* **by** *blast*
qed

lemma *somega-omega-sem-2*:
assumes *x* ∈ *somega* (*lan f*)
shows *x* ⊨ f^{ω}
proof −
have *1*: *x* ∈ − (*somega* (*lan f*)) ∪ (((((*lan f*) ∩ *SMore*)∩*SFinite*)·(*somega* (*lan f*))))
using *SOmegaCases* **by** *blast*
have *2*: ⋀*x*. (λ *x*. *x*∈ *somega* (*lan f*)) *x* ⟹
x ∈ (((((*lan f*) ∩ *SMore*)∩*SFinite*)·(*somega* (*lan f*))))
using *SOmegaCases* **by** *blast*
have *3*: ⋀*x*. (λ *x*. *x*∈ *somega* (*lan f*)) *x* ⟹
((∃ *n*. *f* ((*ntaken n x*)) ∧ *n*>*0* ∧
(λ *x*. *x*∈ *somega* (*lan f*)) ((*ndropn n* (*x*)))))

using *interval-lan*[*of - f*] *somega.cases*[*of -* (*lan f*)]
by (*metis enat-ord-simps*(*2*) *ndropn-nfuse nfinite-nlength-enat ntaken-nfuse the-enat.simps zero-enat-def*)
have *4*: ⋀*x*. (λ *x*. *x*∈ *somega* (*lan f*)) *x* ⟹
x ⊨ ((*f* ∧ *more*)∧*finite*);(λ *x*. *x*∈ *somega* (*lan f*))

by (*metis 3 FMoreSem-var i0-less min-enat-simps*(*2*) *ntaken-nfuse ntaken-nlength somega.simps*)
have *5*: (λ *x*. *x*∈ *somega* (*lan f*)) *x*
by (*simp add*: *assms*)
show *?thesis* **using** *4 5 OmegaWeakCoinductSem*[*of* (λ *x*. *x*∈ *somega* (*lan f*)) *f*]
by (*metis Int-iff Prop10 intI inteq-reflection inter-and-1 interval-lan*)

qed

lemma *somega-omega*:
$x \in somega\ (lan\ f) \longleftrightarrow (x \models f^{\omega})$
using *somega-omega-sem-1 somega-omega-sem-2* **by** *blast*

lemma *somega-omega-1*:
$somega\ (lan\ f) = lan(LIFT(\ f^{\omega}))$
using *somega-omega* **by** *fastforce*

end

12 Until operator for Finite and Infinite Intervals

theory *Until*
imports *Semantics SChopTheorems*
begin

This theory introduces the weak and strong versions of the until operator. The theorems from [11] are proven in a mostly deductive style.

12.1 Definitions

definition *until-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where $until\text{-}d\ F\ G \equiv \lambda s.\ (\ (\exists\ k\ .\ k \leq nlength\ s \wedge (\ (ndropn\ k\ s\ ) \models G) \wedge$
$(\forall j.\ j < k\ \longrightarrow\ (\ (ndropn\ j\ s\ ) \models F)\ ))\ )$

syntax
-until-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{U}$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-until-d :: [*lift*,*lift*] ⇒ *lift* ((- *until* -) [*84*,*84*] *83*)

translations
-until-d ⇌ *CONST until-d*

definition *suntil-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where $suntil\text{-}d\ F\ G \equiv LIFT(\bigcirc(F\ \mathcal{U}\ G))$

definition *wait-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where $wait\text{-}d\ F\ G \equiv LIFT(\Box\ F \vee F\ \mathcal{U}\ G)$

definition *release-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where $release\text{-}d\ F\ G \equiv LIFT(\neg((\neg\ F)\ \mathcal{U}\ (\neg\ G)))$

syntax
-wait-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{W}$ -) [*84*,*84*] *83*)
-release-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{R}$ -) [*84*,*84*] *83*)
-suntil-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{U}^s$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-wait-d :: [*lift*,*lift*] ⇒ *lift* ((- *wait* -) [*84*,*84*] *83*)
-release-d :: [*lift*,*lift*] ⇒ *lift* ((- *release* -) [*84*,*84*] *83*)
-suntil-d :: [*lift*,*lift*] ⇒ *lift* ((- *suntil* -) [*84*,*84*] *83*)

translations
-wait-d ⇌ *CONST wait-d*
-release-d ⇌ *CONST release-d*
-suntil-d ⇌ *CONST suntil-d*

definition *srelease-d* :: $('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$
where *srelease-d* $F\ G \equiv LIFT(\neg((\neg\ F)\ \mathcal{W}\ (\neg\ G)))$

syntax
-srelease-d :: [*lift*,*lift*] ⇒ *lift* ((- $\mathcal{M}$ -) [*84*,*84*] *83*)

syntax (*ASCII*)
-srelease-d :: [*lift*,*lift*] ⇒ *lift* ((- *srelease* -) [*84*,*84*] *83*)

translations
-srelease-d ⇌ *CONST srelease-d*

12.2 Axioms

12.2.1 NextUntil

lemma *NextUntilsema*:
assumes $(\sigma \models \bigcirc(f\ \mathcal{U}\ g))$
shows $(\sigma \models (\bigcirc\ f)\ \mathcal{U}\ (\bigcirc\ g))$
proof −
have *0*: $0 < nlength\ \sigma\ \wedge$
$(\exists k.\ k \le nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ f\ (\ (ndropn\ (Suc\ j)\ \sigma))\ ))$
using *assms zero-enat-def* **by** (*auto simp add*: *next-defs until-d-def ndropn-ndropn*)
have *1*: $0 < nlength\ \sigma$
using *0* **by** *auto*
have *2*: $(\exists k.\ k \le nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ f\ ((\ (ndropn\ (Suc\ j)\ \sigma))\ )))$
using *0* **by** *auto*
obtain *k* **where** *3*: $k \le nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((\ (ndropn\ (Suc\ k)\ \sigma))) \wedge$
$(\forall j<k.\ f\ ((\ (ndropn\ (Suc\ j)\ \sigma))\ ))$
using *2* **by** *auto*
have *4*: $g\ ((\ (ndropn\ (Suc\ k)\ \sigma)))$
using *3* **by** *auto*
have *5*: $k \le\ nlength\ \sigma$
using *3 0*
by (*metis diff-le-self dual-order.order-iff-strict enat-ile enat-ord-simps*(*1*) *idiff-enat-enat less-le-trans ndropn-nlength not-less*)
have *6*: $0 < nlength\ (ndropn\ k\ \sigma)$
using *1 3*

by (*metis gr-zeroI iless-Suc-eq is-NNil-ndropn leD le-numeral-extra*(*3*) *ndropn-0* *ndropn-Suc-conv-ndropn nlength-NCons zero-enat-def*)
have *7*: $(\forall j<k.\ 0 < nlength\ (ndropn\ j\ \sigma) \wedge f\ ((\ (ndropn\ (Suc\ j)\ \sigma))))$
using *3 5*
by (*metis enat-ord-simps*(*2*) *is-NNil-ndropn ndropn-0 not-less order.trans zero-le*)
have *71*: $nlength\ \sigma - enat\ k \neq enat\ 0$
using *6 zero-enat-def* **by** *auto*
have *72*: $(\forall j<k.\ nlength\ \sigma - enat\ j \neq enat\ 0 \wedge f\ (ndropn\ (Suc\ j)\ \sigma))$
using *7 zero-enat-def* **by** *auto*
have *8*: $\exists k.\ k\leq nlength\ \sigma \wedge$
$nlength\ \sigma - enat\ k \neq enat\ 0 \wedge$
$g\ ((\ (ndropn\ (Suc\ k)\ \sigma))) \wedge$
$(\forall j<k.\ nlength\ \sigma - enat\ j \neq enat\ 0 \wedge f\ (ndropn\ (Suc\ j)\ \sigma))$
using *3 5 6 71 7 72* **by** *blast*
from *8* **show** *?thesis*
by (*simp add*: *next-defs until-d-def ndropn-ndropn*)
qed

lemma *NextUntilsemb*:
assumes $(\sigma \models (\bigcirc f)\ \mathcal{U}\ (\bigcirc g))$
shows $(\ \sigma \models \bigcirc(f\ \mathcal{U}\ g))$
proof –
have *1*: $\exists k.\ k\leq nlength\ \sigma \wedge 0 < nlength\ (ndropn\ k\ \sigma) \wedge g\ (\ (ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ j < nlength\ \sigma \wedge f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *assms*
by (*auto simp add*: *next-defs until-d-def ndropn-ndropn*)
(*metis is-NNil-ndropn le-zero-eq ndropn-0 ndropn-nlength not-less zero-enat-def*)
obtain *k* **where** *2*: $k\leq nlength\ \sigma \wedge 0 < nlength\ (ndropn\ k\ \sigma) \wedge$
$g\ (\ (ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ j < nlength\ \sigma \wedge f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *1* **by** *auto*
have *3*: $0 < nlength\ \sigma$
using *2* **by** *auto*
have *4*: $k\leq nlength\ (ndropn\ (Suc\ 0)\ \sigma)$
by *auto*
(*metis 2 3 add.commute add.right-neutral enat-min iless-Suc-eq ndropn-0* *ndropn-Suc-conv-ndropn ndropn-nlength nlength-NCons zero-enat-def*)
have *5*: $g\ (\ (ndropn\ (Suc\ k)\ \sigma))$
using *2* **by** *auto*
have *6*: $(\forall j<k.\ j < nlength\ \sigma \wedge f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *2* **by** *blast*
have *7*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ k\leq nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ (\ (ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ f\ (\ (ndropn\ (Suc\ j)\ \sigma))\ )\ )$
using *2 3 4* **by** *blast*
from *7* **show** *?thesis* **by** (*auto simp*: *next-defs until-d-def zero-enat-def ndropn-ndropn*)
qed

lemma *NextUntilsem*:
$\sigma \models \bigcirc(f\ \mathcal{U}\ g) = (\bigcirc f)\ \mathcal{U}\ (\bigcirc g)$

using *NextUntilsema NextUntilsemb* **using** *unl-lift2* **by** *blast*

lemma *NextUntil*:
$\vdash \bigcirc(f\ \mathcal{U}\ g) = (\bigcirc\ f)\ \mathcal{U}\ (\bigcirc\ g)$
using *NextUntilsem Valid-def* **by** *blast*

12.2.2 UntilNextUntil

lemma *UntilNextUntilsema*:
assumes $0 < nlength\ \sigma \wedge$
$(\exists k.\ 0 < k \wedge k \leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma))))$
shows $(\ \sigma) \models \bigcirc\ (\ f\ \mathcal{U}\ g)$
proof −
have *1*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ 0<k \wedge k \leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma))))$
using *assms* **by** *auto*
have *3*: $(\exists k.\ 0<k \wedge k \leq nlength\ \sigma \wedge g\ ((\ (ndropn\ k\ \sigma))) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma)))\ )$
using *1* **by** *auto*
obtain *k* **where** *4*: $0<\ (Suc\ k) \wedge (Suc\ k) \leq nlength\ \sigma \wedge g\ (\ (ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j.\ 0 < j \wedge j < (Suc\ k) \longrightarrow f\ (\ (ndropn\ j\ \sigma)))$
using *3* **by** (*metis Suc-pred*)
have *5*: $k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma)$
by (*metis 4 One-nat-def add.commute co.enat.sel*(*2*) *eSuc-enat epred-conv-minus le-cases min-absorb1 ndropn-nlength ntaken-all ntaken-ndropn-swap-nlength ntaken-nlength one-enat-def plus-1-eSuc*(*2*) *plus-1-eq-Suc*)
have *6*: $g\ (\ (ndropn\ (Suc\ k)\ \sigma))$
using *4* **by** *auto*
have *7*: $(\forall j<k.\ f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *4* **by** *blast*
have *8*: $(\exists k.\ k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge (\forall j<k.\ f\ ((ndropn\ (Suc\ j)\ \sigma))))$
using *4 5* **by** *blast*
have *9*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ k \leq nlength\ (ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge (\forall j<k.\ f\ ((ndropn\ (Suc\ j)\ \sigma))))$
using *1 8* **by** *blast*
from *9* **show** *?thesis* **by** (*auto simp add*: *next-defs until-d-def zero-enat-def ndropn-ndropn*)
qed

lemma *UntilNextUntilsemb*:
assumes $\sigma \models \bigcirc\ (\ f\ \mathcal{U}\ g)$
shows $0 < nlength\ \sigma \wedge$
$(\exists k.\ 0 < k \wedge k \leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge (\forall j.\ 0 < j \wedge j < k \longrightarrow f\ ((ndropn\ j\ \sigma))))$
proof −
have *1*: $0 < nlength\ \sigma \wedge$
$(\exists k.\ k \leq nlength(ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge (\forall j<k.\ f\ ((ndropn\ (Suc\ j)\ \sigma))))$
using *assms* **by** (*auto simp add*: *next-defs until-d-def ndropn-ndropn*) (*simp add*: *zero-enat-def*)
have *2*: $(\exists k.\ k \leq nlength(ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge (\forall j<k.\ f\ ((ndropn\ (Suc\ j)\ \sigma))))$
using *1* **by** *auto*
obtain *k* **where** *3*: $k \leq nlength(ndropn\ (Suc\ 0)\ \sigma) \wedge g\ ((ndropn\ (Suc\ k)\ \sigma)) \wedge$
$(\forall j<k.\ f\ (\ (ndropn\ (Suc\ j)\ \sigma)))$
using *2* **by** *auto*

have *4*: *0*< (*Suc k*)
by *simp*
have *5*: *g* ((*ndropn* (*Suc k*) *σ*))
using *3* **by** *auto*
have *6*: (*Suc k*) ≤ *nlength σ*
using *3* **by** *auto*
(*metis 1 dual-order.eq-iff eSuc-enat is-NNil-ndropn le-cases ndropn-0 ndropn-Suc-conv-ndropn ndropn-ndropn ndropn-nlength nlength-NCons plus-1-eq-Suc zero-enat-def*)
have *7*: (∀ *j*. *0* < *j* ∧ *j* < (*Suc k*) ⟶ *f* ((*ndropn j σ*)))
using *3 less-Suc-eq-0-disj* **by** *auto*
have *8*: (∃ *k*. *0*<*k* ∧ *k* ≤ *nlength σ* ∧ *g* ((*ndropn k σ*)) ∧ (∀ *j*. *0* < *j* ∧ *j* < *k* ⟶ *f* ((*ndropn j σ*))))
using *3 6 7* **by** *blast*
show *?thesis* **using** *1 8* **by** *blast*
qed

lemma *UntilNextUntilsem*:
(*σ* ⊨ ○ (*f* 𝒰 *g*)) =
(*0* < *nlength σ* ∧
(∃ *k*. *0*< *k* ∧ *k* ≤ *nlength σ* ∧ *g* ((*ndropn k σ*)) ∧ (∀ *j*. *0* < *j* ∧ *j* < *k* ⟶ *f* ((*ndropn j σ*)))))
using *UntilNextUntilsema*[*of σ g f*] *UntilNextUntilsemb*[*of f g σ*] **by** *meson*

lemma *UntilNextUntilsem1*:
(*σ* ⊨ *f* 𝒰 *g*) = (*σ* ⊨ (*g* ∨ (*f* ∧ ○(*f* 𝒰 *g*))))
unfolding *UntilNextUntilsem*
proof
assume *a*: (*σ* ⊨ *f* 𝒰 *g*)
show (*σ* ⊨ *g* ∨
f ∧
(*λσ*. *0* < *nlength σ* ∧
(∃ *k*. *0*<*k* ∧ *k* ≤ *nlength σ* ∧ *g*((*ndropn k σ*)) ∧ (∀ *j*. *0* < *j* ∧ *j* < *k* ⟶ *f*((*ndropn j σ*))))
))
using *a* **by** (*simp add*: *until-d-def*) (*metis enat-0-iff*(*2*) *i0-less ndropn-0 neq0-conv not-le*)
next
next
assume *b*: (*σ* ⊨ *g* ∨
f ∧
(*λσ*. *0* < *nlength σ* ∧
(∃ *k*. *0*<*k* ∧ *k* ≤ *nlength σ* ∧ *g*((*ndropn k σ*)) ∧ (∀ *j*. *0* < *j* ∧ *j* < *k* ⟶ *f*((*ndropn j σ*))))))
show (*σ* ⊨ *f* 𝒰 *g*)
using *b* **by** (*simp add*: *until-d-def*) (*metis i0-lb linorder-cases ndropn-0 not-less-zero zero-enat-def*)
qed

lemma *UntilNextUntil*:
⊢ *f* 𝒰 *g* = (*g* ∨ (*f* ∧ ○(*f* 𝒰 *g*)))
by (*simp add*: *UntilNextUntilsem1 Valid-def*)

12.2.3 NotUntilFalse

lemma *NotUntilFalse*:
⊢ ¬ (*f* 𝒰 #*False*)

by (*simp add*: *intI until-d-def*)

12.2.4 UntilOrDist

lemma *UntilOrDistsem*:
$\sigma \models f\ \mathcal{U}\ (g \lor h) = (\ f\ \mathcal{U}\ g \lor f\ \mathcal{U}\ h)$
by (*auto simp add*: *until-d-def*)

lemma *UntilOrDist*:
$\vdash f\ \mathcal{U}\ (g \lor h) = (\ f\ \mathcal{U}\ g \lor f\ \mathcal{U}\ h)$
using *UntilOrDistsem Valid-def* **by** *blast*

12.2.5 UntilRightDistOr

lemma *UntilRightDistOr*:
$\vdash f\ \mathcal{U}\ h \lor g\ \mathcal{U}\ h \longrightarrow (f \lor g)\ \mathcal{U}\ h$
by (*auto simp add*: *Valid-def until-d-def*)

12.2.6 UntilLeftDistAnd

lemma *UntilLeftDistAnd*:
$\vdash f\ \mathcal{U}\ (g \land h) \longrightarrow f\ \mathcal{U}\ g \land f\ \mathcal{U}\ h$
by (*auto simp add*: *Valid-def until-d-def*)

12.2.7 UntilAndDist

lemma *UntilAndDistsem*:
$\sigma \models (f \land g)\ \mathcal{U}\ h = ((f\ \mathcal{U}\ h) \land (g\ \mathcal{U}\ h))$
by (*auto simp add*: *until-d-def*)
(*metis* (*no-types*, *lifting*) *less-le-trans not-less-iff-gr-or-eq order.order-iff-strict*)

lemma *UntilAndDist*:
$\vdash (f \land g)\ \mathcal{U}\ h = ((f\ \mathcal{U}\ h) \land (g\ \mathcal{U}\ h))$
using *UntilAndDistsem Valid-def* **by** *blast*

12.2.8 untilNotImp

lemma *UntilNotImp*:
$\vdash f\ \mathcal{U}\ g \land (\neg\ g)\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ h$
by (*simp add*: *Valid-def until-d-def*)
(*metis not-less-iff-gr-or-eq order.strict-trans*)

12.2.9 UntilUntil

lemma *UntilUntilsem*:
$(\sigma \models f\ \mathcal{U}\ g)\ = (\sigma \models f\ \mathcal{U}\ (f\ \mathcal{U}\ g))$
proof *auto*
show $\sigma \models (f\ \mathcal{U}\ g) \implies \sigma \models (f\ \mathcal{U}\ (f\ \mathcal{U}\ g))$
by (*simp add*: *until-d-def*)
(*metis enat-add-sub-same enat-le-plus-same*(*1*) *enat-ord-code*(*4*) *enat-ord-simps*(*4*) *gen-nlength-def ndropn-0 nlength-code not-less-zero*)
show $\sigma \models (f\ \mathcal{U}\ (f\ \mathcal{U}\ g)) \implies \sigma \models (f\ \mathcal{U}\ g)$

proof −
assume *a*: $\sigma \models (f\ \mathcal{U}\ (f\ \mathcal{U}\ g))$
show $\sigma \models (f\ \mathcal{U}\ g)$
proof −
have *1*: $\exists\, k.\ enat\ k \le nlength\ \sigma \wedge$
$(\exists\, ka.\ enat\ ka \le nlength\ (ndropn\ k\ \sigma) \wedge g\ (ndropn\ ka\ (ndropn\ k\ \sigma)) \wedge$
$(\forall\, j{<}ka.\ f\ (ndropn\ j\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall\, j{<}k.\ f\ (ndropn\ j\ \sigma))$
using *a* **unfolding** *until-d-def* **by** *blast*
obtain *k* **where** *2*:
$enat\ k \le nlength\ \sigma \wedge$
$(\exists\, ka.\ enat\ ka \le nlength\ (ndropn\ k\ \sigma) \wedge g\ (ndropn\ ka\ (ndropn\ k\ \sigma)) \wedge$
$(\forall\, j{<}ka.\ f\ (ndropn\ j\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall\, j{<}k.\ f\ (ndropn\ j\ \sigma))$
using *1* **by** *auto*
have *3*: $(\exists\, ka.\ enat\ ka \le nlength\ (ndropn\ k\ \sigma) \wedge g\ (ndropn\ ka\ (ndropn\ k\ \sigma)) \wedge$
$(\forall\, j{<}ka.\ f\ (ndropn\ j\ (ndropn\ k\ \sigma))))$
using *2* **by** *auto*
obtain *ka* **where** *4*:
$enat\ ka \le nlength\ (ndropn\ k\ \sigma) \wedge g\ (ndropn\ ka\ (ndropn\ k\ \sigma)) \wedge$
$(\forall\, j{<}ka.\ f\ (ndropn\ j\ (ndropn\ k\ \sigma)))$
using *3* **by** *auto*
have *41*: $enat\ ka \le nlength\ \sigma - (enat\ k)$
using *4* **by** *auto*
have *5*: $enat\ (ka{+}k) \le nlength\ \sigma$
using *2 41* **by** *auto*
(*metis add.commute antisym-conv2 enat.simps*(*3*) *enat-add-sub-same enat-min le-iff-add less-imp-le order-refl plus-enat-simps*(*1*))
have *6*: $g\ (ndropn\ (ka{+}k)\ \sigma)$
by (*metis 4 add.commute ndropn-ndropn*)
have *7*: $(\forall\, j{<}(ka{+}k).\ f\ (ndropn\ j\ \sigma))$
by (*metis 2 4 add-diff-inverse-nat less-diff-conv2 linorder-not-less ndropn-ndropn*)
have *8*: $\exists\, k.\ k \le nlength\ \sigma \wedge g\ (ndropn\ k\ \sigma) \wedge (\forall\, j{<}k.\ f\ (ndropn\ j\ \sigma))$
using *5 6 7* **by** *blast*
show *?thesis* **unfolding** *until-d-def* **by** (*simp add*: *8*)
qed
qed
qed

lemma *UntilUntil*:
$\vdash f\ \mathcal{U}\ g = f\ \mathcal{U}\ (f\ \mathcal{U}\ g)$
using *UntilUntilsem* **by** *fastforce*

12.2.10 UntilRightor

lemma *UntilRightOr*:
$\vdash f\ \mathcal{U}\ (\ g\ \mathcal{U}\ h\ ) \longrightarrow (f \vee g)\ \mathcal{U}\ h$
proof (*auto simp add*: *Valid-def until-d-def ndropn-ndropn*)
fix *w* :: *'a nellist*
fix *k*

fix *ka*
assume *a0*: $enat\ k \leq nlength\ w$
assume *a1*: $\forall j<k.\ f\ (ndropn\ j\ w)$
assume *a2*: $enat\ ka \leq nlength\ w - enat\ k$
assume *a3*: $h\ (ndropn\ (k + ka)\ w)$
assume *a4*: $\forall j<ka.\ g\ (ndropn\ (k + j)\ w)$
show $\exists k.\ enat\ k \leq nlength\ w \wedge h\ (ndropn\ k\ w) \wedge (\forall j<k.\ f\ (ndropn\ j\ w) \vee g\ (ndropn\ j\ w))$
proof −
have *1*: $ka+k \leq nlength\ w$
by (*metis a0 a2 add.commute dual-order.order-iff-strict enat.simps*(*3*) *enat-add-sub-same enat-less-enat-plusI2 less-eqE plus-enat-simps*(*1*))
have *2*: $h\ (ndropn\ (ka+k)\ w)$
using *a3* **by** (*simp add*: *add.commute*)
have *3*: $(\forall j<(ka+k).\ f\ (ndropn\ j\ w) \vee g\ (ndropn\ j\ w))$
by (*metis a1 a4 add-diff-inverse-nat less-diff-conv2 not-less*)
show *?thesis* **using** *1 2 3* **by** *blast*
qed
qed

12.2.11 UntilRightAnd

lemma *UntilRightAndsem*:
assumes $(\sigma \models f\ \mathcal{U}\ (g \wedge h))$
shows $(\sigma \models (f\ \mathcal{U}\ g)\ \mathcal{U}\ h)$
proof −
have *1*: $\exists k.\ k\leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge h\ ((ndropn\ k\ \sigma)) \wedge (\forall j<k.\ f\ ((ndropn\ j\ \sigma)))$
using *assms* **by** (*simp add*: *until-d-def*)
obtain *k* **where** *2*: $k\leq nlength\ \sigma \wedge g\ ((ndropn\ k\ \sigma)) \wedge h\ ((ndropn\ k\ \sigma)) \wedge (\forall j<k.\ f\ ((ndropn\ j\ \sigma)))$
using *1* **by** *auto*
have *3*: $h\ (\ (ndropn\ k\ \sigma))$
using *2* **by** *auto*
have *4*: $k\leq nlength\ \sigma$
using *2* **by** *auto*
have *5*: $(\forall j<k.$
$\exists ka.\ ka\leq nlength\ (ndropn\ j\ \sigma) \wedge g\ ((ndropn\ (ka + j)\ \sigma)) \wedge (\forall ja<ka.\ f\ ((ndropn\ (ja + j)\ \sigma))))$
proof *auto*
fix *j*
assume *a0*: $j < k$
show $\exists ka.\ enat\ ka \leq nlength\ \sigma - enat\ j \wedge g\ (ndropn\ (ka + j)\ \sigma) \wedge (\forall ja<ka.\ f\ (ndropn\ (ja + j)\ \sigma))$
proof −
have *51*: $k-j \leq nlength\ (ndropn\ j\ \sigma)$
using *4 a0*
by (*metis enat-minus-mono1 idiff-enat-enat ndropn-nlength*)
have *52*: $g\ (\ (ndropn\ ((k-j) + j)\ \sigma))$
by (*simp add*: *2 a0 less-imp-le-nat*)
have *53*: $(\forall ja<(k-j).\ f\ (\ (ndropn\ (ja + j)\ \sigma)))$
using *2 less-diff-conv* **by** *blast*
show *?thesis*
using *51 52 53* **by** *auto*
qed

qed
have *6*: $\exists k.\ k \leq nlength\ \sigma\ \wedge$
$h\ (\ (ndropn\ k\ \sigma))\ \wedge$
$(\forall j{<}k.\ \exists k \leq nlength\ (ndropn\ j\ \sigma).\ g\ ((ndropn\ (k + j)\ \sigma))\ \wedge\ (\forall ja{<}k.\ f\ ((ndropn\ (ja + j)\ \sigma))))$
using *2 5* **by** *blast*
from *6* **show** *?thesis* **by** (*simp add*: *until-d-def ndropn-ndropn add.commute*)
qed

lemma *UntilRightAnd*:
$\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow (f\ \mathcal{U}\ g)\ \mathcal{U}\ h$
using *UntilRightAndsem Valid-def* **by** *auto*

12.2.12 DiamondEqvTrueUntil

lemma *DiamondEqvTrueUntil*:
$\vdash \Diamond\ f = \#True\ \mathcal{U}\ f$
by (*simp add*: *Valid-def sometimes-defs until-d-def*)

12.2.13 TrueUntilImpNotUntil

lemma *nellist-ndropn-first-upto*:
assumes $(\exists i \leq\ k.\ f\ (\ (ndropn\ i\ xs)))$
shows $(\exists i \leq\ k.\ f\ (\ (ndropn\ i\ xs))\ \wedge\ (\forall j{<}i\ .\ \neg\ (f\ (\ (ndropn\ j\ xs)))))$
using *assms*
proof (*induct k arbitrary*: *xs*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
by (*metis le-Suc-eq less-Suc-eq-le*)
qed

lemma *nellist-ndropn-first*:
assumes $(\exists i \leq nlength\ xs.\ f\ (\ (ndropn\ i\ xs)))$
shows $(\exists i \leq nlength\ xs.\ f\ (\ (ndropn\ i\ xs))\ \wedge\ (\forall j.\ j{<}i \longrightarrow \neg\ (f\ (\ (ndropn\ j\ xs)))))$
proof (*cases nfinite xs*)
case *True*
then show *?thesis* **using** *assms nellist-ndropn-first-upto*[*of - f xs*] *nfinite-nlength-enat*[*of xs*]
by *force*
next
case *False*
then show *?thesis* **using** *assms nellist-ndropn-first-upto*[*of - f xs*]
proof –
assume *a1*: $\bigwedge k.\ \exists i \leq k.\ f\ (ndropn\ i\ xs) \Longrightarrow \exists i \leq k.\ f\ (ndropn\ i\ xs)\ \wedge\ (\forall j{<}i.\ \neg\ f\ (ndropn\ j\ xs))$
obtain *nn* :: *nat* **where**
f2: $f\ (ndropn\ nn\ xs)\ \wedge\ enat\ nn \leq nlength\ xs$
using *assms* **by** *blast*
then have $\forall e.\ \neg\ e \leq enat\ nn\ \vee\ e \leq nlength\ xs$
by *force*
then show *?thesis*

using *f2 a1* **by** (*meson enat-ord-simps*(*1*) *less-imp-le-nat*)
qed
qed

lemma *NotSuffixFirstfinite*:
assumes $(\exists\, n \le nlength\ xs.\ \neg\ f\ (\ (ndropn\ n\ xs)))$
shows $(\exists\, n \le nlength\ xs.\ \neg\ f\ (\ (ndropn\ n\ xs)) \wedge (\forall\, k.\ k<n \longrightarrow f\ (\ (ndropn\ k\ xs))))$
using *assms nellist-ndropn-first*[*of xs LIFT*(¬ *f*)] **by** *auto*

lemma *TrueUntilImpNotUntilsem*:
assumes $\sigma\ \models \#True\ \mathcal{U}\ g$
shows $\sigma\ \models (\neg g)\ \mathcal{U}\ g$
using *assms*
by (*simp add*: *until-d-def nellist-ndropn-first*)

lemma *TrueUntilImpNotUntil*:
$\vdash \#True\ \mathcal{U}\ g \longrightarrow (\neg g)\ \mathcal{U}\ g$
by (*simp add*: *intI nellist-ndropn-first until-d-def*)

12.2.14 WaitNotDistUntil

lemma *WaitNotDistUntilsem1*:
assumes $(\sigma \models \neg(f\ \mathcal{W}\ g))$
shows $(\ \sigma \models ((\neg\ g)\ \mathcal{U}\ ((\neg\ f) \wedge (\neg g))))$
proof –
have *1*: $(\forall\, k.\ g\ (\ (ndropn\ k\ \sigma)) \longrightarrow k \le nlength\ \sigma \longrightarrow (\exists\, j<k.\ \neg\ f\ (\ (ndropn\ j\ \sigma)))) \wedge$
$(\exists\, n \le nlength\ \sigma.\ \neg\ f\ (\ (ndropn\ n\ \sigma)))$
using *assms* **by** (*simp add*: *wait-d-def until-d-def always-defs*)
have *2*: $(\forall\, k.\ k \le nlength\ \sigma \longrightarrow \neg\ g\ (\ (ndropn\ k\ \sigma)) \vee (\exists\, j<k.\ \neg\ f\ (\ (ndropn\ j\ \sigma))))$
using *1* **by** *auto*
have *3*: $(\exists\, n \le nlength\ \sigma.\ \neg\ f\ (\ (ndropn\ n\ \sigma)))$
using *1* **by** *auto*
obtain *n* **where** *4*: $n \le\ nlength\ \sigma \wedge \neg\ f\ (\ (ndropn\ n\ \sigma)) \wedge$
$(\forall\ \ k\ <n\ .\ f\ (\ (ndropn\ k\ \sigma)))$
using *3* **using** *NotSuffixFirstfinite* **by** *blast*
have *16*: $n \le nlength\ \sigma$
by (*simp add*: *4*)
have *17*: $\neg\ g\ (\ (ndropn\ n\ \sigma))$
using *1 4* **by** *blast*
have *18*: $(\forall\, j<n.\ \neg\ g\ (\ (ndropn\ j\ \sigma)))$
by (*metis 2 4 dual-order.strict-iff-order dual-order.strict-trans1 enat-ord-simps*(*2*))
have *19*: $\exists\, k \le nlength\ \sigma.\ \neg\ f\ (\ (ndropn\ k\ \sigma)) \wedge \neg\ g\ (\ (ndropn\ k\ \sigma)) \wedge (\forall\, j<k.\ \neg\ g\ (\ (ndropn\ j\ \sigma)))$
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 *WaitNotDistUntil*:
$\vdash (\neg(f\ \mathcal{W}\ g)) = ((\neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg g))$
using *WaitNotDistUntilsem* *Valid-def* **by** (*metis*)

12.2.15 UntilInduction

lemma *LFPUntilsem1*:
assumes $\forall\, n \le nlength\ \sigma.$
$(g\ (\ (ndropn\ n\ \sigma)) \longrightarrow h\ (\ (ndropn\ n\ \sigma))) \wedge$
$(f\ (\ (ndropn\ n\ \sigma)) \wedge n < nlength\ \sigma \wedge h\ (\ (ndropn\ (Suc\ n)\ \sigma)) \longrightarrow$
$h\ (\ (ndropn\ n\ \sigma)))$
$k \le nlength\ \sigma$
$g\ (\ (ndropn\ k\ \sigma))$
$\forall\, j<k.\ f\ (\ (ndropn\ j\ \sigma))$
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 –
have *1*: $g\ (ndropn\ (Suc\ k)\ \sigma) \longrightarrow h\ \ (ndropn\ (Suc\ k\ )\ \sigma)$
using *Suc.prems*(*1*) *Suc.prems*(*2*) **by** *blast*
have *2*: $(f\ (\ (ndropn\ k\ \sigma)) \wedge k < nlength\ \sigma \wedge h\ (\ (ndropn\ (Suc\ k)\ \sigma)) \longrightarrow$
$h\ (\ (ndropn\ k\ \sigma)))$
by (*simp add*: *Suc.prems*(*1*) *less-le-not-le*)
have *3*: $k < nlength\ \sigma$
using *Suc.prems*(*2*) *Suc-ile-eq* **by** *auto*
have *4*: $f\ (\ (ndropn\ k\ \sigma))$
by (*simp add*: *Suc.prems*(*4*))
have *5*: $h\ (\ (ndropn\ k\ \sigma))$
using *1 2 3 4 Suc.prems*(*3*) **by** *blast*
have *6*: $h\ (\ (ndropn\ 0\ \sigma))$
using *zero-induct*[*of* $\lambda k\ .\ h\ (ndropn\ k\ \ \sigma)\ k$]
3 5 Suc.prems nellist-ndropn-first[*of* σ *h*]
by (*metis Suc-ile-eq less-Suc-eq-0-disj less-imp-le not-less-eq*)
show *?thesis*
using *6* **by** *auto*
qed
qed

lemma *LFPUntilsem*:
$\sigma \models \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h)$
using *LFPUntilsem1*[*of* σ *g h f*]
by (*auto simp add*: *always-defs next-defs until-d-def ndropn-ndropn*)
(*metis canonically-ordered-monoid-add-class.lessE enat.simps*(*3*) *enat-add-sub-same zero-enat-def*)

lemma *LFPUntil*:
$\vdash \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h)$
using *LFPUntilsem Valid-def*
by (*metis*)

lemma *UntilInduction-a*:
$\vdash \Box(f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow (f \longrightarrow \Box\ g \vee g\ \mathcal{U}\ h)$
proof −
have *1*: $\vdash (\Box\ g \vee g\ \mathcal{U}\ h) = g\ \mathcal{W}\ h$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash (f \longrightarrow \Box\ g \vee g\ \mathcal{U}\ h) = (\ (\neg\ h)\ \mathcal{U}\ (\neg\ g \wedge \neg h) \longrightarrow \neg\ f)$
using *1 WaitNotDistUntil* **by** *fastforce*
have *3*: $\vdash \Box(\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f)))\longrightarrow \neg f) \longrightarrow (\ (\neg\ h)\ \mathcal{U}\ (\neg\ g \wedge \neg h) \longrightarrow \neg\ f)$
using *LFPUntil* **by** *blast*
have *4*: $\vdash (f \longrightarrow ((\bigcirc\ f) \wedge g) \vee h) \longrightarrow (\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f)))\longrightarrow \neg f)$
using *NextImpNotNextNot*[*of f*] **by** *auto*
have *5*: $\vdash \Box(f \longrightarrow ((\bigcirc\ f) \wedge g) \vee h) \longrightarrow \Box(\ ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg\ f)))\longrightarrow \neg f)$
using *4* **by** (*rule ImpBoxRule*)
show *?thesis*
using *2 3 5* **by** *fastforce*
qed

lemma *UntilInduction-b*:
$\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow (f \longrightarrow \Box\ f \vee f\ \mathcal{U}\ g)$
proof −
have *1*: $\vdash (\Box\ f \vee f\ \mathcal{U}\ g) = f\ \mathcal{W}\ g$
by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash (f \longrightarrow \Box\ f \vee f\ \mathcal{U}\ g) = (\ (\neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg\ g) \longrightarrow \neg\ f)$
using *1 WaitNotDistUntil* **by** *fastforce*
have *3*: $\vdash \Box\ (\ ((\neg\ f \wedge \neg\ g) \vee (\neg g \wedge \bigcirc\ (\neg\ f))) \longrightarrow \neg\ f) \longrightarrow (\ (\neg\ g)\ \mathcal{U}\ (\neg\ f \wedge \neg\ g) \longrightarrow \neg\ f)$
using *LFPUntil* **by** *blast*
have *4*: $\vdash(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow\ \ (\ ((\neg\ f \wedge \neg\ g) \vee (\neg g \wedge \bigcirc\ (\neg\ f))) \longrightarrow \neg\ f)$
using *NextImpNotNextNot*[*of f*] **by** *auto*
have *5*: $\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow\ \ \Box(\ ((\neg\ f \wedge \neg\ g) \vee (\neg g \wedge \bigcirc\ (\neg\ f))) \longrightarrow \neg\ f)$
using *4 BoxImpBoxRule* **by** *blast*
show *?thesis*
using *2 3 5* **by** *fastforce*
qed

12.3 Theorems

lemma *NextFalseSUntil*:
$\vdash \bigcirc\ g = \#False\ \mathcal{U}^s\ g$

proof −
 have *1*: $\vdash$ #*False* $\mathcal{U}$ $g = g$
  using *UntilNextUntil*[*of LIFT*(#*False*) *g*] **by** *auto*
 show *?thesis* **unfolding** *suntil-d-def* **using** *1 inteq-reflection* **by** *force*
qed

lemma *WNextUntil*:
$\vdash$ *wnext*(f $\mathcal{U}$ g) = (*empty* $\vee$ ($\bigcirc$ f) $\mathcal{U}$ ($\bigcirc$ g))
by (*meson NextUntil Prop06 WnextEqvEmptyOrNext*)

lemma *UntilRelease*:
$\vdash f\ \mathcal{R}\ g = (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g)))$
by (*simp add*: *release-d-def*)

lemma *SReleaseWait*:
$\vdash f\ \mathcal{M}\ g = (\neg\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))$
by (*simp add*: *srelease-d-def*)

lemma *ReleaseUntil*:
$\vdash f\ \mathcal{U}\ g = (\neg\ (\ (\neg\ f)\ \mathcal{R}\ (\neg\ g)))$
by (*simp add*: *release-d-def*)

lemma *WaitSRelease*:
$\vdash f\ \mathcal{W}\ g = (\neg\ (\neg\ f)\ \mathcal{M}\ (\neg\ g))$
by (*simp add*: *srelease-d-def*)

lemma *NotUntilRelease*:
$\vdash \neg(f\ \mathcal{U}\ g) = (\neg\ f)\ \mathcal{R}\ (\neg\ g)$
by (*simp add*: *ReleaseUntil*)

lemma *NotWaitSRelease*:
$\vdash \neg(f\ \mathcal{W}\ g) = (\neg\ f)\ \mathcal{M}\ (\neg\ g)$
by (*simp add*: *WaitSRelease*)

lemma *NotReleaseUntil*:
$\vdash \neg(f\ \mathcal{R}\ g) = (\neg\ f)\ \mathcal{U}\ (\neg\ g)$
by (*simp add*: *UntilRelease*)

lemma *NotSReleaseWait*:
$\vdash \neg(f\ \mathcal{M}\ g) = (\neg\ f)\ \mathcal{W}\ (\neg\ g)$
by (*simp add*: *SReleaseWait*)

lemma *BoxEqvFalseRelease*:
$\vdash \Box\ f$ = #*False* $\mathcal{R}$ f
unfolding *release-d-def*
by (*metis DiamondEqvTrueUntil Prop11 always-d-def int-simps*(*3*) *inteq-reflection lift-imp-neg*)

lemma *UntilTrue*:
 $\vdash f\ \mathcal{U}$ #*True*
using *UntilNextUntil* **by** *fastforce*

lemma *UntilIdempotent*:
$\vdash f\ \mathcal{U}\ f = f$
using *UntilNextUntil*[*of f f*] **by** *auto*

lemma *UntilImpUntil*:
assumes $\vdash f0 \longrightarrow f1$
$\vdash g0 \longrightarrow g1$
shows $\vdash f0\ \mathcal{U}\ g0 \longrightarrow f1\ \mathcal{U}\ g1$
using *assms*
by (*metis Prop10 Prop12 UntilAndDist UntilLeftDistAnd int-eq*)

lemma *UntilEqvUntil*:
assumes $\vdash f0 = f1$
$\vdash g0 = g1$
shows $\vdash f0\ \mathcal{U}\ g0 = f1\ \mathcal{U}\ g1$
proof −
have *1*: $\vdash f0 \longrightarrow f1$
using *assms* **by** *auto*
have *2*: $\vdash g0 \longrightarrow g1$
using *assms* **by** *auto*
have *3*: $\vdash f0\ \mathcal{U}\ g0 \longrightarrow f1\ \mathcal{U}\ g1$
using *1 2 UntilImpUntil*[*of f0 f1 g0 g1*] **by** *auto*
have *4*: $\vdash f1 \longrightarrow f0$
using *assms* **by** *auto*
have *5*: $\vdash g1 \longrightarrow g0$
using *assms* **by** *auto*
have *6*: $\vdash f1\ \mathcal{U}\ g1 \longrightarrow f0\ \mathcal{U}\ g0$
using *4 5 UntilImpUntil*[*of f1 f0 g1 g0*] **by** *auto*
from *3 6* **show** *?thesis* **by** *fastforce*
qed

lemma *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 UntilNextUntil 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*: $\vdash f\ \mathcal{U}\ g \wedge (\neg\ g)\ \mathcal{U}\ f \longrightarrow f\ \mathcal{U}\ f$
  using *UntilNotImp* **by** *auto*
have *2*: $\vdash\ \ f\ \mathcal{U}\ f = f$
  using *UntilIdempotent* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *AndNotUntilImp*:
$\vdash f \wedge (\neg\ f)\ \mathcal{U}\ g \longrightarrow g$
proof –
have *1*: $\vdash f = f\ \mathcal{U}\ f$
  by (*simp add*: *UntilIdempotent int-iffD1 int-iffD2 int-iffI*)
have *2*: $\vdash g =\ \#False\ \mathcal{U}\ g$
  by (*meson FalseUntil Prop11*)
have *3*: $\vdash f\ \mathcal{U}\ f \wedge (\neg\ f)\ \mathcal{U}\ g \longrightarrow \#False\ \mathcal{U}\ g$
  by (*metis 1 FalseUntil UntilNotImp inteq-reflection*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilImpOr*:
$\vdash f\ \mathcal{U}\ g \longrightarrow f \vee g$
proof –
have $\vdash f \wedge \bigcirc(f\ \mathcal{U}\ g) \longrightarrow f \vee g$
  by *force*
then show *?thesis*
  using *UntilNextUntil*[*of f g*] **by** *auto*
qed

lemma *UntilIntro*:
$\vdash\ g \longrightarrow f\ \mathcal{U}\ g$
proof –
have *1*: $\vdash\ g = \#False\ \mathcal{U}\ g$
  by (*meson FalseUntil Prop11*)
have *2*: $\vdash \#False \longrightarrow f$
  by *auto*
have *3*: $\vdash\ g \longrightarrow g$
  by *auto*
have *4*: $\vdash \#False\ \mathcal{U}\ g \longrightarrow f\ \mathcal{U}\ g$
  by (*simp add*: *UntilImpUntil*)
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *OrImpUntil*:
$\vdash f \wedge g \longrightarrow f\ \mathcal{U}\ g$
by (*simp add*: *Prop01 Prop05 UntilIntro*)

lemma *UntilAbsorp-a*:
$\vdash (f \vee f\ \mathcal{U}\ g) = (f \vee g)$
proof –
have *1*: $\vdash (f \vee f\ \mathcal{U}\ g) \longrightarrow f \vee g$

  using *UntilImpOr* **by** *fastforce*
 have *2*: $\vdash f \vee g \longrightarrow (f \vee f\ \mathcal{U}\ g)$
  using *UntilIntro* **by** *fastforce*
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilAbsorp-b*:
$\vdash (f\ \mathcal{U}\ g \vee g) = f\ \mathcal{U}\ g$
using *UntilNextUntil* **by** *fastforce*

lemma *UntilAbsorp-c*:
$\vdash (f\ \mathcal{U}\ g \wedge g) = g$
using *UntilIntro* **by** *fastforce*

lemma *UntilAbsorp-d*:
$\vdash (f\ \mathcal{U}\ g \vee (f \wedge g)) = f\ \mathcal{U}\ g$
using *UntilNextUntil* **by** *fastforce*

lemma *UntilAbsorp-e*:
$\vdash (f\ \mathcal{U}\ g \wedge (f \vee g)) = f\ \mathcal{U}\ g$
by (*meson Prop10 Prop11 UntilImpOr*)

lemma *LeftUntilAbsorp*:
$\vdash f\ \mathcal{U}\ (f\ \mathcal{U}\ g) = f\ \mathcal{U}\ g$
by (*meson Prop11 UntilUntil*)

lemma *RightUntilAbsorp*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*metis Prop11 UntilAbsorp-b UntilAbsorp-c UntilImpOr UntilRightAnd UntilUntil inteq-reflection*)

lemma *UntilAbsorpAndDiamond*:
$\vdash (f\ \mathcal{U}\ g \wedge \Diamond\ g) = f\ \mathcal{U}\ g$
by (*metis DiamondEqvTrueUntil Prop11 Prop12 UntilIdempotent UntilImpUntil int-simps*(*12*) *inteq-reflection*)

lemma *UntilAbsorpOrDiamond*:
$\vdash (f\ \mathcal{U}\ g \vee \Diamond\ g) = \Diamond\ g$
using *UntilAbsorpAndDiamond* **by** *fastforce*

lemma *UntilAbsorpDiamond*:
$\vdash f\ \mathcal{U}\ (\Diamond\ g) = \Diamond\ g$
using *DiamondDiamondEqvDiamond UntilAbsorpOrDiamond UntilAbsorp-b* **by** *fastforce*

lemma *UntilImpDiamond*:
$\vdash f\ \mathcal{U}\ g \longrightarrow \Diamond\ g$
using *UntilAbsorpAndDiamond* **by** *fastforce*

lemma *AlwaysImpNotUntilNot*:
$\vdash \Box\ f \longrightarrow \neg(g\ \mathcal{U}\ (\neg\ f))$
by (*simp add*: *UntilImpDiamond always-d-def*)

lemma *UntilAlwaysAndDist*:
$\vdash \Box\, f \wedge g\, \mathcal{U}\ h \longrightarrow (f \wedge g)\, \mathcal{U}\ (f \wedge h)$
proof –
have *1*: $\vdash \Box(h \vee g \wedge \bigcirc\ ((f \wedge g)\, \mathcal{U}\ (f \wedge h)) \longrightarrow (f \wedge g)\, \mathcal{U}\ (f \wedge h)) \longrightarrow$
$g\, \mathcal{U}\ h \longrightarrow (f \wedge g)\, \mathcal{U}\ (f \wedge h)$
using *LFPUntil* **by** *blast*
have *2*: $\vdash\ f \longrightarrow (h \vee g \wedge \bigcirc\ ((f \wedge g)\, \mathcal{U}\ (f \wedge h)) \longrightarrow (f \wedge g)\, \mathcal{U}\ (f \wedge h))$
using *UntilNextUntil*[*of LIFT*$(f \wedge g)$ *LIFT*$(f \wedge h)$] **by** *auto*
have *3*: $\vdash\ \Box\ f \longrightarrow \Box(h \vee g \wedge \bigcirc\ ((f \wedge g)\, \mathcal{U}\ (f \wedge h)) \longrightarrow (f \wedge g)\, \mathcal{U}\ (f \wedge h))$
using *2 BoxImpBoxRule* **by** *blast*
have *4*: $\vdash \Box\ f\ \longrightarrow (g\, \mathcal{U}\ h \longrightarrow (f \wedge g)\, \mathcal{U}\ (f \wedge h))$
using *1 3 lift-imp-trans* **by** *blast*
show *?thesis* **using** *4* **by** *fastforce*
qed

lemma *UntilAndImp*:
$\vdash \Box\ f \wedge \Diamond\ g \longrightarrow f\, \mathcal{U}\ g$
proof –
have *1*: $\vdash \Diamond\ g = \#True\, \mathcal{U}\ g$
by (*simp add*: *DiamondEqvTrueUntil*)
have *2*: $\vdash \Box\ f \wedge \#True\, \mathcal{U}\ g \longrightarrow (f \wedge \#True)\, \mathcal{U}\ (f \wedge g)$
using *UntilAlwaysAndDist* **by** *blast*
have *3*: $\vdash (f \wedge \#True)\, \mathcal{U}\ (f \wedge g) = f\ \mathcal{U}\ (f \wedge g)$
by *simp*
have *4*: $\vdash f\ \mathcal{U}\ (f \wedge g) \longrightarrow (f\, \mathcal{U}\ f)\, \mathcal{U}\ g$
by (*simp add*: *UntilRightAnd*)
have *5*: $\vdash (f\, \mathcal{U}\ f) = f$
by (*simp add*: *UntilIdempotent*)
have *6*: $\vdash (f\, \mathcal{U}\ f)\, \mathcal{U}\ g\ = f\, \mathcal{U}\ g$
by (*simp add*: *5 UntilEqvUntil*)
show *?thesis*
by (*metis 1 2 3 4 5 inteq-reflection lift-imp-trans*)
qed

lemma *UntilRightMono*:
$\vdash \Box(f \longrightarrow g) \longrightarrow (h\, \mathcal{U}\ f \longrightarrow h\, \mathcal{U}\ g)$
proof –
have *1*: $\vdash \Box(f \longrightarrow g) \wedge h\, \mathcal{U}\ f \longrightarrow (\ (f \longrightarrow g) \wedge h)\, \mathcal{U}\ ((f {\longrightarrow} g) \wedge f)$
using *UntilAlwaysAndDist* **by** *blast*
have *2*: $\vdash ((f \longrightarrow g) \wedge h)\, \mathcal{U}\ ((f {\longrightarrow} g) \wedge f)\ \ \longrightarrow h\, \mathcal{U}\ ((f {\longrightarrow} g) \wedge f)$
by (*meson Prop12 UntilImpUntil int-iffD2 lift-and-com*)
have *3*: $\vdash ((f {\longrightarrow} g) \wedge f) \longrightarrow g$
by *auto*
have *4*: $\vdash\ h\, \mathcal{U}\ ((f {\longrightarrow} g) \wedge f) \longrightarrow h\, \mathcal{U}\ g$
by (*simp add*: *3 UntilImpUntil*)
show *?thesis*
by (*meson 1 2 4 Prop09 lift-imp-trans*)
qed

lemma *UntilLeftMono*:

$\vdash \square\ (f \longrightarrow g) \longrightarrow (f\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h)$
proof –
have *1*: $\vdash \square\ (f \longrightarrow g) \wedge f\ \mathcal{U}\ h \longrightarrow ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ ((f \longrightarrow g) \wedge h)$
by (*simp add*: *UntilAlwaysAndDist*)
have *2*: $\vdash ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ ((f \longrightarrow g) \wedge h) \longrightarrow ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ h$
by (*meson Prop12 UntilLeftDistAnd*)
have *3*:$\vdash ((f \longrightarrow g) \wedge f) \longrightarrow g$
by *auto*
have *4*: $\vdash ((f \longrightarrow g) \wedge f)\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ h$
by (*simp add*: *3 UntilImpUntil*)
show *?thesis*
by (*meson 1 2 4 Prop09 lift-imp-trans*)
qed

lemma *UntilCatRule*:
$\vdash \square\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow (f \longrightarrow (g\ \mathcal{U}\ i))$
proof –
have *1*: $\vdash \square\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow \square\ (f \longrightarrow g\ \mathcal{U}\ h)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *2*: $\vdash \square\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow \square\ (h \longrightarrow g\ \mathcal{U}\ i)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *3*: $\vdash \square\ (h \longrightarrow g\ \mathcal{U}\ i) \longrightarrow \square\ (\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ (g\ \mathcal{U}\ i))$
by (*metis BoxEqvBoxBox BoxImpBoxRule UntilRightMono inteq-reflection*)
have *4*: $\vdash \square\ (\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ (g\ \mathcal{U}\ i)) \longrightarrow \square(\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
by (*metis BoxEqvBoxBox UntilUntil int-iffD1 inteq-reflection*)
have *5*: $\vdash\ \ \square\ (f \longrightarrow g\ \mathcal{U}\ h) \longrightarrow (f \longrightarrow g\ \mathcal{U}\ h)$
by (*simp add*: *BoxElim*)
have *6*: $\vdash \square(\ g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i) \longrightarrow (g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
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*: $\vdash \square\ (\ (f \longrightarrow g\ \mathcal{U}\ h) \wedge (h \longrightarrow g\ \mathcal{U}\ i)) \longrightarrow$
$(f \longrightarrow g\ \mathcal{U}\ h) \wedge (g\ \mathcal{U}\ h \longrightarrow g\ \mathcal{U}\ i)$
using *1 2 3 4 5 6* **by** *fastforce*
from *7 8* **show** *?thesis* **by** *auto*
qed

lemma *UntilStrengthen*:
$\vdash \square\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
proof –
have *11*: $\vdash \square\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow \square(f \longrightarrow h)$
by (*meson BoxImpBoxRule Prop12 int-iffD2 lift-and-com*)
have *1*: $\vdash \square\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g)$
using *11 UntilLeftMono*[*of f h g*] **by** *fastforce*
have *21*: $\vdash \square\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow \square(g \longrightarrow i)$
by (*simp add*: *BoxImpBoxRule Prop01 Prop05*)
have *2*: $\vdash \square\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
using *21 UntilRightMono*[*of g i h*] **by** *fastforce*
have *3*: $\vdash \square\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g) \wedge (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
using *1 2* **by** *fastforce*

have *4*: $\vdash (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ g) \wedge (\ h\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i) \longrightarrow (f\ \mathcal{U}\ g \longrightarrow h\ \mathcal{U}\ i)$
  by *auto*
 from *3 4* **show** *?thesis* **by** *auto*
qed

lemma *UntilInduction*:
$\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc\ f) \longrightarrow (f \longrightarrow \neg(h\ \mathcal{U}\ g))$
proof –
 have *1*: $\vdash \Box\ (\neg\ g) \longrightarrow \neg(h\ \mathcal{U}\ g)$
  by (*simp add*: *UntilImpDiamond always-d-def*)
 have *15*: $\vdash (f \longrightarrow \neg\ g \wedge \bigcirc\ f) \longrightarrow (g \vee\ \bigcirc\ (\neg\ f) \longrightarrow \neg\ f)$
  using *NextImpNotNextNot*[*of f*] **by** *fastforce*
 have *16*: $\vdash (f \longrightarrow \neg\ g \wedge \bigcirc\ f) \longrightarrow (g \vee\ \#True \wedge \bigcirc\ (\neg\ f) \longrightarrow \neg\ f)$
  using *15* **by** *auto*
 have *2*: $\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc\ f) \longrightarrow \Box(g \vee \#True \wedge \bigcirc\ (\neg\ f) \longrightarrow \neg\ f)$
  using *16 BoxImpBoxRule* **by** *blast*
 have *3*: $\vdash \Box\ (f \longrightarrow \neg\ g \wedge \bigcirc\ f) \longrightarrow (\#True\ \mathcal{U}\ g \longrightarrow \neg\ f)$
  using *2 LFPUntil*[*of g LIFT*(*#True*) *LIFT*(¬ *f*)]
  by *fastforce*
 have *4*: $\vdash (\#True\ \mathcal{U}\ g \longrightarrow \neg\ f) \longrightarrow (f \longrightarrow \neg(\#True\ \mathcal{U}\ g))$
  by *auto*
 have *5*: $\vdash\ \neg(\#True\ \mathcal{U}\ g) = \Box\ (\neg\ g)$
  using *BoxEqvFalseRelease NotUntilRelease inteq-reflection* **by** *fastforce*
 from *5 4 3 1* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilBoxImp*:
$\vdash f\ \mathcal{U}\ (\Box\ g) \longrightarrow \Box(f\ \mathcal{U}\ g)$
proof –
 have *1*: $\vdash f\ \mathcal{U}\ \Box g \longrightarrow f\ \mathcal{U}\ g$
  by (*meson BoxElim BoxGen MP UntilRightMono*)
 have *2*: $\vdash wnext\ (f\ \mathcal{U}\ \Box g) = (empty \vee \bigcirc\ (f\ \mathcal{U}\ \Box g))$
  by (*meson WnextEqvEmptyOrNext*)
 have *3*: $\vdash \Box\ g = (g \wedge wnext\ (\Box g))$
  by (*metis* (*no-types*) *BoxEqvAndWnextBox*)
 have *4*: $\vdash f\ \mathcal{U}\ \Box g = (\Box g \vee f \wedge \bigcirc\ (f\ \mathcal{U}\ \Box g))$
  by (*meson UntilNextUntil*)
 have *5*: $\vdash g \wedge wnext\ (\Box g) \longrightarrow empty \vee \bigcirc\ (f\ \mathcal{U}\ \Box g)$
  by (*metis NextUntil Prop01 Prop05 Prop08 UntilIntro WnextEqvEmptyOrNext int-iffD1 inteq-reflection*)
 have *6*: $\vdash more \wedge f\ \mathcal{U}\ \Box g \longrightarrow empty \vee \bigcirc\ (f\ \mathcal{U}\ \Box g)$
  using *5 3 4* **by** *fastforce*
 have *7*: $\vdash more \wedge f\ \mathcal{U}\ \Box g \longrightarrow \bigcirc\ (f\ \mathcal{U}\ \Box g)$
  using *2 6 WnextAndMoreEqvNext* **by** *fastforce*
 from *1 7* **show** *?thesis* **using** *BoxIntro*[*of LIFT*(*f* 𝒰 (□ *g*)) *LIFT*(*f* 𝒰 *g*)]
  by *auto*
qed

lemma *UntilBoxEqvBox*:
$\vdash f\ \mathcal{U}\ (\Box\ f) = \Box\ f$

proof −
 have *1*: $\vdash\ f\ \mathcal{U}\ (\Box\ f) \longrightarrow \Box(f\ \mathcal{U}\ f)$
  using *UntilBoxImp*[*of f f*] **by** *auto*
 have *2*: $\vdash \Box(f\ \mathcal{U}\ f) = \Box\ f$
  by (*simp add*: *BoxEqvBox UntilIdempotent*)
 have *3*: $\vdash \Box\ f \longrightarrow f\ \mathcal{U}\ (\Box\ f)$
  by (*simp add*: *UntilIntro*)
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilRightStrengthen*:
$\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow f\ \mathcal{U}\ (g\ \mathcal{U}\ h)$
by (*meson BoxGen MP OrImpUntil UntilRightMono*)

lemma *UntilLeftStrengthen*:
$\vdash (f \wedge g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{U}\ g)\ \mathcal{U}\ h$
by (*simp add*: *OrImpUntil UntilImpUntil*)

lemma *UntilLeftAndOrder*:
$\vdash (f \wedge g)\ \mathcal{U}\ h \longrightarrow f\ \mathcal{U}\ (g\ \mathcal{U}\ h)$
by (*metis Prop12 UntilIdempotent UntilImpUntil UntilIntro inteq-reflection*)

lemma *UntilFrameNext*:
$\vdash \Box\ f \longrightarrow (\bigcirc\ g \longrightarrow \bigcirc\ (f\ \mathcal{U}\ g))$
by (*simp add*: *NextImpNext Prop01 Prop05 Prop09 UntilIntro*)

lemma *UntilFrameDiamond*:
$\vdash \Box\ f \longrightarrow (\Diamond\ g \longrightarrow \Diamond\ (f\ \mathcal{U}\ g))$
by (*meson NowImpDiamond Prop09 UntilAndImp lift-imp-trans*)

lemma *UntilFrameBox*:
$\vdash \Box\ f \longrightarrow (\Box\ g \longrightarrow \Box\ (f\ \mathcal{U}\ g))$
by (*simp add*: *BoxAndBoxImpBoxRule OrImpUntil Prop09*)

lemma *UntilImpNot*:
$\vdash f\ \mathcal{U}\ g \longrightarrow (f \wedge \neg g)\ \mathcal{U}\ g$
proof −
 have *1*: $\vdash f\ \mathcal{U}\ g \longrightarrow \Diamond\ g$
  by (*simp add*: *UntilImpDiamond*)
 have *2*: $\vdash \Diamond\ g = \#True\ \mathcal{U}\ g$
  by (*simp add*: *DiamondEqvTrueUntil*)
 have *3*: $\vdash \#True\ \mathcal{U}\ g \longrightarrow (\neg g)\ \mathcal{U}\ g$
  by (*simp add*: *TrueUntilImpNotUntil*)
 have *4*: $\vdash (f\ \mathcal{U}\ g \wedge (\neg g)\ \mathcal{U}\ g) = (f \wedge \neg g)\ \mathcal{U}\ g$
  using *UntilAndDist* **by** *fastforce*
 show *?thesis*
  by (*meson 1 2 3 4 Prop10 int-iffD1 lift-imp-trans*)
qed

lemma *UntilAndRule*:

$\vdash f\ \mathcal{U}\ g = (f \wedge \neg g)\ \mathcal{U}\ g$
proof –
have *1*: $\vdash (f \wedge \neg g)\ \mathcal{U}\ g \longrightarrow f\ \mathcal{U}\ g$
using *UntilAndDist* **by** *fastforce*
show *?thesis* **by** (*simp add*: *1 UntilImpNot int-iffI*)
qed

lemma *UntilWait*:
$\vdash f\ \mathcal{U}\ g = (f\ \mathcal{W}\ g \wedge \diamond\ g)$
proof –
have *1*: $\vdash f\ \mathcal{U}\ g \longrightarrow f\ \mathcal{W}\ g \wedge \diamond\ g$
by (*simp add*: *Prop05 Prop12 UntilImpDiamond wait-d-def*)
have *2*: $\vdash (f\ \mathcal{W}\ g \wedge \diamond\ g) = ((\square\ f \vee f\ \mathcal{U}\ g) \wedge \diamond\ g)$
by (*auto simp add*: *wait-d-def*)
have *3*: $\vdash ((\square\ f \vee f\ \mathcal{U}\ g) \wedge \diamond\ g) = (\ (\square\ f \wedge \diamond g) \vee (f\ \mathcal{U}\ g \wedge \diamond\ g))$
by *auto*
have *4*: $\vdash (\square\ f \wedge \diamond g) \longrightarrow f\ \mathcal{U}\ g$
by (*simp add*: *UntilAndImp*)
have *5*: $\vdash (f\ \mathcal{U}\ g \wedge \diamond\ g) \longrightarrow f\ \mathcal{U}\ g$
by *auto*
show *?thesis*
using *1 2 4* **by** *fastforce*
qed

lemma *WaitLeftDistAnd*:
$\vdash f\ \mathcal{W}\ (g \wedge h) \longrightarrow f\ \mathcal{W}\ g \wedge f\ \mathcal{W}\ h$
proof –
have *1*: $\vdash f\ \mathcal{W}\ (g \wedge h) \longrightarrow f\ \mathcal{W}\ g$
unfolding *wait-d-def*
by (*metis Prop08 Prop12 UntilAbsorp-a UntilLeftDistAnd int-iffD1 inteq-reflection*)
have *2*: $\vdash f\ \mathcal{W}\ (g \wedge h) \longrightarrow\ f\ \mathcal{W}\ h$
unfolding *wait-d-def*
by (*metis Prop08 Prop12 UntilAbsorp-a UntilLeftDistAnd int-iffD1 inteq-reflection*)
show *?thesis* **by** (*simp add*: *1 2 Prop12*)
qed

lemma *WaitRightDistAnd*:
$\vdash (f \wedge g)\ \mathcal{W}\ h = (f\ \mathcal{W}\ h \wedge g\ \mathcal{W}\ h)$
proof –
have *1*: $\vdash \square(f \wedge g) = (\square\ f \wedge \square\ g)$
by (*metis BoxAndBoxEqvBoxRule inteq-reflection lift-and-com*)
have *2*: $\vdash (f \wedge g)\ \mathcal{U}\ h = (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h)$
by (*simp add*: *UntilAndDist*)
have *3*: $\vdash ((\square\ f \wedge \square\ g)\ ) \longrightarrow ((\square f \vee f\ \mathcal{U}\ h) \wedge (\square g \vee g\ \mathcal{U}\ h))$
by (*simp add*: *intI*)
have *4*: $\vdash (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h) \longrightarrow ((\square f \vee f\ \mathcal{U}\ h) \wedge (\square g \vee g\ \mathcal{U}\ h))$
by *auto*
have *5*: $\vdash ((\square\ f \wedge \square\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h)) \longrightarrow ((\square f \vee f\ \mathcal{U}\ h) \wedge (\square g \vee g\ \mathcal{U}\ h))$
using *3 4* **by** *fastforce*
have *6*: $\vdash \square\ f \wedge \square\ g \longrightarrow ((\square\ f \wedge \square\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$

by *auto*
have *7*: $\vdash \Box f \wedge g\ \mathcal{U}\ h \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
by (*metis 2 Prop05 Prop12 UntilAlwaysAndDist UntilLeftDistAnd inteq-reflection lift-imp-trans*)
have *8*: $\vdash f\ \mathcal{U}\ h \wedge \Box\ g \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
by (*metis Prop05 Prop08 Prop12 UntilAlwaysAndDist UntilAndDist UntilLeftDistAnd inteq-reflection lift-and-com*)
have *9*: $\vdash f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
by *auto*
have *10*: $\vdash ((\Box f \vee f\ \mathcal{U}\ h) \wedge (\Box g \vee g\ \mathcal{U}\ h)) \longrightarrow ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
using *6 7 8 9* **by** *fastforce*
have *11*: $\vdash ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h)) = ((\Box f \vee f\ \mathcal{U}\ h) \wedge (\Box g \vee g\ \mathcal{U}\ h))$
using *5 10* **by** *auto*
have *12*: $\vdash\ (\Box(f \wedge g) \vee (f \wedge g)\ \mathcal{U}\ h) = ((\Box\ f \wedge \Box\ g) \vee (f\ \mathcal{U}\ h \wedge g\ \mathcal{U}\ h))$
using *1 2* **by** *fastforce*
show *?thesis* **unfolding** *wait-d-def* **using** *11 12*
by (*meson Prop04 UntilIdempotent*)
qed

lemma *WaitAndRule*:
$\vdash f\ \mathcal{W}\ g = (f \wedge \neg\ g)\ \mathcal{W}\ g$
proof –
have *1*: $\vdash (f\ \mathcal{W}\ g \wedge (\neg g\ )\ \mathcal{W}\ g) = (f \wedge \neg\ g)\ \mathcal{W}\ g$
by (*meson Prop11 WaitRightDistAnd*)
have *2*: $\vdash (\neg g\ )\ \mathcal{W}\ g$
by (*metis* (*no-types*, *opaque-lifting*) *DiamondEqvTrueUntil FalseUntil UntilAndRule UntilExclMid always-d-def int-simps*(*17*) *int-simps*(*4*) *inteq-reflection wait-d-def*)
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *WaitUntilb*:
$\vdash f\ \mathcal{W}\ \ g = (\Box\ (\ f \wedge \neg\ g) \vee f\ \mathcal{U}\ g)$
proof –
have *1*: $\vdash f\ \mathcal{W}\ g = (f \wedge \neg\ g)\ \mathcal{W}\ g$
by (*simp add*: *WaitAndRule*)
have *2*: $\vdash (f \wedge \neg\ g)\ \mathcal{W}\ g = (\Box\ (f \wedge \neg\ g) \vee (f \wedge \neg\ g)\ \mathcal{U}\ g)$
by (*auto simp add*: *wait-d-def*)
have *3*: $\vdash (f \wedge \neg\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*meson Prop11 UntilAndRule*)
show *?thesis*
using *1 2 3* **by** *fastforce*
qed

lemma *UntilNotDistWait*:
$\vdash (\neg\ (f\ \mathcal{U}\ g)) = (\neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
proof –
have *1*: $\vdash (\neg\ ((\neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g))) = (\neg(\neg\ f \wedge \neg\ g))\ \mathcal{U}\ (\ \neg\ (\neg\ g) \wedge \neg\ (\neg\ f \wedge \neg\ g))$
using *WaitNotDistUntil* **by** *blast*
have *2*: $\vdash (\neg(\neg\ f \wedge \neg\ g)) = (f \vee g)$
by *auto*

have *3*: $\vdash$ ($\neg$ ($\neg$ g) $\wedge$ $\neg$ ($\neg$ f $\wedge$ $\neg$ g)) = g
  by *auto*
have *4*: $\vdash$ ($\neg$($\neg$ f $\wedge$ $\neg$ g)) $\mathcal{U}$ ($\neg$ ($\neg$ g) $\wedge$ $\neg$ ($\neg$ f $\wedge$ $\neg$ g)) =
  (f $\vee$ g) $\mathcal{U}$ g
  using *2 3 UntilEqvUntil* **by** *blast*
have *5*: $\vdash$ (f $\vee$ g) $\mathcal{U}$ g = ((f $\vee$ g) $\wedge$ $\neg g$) $\mathcal{U}$ g
  by (*simp add*: *UntilAndRule*)
have *6*: $\vdash$ ((f $\vee$ g) $\wedge$ $\neg g$) = (f $\wedge$ $\neg$ g)
  by *auto*
have *7*: $\vdash$ ((f $\vee$ g) $\wedge$ $\neg g$) $\mathcal{U}$ g = (f $\wedge$ $\neg g$) $\mathcal{U}$ g
  using *6 inteq-reflection* **by** *fastforce*
have *8*: $\vdash$ (f $\wedge$ $\neg g$) $\mathcal{U}$ g = f $\mathcal{U}$ g
  by (*meson Prop11 UntilAndRule*)
have *9*: $\vdash$ f $\mathcal{U}$ g = ($\neg$ (($\neg$ g) $\mathcal{W}$ ($\neg$ f $\wedge$ $\neg$ g)))
  using *1 4 5 7 8* **by** *fastforce*
show *?thesis* **using** *9* **by** *auto*
qed

lemma *UntilImpWait*:
$\vdash$ f $\mathcal{U}$ g $\longrightarrow$ f $\mathcal{W}$ g
by (*meson Prop03 WaitUntilb*)

lemma *WaitAndDist*:
$\vdash$ ($\Box$ f $\wedge$ g $\mathcal{W}$ h) $\longrightarrow$ (f $\wedge$ g) $\mathcal{W}$ (f $\wedge$ h)
proof –
have *1*: $\vdash$ ($\Box$ f $\wedge$ g $\mathcal{W}$ h) = ($\Box$ f $\wedge$ ($\Box$ g $\vee$ g $\mathcal{U}$ h))
  by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash$ ($\Box$ f $\wedge$ ($\Box$ g $\vee$ g $\mathcal{U}$ h)) = (($\Box f$ $\wedge$ $\Box g$) $\vee$ ($\Box$ f $\wedge$ g $\mathcal{U}$ h))
  by *auto*
have *3*: $\vdash$ ($\Box f$ $\wedge$ $\Box g$) = $\Box$(f $\wedge$ g)
  by (*simp add*: *BoxAndBoxEqvBoxRule*)
have *4*: $\vdash$ ($\Box$ f $\wedge$ g $\mathcal{U}$ h) $\longrightarrow$ (f $\wedge$ g) $\mathcal{U}$ (f $\wedge$ h)
  by (*simp add*: *UntilAlwaysAndDist*)
have *5*: $\vdash$ (($\Box f$ $\wedge$ $\Box g$) $\vee$ ($\Box$ f $\wedge$ g $\mathcal{U}$ h)) $\longrightarrow$ $\Box$(f $\wedge$ g) $\vee$ (f $\wedge$ g) $\mathcal{U}$ (f $\wedge$ h)
  using *3 4* **by** *fastforce*
have *6*: $\vdash$ ($\Box$(f $\wedge$ g) $\vee$ (f $\wedge$ g) $\mathcal{U}$ (f $\wedge$ h)) = (f $\wedge$ g) $\mathcal{W}$ (f $\wedge$ h)
  by (*auto simp add*: *wait-d-def*)
show *?thesis*
  using *1 5 6* **by** *fastforce*
qed

lemma *WaitDiamondOr*:
$\vdash$ f $\mathcal{W}$ ($\Diamond$ g) = ($\Box$ f $\vee$ $\Diamond$ g)
proof –
have *1*: $\vdash$ f $\mathcal{W}$ ($\Diamond$ g) = ($\Box$ f $\vee$ f $\mathcal{U}$ ($\Diamond$ g))
  by (*auto simp add*: *wait-d-def*)
have *2*: $\vdash$ f $\mathcal{U}$ ($\Diamond$ g) = $\Diamond$ g
  by (*simp add*: *UntilAbsorpDiamond*)
show *?thesis* **using** *1 2 Prop06* **by** *blast*
qed

lemma *WaitBoxImp*:
$\vdash f\ \mathcal{W}\ (\Box\ g) \longrightarrow \Box\ (\ f\ \mathcal{W}\ g)$
proof –
 have *1*: $\vdash f\ \mathcal{W}\ (\Box\ g) = (\Box\ f \lor f\ \mathcal{U}\ (\Box\ g))$
  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*: $\vdash (\Box\ f \lor f\ \mathcal{U}\ (\Box\ g)) \longrightarrow (\Box\ (\Box\ f) \lor \Box(f\ \mathcal{U}\ g))$
  using *2 3* **by** *fastforce*
 have *5*: $\vdash \Box\ (\Box\ f) \longrightarrow \Box(\ \Box\ f \lor f\ \mathcal{U}\ g)$
  by (*metis BoxImpBoxRule Prop08 UntilIdempotent UntilIntro int-simps*(*11*) *int-simps*(*25*) *inteq-reflection*)
 have *6*: $\vdash \Box(f\ \mathcal{U}\ g) \longrightarrow \Box(\ \Box\ f \lor f\ \mathcal{U}\ g)$
  by (*metis BoxImpBoxRule UntilImpWait wait-d-def*)
 have *7*: $\vdash (\Box\ (\Box\ f) \lor \Box(f\ \mathcal{U}\ g)) \longrightarrow \Box(\ \Box\ f \lor f\ \mathcal{U}\ g)$
  using *5 6* **by** *fastforce*
 have *6*: $\vdash \Box(\ \Box\ f \lor f\ \mathcal{U}\ g) = \Box\ (\ f\ \mathcal{W}\ g)$
  by (*simp add*: *wait-d-def*)
 show *?thesis*
  by (*metis 4 7 lift-imp-trans wait-d-def*)
qed

lemma *WaitAbsorbBox*:
$\vdash f\ \mathcal{W}\ (\Box\ f) = \Box\ f$
by (*metis Prop02 Prop11 UntilBoxEqvBox UntilImpWait inteq-reflection wait-d-def*)

lemma *BoxImpWait*:
$\vdash \Box\ f \longrightarrow f\ \mathcal{W}\ g$
by (*auto simp add*: *wait-d-def*)

lemma *WaitDistNext*:
$\vdash \bigcirc\ (\ f\ \mathcal{W}\ g) = (\bigcirc\ f)\ \mathcal{W}\ (\bigcirc\ g)$
— nitpick finds counterexample, does not hold because of finite intervals
oops

lemma *WaitDistNextInfinite*:
$\vdash inf \longrightarrow \bigcirc\ (\ f\ \mathcal{W}\ g) = (\bigcirc\ f)\ \mathcal{W}\ (\bigcirc\ g)$
proof –
 have *1*: $\vdash \bigcirc\ (\Box\ f \lor f\ \mathcal{U}\ g) \longrightarrow (\bigcirc\ (\Box\ f) \lor \bigcirc(f\ \mathcal{U}\ g))$
  by (*simp add*: *ChopOrImpRule next-d-def*)
 have *2*: $\vdash (\bigcirc\ (\Box\ f) \lor \bigcirc(f\ \mathcal{U}\ g)) \longrightarrow \bigcirc\ (\Box\ f \lor f\ \mathcal{U}\ g)$
  by (*metis BoxImpWait NextImpNext Prop02 UntilImpWait wait-d-def*)
 have *21*: $\vdash \bigcirc\ (\Diamond(\neg\ f)) = \Diamond(\bigcirc(\neg f))$
  by (*metis* (*no-types*, *lifting*) *ChopAssoc FiniteChopSkipEqvSkipChopFinite inteq-reflection next-d-def sometimes-d-def*)
 have *22*: $\vdash (\neg(\ \bigcirc\ f)) = (empty \lor \bigcirc(\ \neg f))$

using *WnextEqvEmptyOrNext*[*of LIFT*($\neg f$)] **unfolding** *wnext-d-def* **by** *simp*
have *23*: $\vdash$ *inf* $\longrightarrow \bigcirc(\neg f) = (\neg(\bigcirc f))$
using *22 MoreAndInfEqvInf WnextAndMoreEqvNext WnextEqvEmptyOrNext* **by** *fastforce*
have *24*: $\vdash$ *inf* $\longrightarrow \Diamond(\bigcirc(\neg f)) = \Diamond(\neg(\bigcirc f))$
unfolding *sometimes-d-def* **using** *23*
InfRightChopEqvChop[*of LIFT*($\bigcirc(\neg f)$) *LIFT*($(\neg (\bigcirc f))$) *LIFT*(*finite*)]
by *simp*
have *25*: $\vdash$ *inf* $\longrightarrow \bigcirc (\Diamond(\neg f)) = \Diamond(\neg(\bigcirc f))$
using *21 24* **by** *fastforce*
have *26*: $\vdash$ *inf* $\longrightarrow$ (*empty* $\vee \bigcirc(\Diamond(\neg f))) = \bigcirc (\Diamond(\neg f))$
using *MoreAndInfEqvInf WnextAndMoreEqvNext WnextEqvEmptyOrNext* **by** *fastforce*
have *27*: $\vdash$ *inf* $\longrightarrow$ *wnext* $(\Diamond(\neg f)) = \bigcirc (\Diamond(\neg f))$
by (*metis 26 WnextEqvEmptyOrNext inteq-reflection*)
have *28*: $\vdash$ *inf* $\longrightarrow (\neg (\bigcirc (\neg \Diamond(\neg f)))) = \Diamond(\neg \bigcirc f)$
using *21 24 27* **unfolding** *wnext-d-def* **by** *fastforce*
have *3*: $\vdash$ *inf* $\longrightarrow \bigcirc (\Box f) = \Box (\bigcirc f)$
unfolding *always-d-def* **using** *28* **by** *fastforce*
have *4*: $\vdash \bigcirc(f\ \mathcal{U}\ g) = \bigcirc f\ \mathcal{U} \bigcirc g$
by (*simp add*: *NextUntil*)
have *5*: $\vdash \bigcirc (\Box f \vee f\ \mathcal{U}\ g) = (\bigcirc (\Box f) \vee \bigcirc(f\ \mathcal{U}\ g))$
using *1 2 int-iffI* **by** *blast*
have *6*: $\vdash$ *inf* $\longrightarrow \bigcirc (f\ \mathcal{W}\ g) = \bigcirc (\Box f \vee f\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *7*: $\vdash$ *inf* $\longrightarrow \bigcirc (\Box f \vee f\ \mathcal{U}\ g) = (\bigcirc (\Box f) \vee \bigcirc(f\ \mathcal{U}\ g))$
using *5* **by** *auto*
have *8*: $\vdash$ *inf* $\longrightarrow (\bigcirc (\Box f) \vee \bigcirc(f\ \mathcal{U}\ g)) = (\Box (\bigcirc f) \vee \bigcirc f\ \mathcal{U} \bigcirc g)$
using *4 3* **by** *fastforce*
show *?thesis*
by (*metis 5 8 inteq-reflection wait-d-def*)
qed

lemma *WnextAlwaysEqvAlwaysWnext*:
$\vdash$ *finite* $\longrightarrow$ *wnext* $(\Box f) = \Box($*wnext* $f)$
by (*metis* (*mono-tags*, *lifting*) *BoxEqvFiniteYields FiniteChopSkipEqvSkipChopFinite SkipYieldsEqvWeakNext YieldsYieldsEqvChopYields intI inteq-reflection unl-lift2*)

lemma *WaitExpand*:
$\vdash f\ \mathcal{W}\ g = (g \vee (f \wedge \bigcirc(f\ \mathcal{W}\ g)))$
— nitpick finds counterexample, does not hold because of finite intervals
oops

lemma *WaitExpand*:
$\vdash f\ \mathcal{W}\ g = (g \vee (f \wedge$ *wnext*$(f\ \mathcal{W}\ g)))$
proof –
have *1*: $\vdash f\ \mathcal{W}\ g = (\Box f \vee f\ \mathcal{U}\ g)$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash \Box f = (f \wedge$ *wnext*$(\Box f))$
by (*simp add*: *BoxEqvAndWnextBox*)
have *3*: $\vdash f\ \mathcal{U}\ g = (g \vee (f \wedge \bigcirc (f\ \mathcal{U}\ g)))$
using *UntilNextUntil* **by** *blast*

have *4*: $\vdash (f \wedge \textit{wnext}(\Box\ f)) = (f \wedge (\textit{empty} \vee \bigcirc\ (\Box\ f)))$
  using *2 BoxEqvAndEmptyOrNextBox*[*of f*] **by** *fastforce*
 have *5*: $\vdash\ \ \textit{wnext}(f\ \mathcal{W}\ g) = (\textit{empty} \vee \bigcirc\ (f\ \mathcal{W}\ g))$
  using *WnextEqvEmptyOrNext* **by** *blast*
 have *6*: $\vdash (f \wedge (\textit{empty} \vee \bigcirc\ (\Box\ f))) = (\ (f \wedge \textit{empty}) \vee (f \wedge \bigcirc\ (\Box\ f)))$
  by *auto*
 have *7*: $\vdash ((\ (f \wedge \textit{empty}) \vee (f \wedge \bigcirc\ (\Box\ f))) \vee$
      $(g \vee (f \wedge \bigcirc\ (f\ \mathcal{U}\ g)))\ ) =$
      $(g \vee (f \wedge (\textit{empty} \vee \bigcirc\ (\Box f)\ \ \vee \bigcirc\ (f\ \mathcal{U}\ g)\ )\ ))$
  by *auto*
 have *8*: $\vdash (\bigcirc\ (\Box f)\ \ \vee \bigcirc\ (f\ \mathcal{U}\ g)) = \bigcirc(\Box f \vee f\ \mathcal{U}\ g)$
  by (*metis ChopOrEqv Prop11 next-d-def*)
 show *?thesis*
  by (*metis 1 2 3 4 5 6 7 8 inteq-reflection*)
qed

lemma *InfImpWnextEqnNext*:
$\vdash \textit{inf} \longrightarrow \textit{wnext}\ f = \bigcirc\ f$
proof –
 have *1*: $\vdash \textit{wnext}\ f = (\textit{empty} \vee \bigcirc\ f)$
  by (*simp add*: *WnextEqvEmptyOrNext*)
 have *2*: $\vdash \textit{inf} \longrightarrow \textit{more}$
  using *MoreAndInfEqvInf* **by** *auto*
 have *3*: $\vdash \textit{inf} \longrightarrow (\textit{empty} \vee \bigcirc\ f) =\ \bigcirc\ f$
   unfolding *empty-d-def* **using** *2* **by** *fastforce*
 show *?thesis*
  by (*metis 1 3 inteq-reflection*)
qed

lemma *WaitExpandInfinite*:
$\vdash \textit{inf} \longrightarrow f\ \mathcal{W}\ g = (g \vee (f \wedge \bigcirc(f\ \mathcal{W}\ g)))$
proof –
 have *1*: $\vdash f\ \mathcal{W}\ g = (g \vee (f \wedge \textit{wnext}(f\ \mathcal{W}\ g)))$
  using *WaitExpand* **by** *blast*
 have *2*: $\vdash \textit{inf} \longrightarrow\ \ \textit{wnext}(f\ \mathcal{W}\ g) = \bigcirc\ (f\ \mathcal{W}\ g)$
  using *InfImpWnextEqnNext* **by** *blast*
 have *3*: $\vdash \textit{inf} \longrightarrow (g \vee (f \wedge \textit{wnext}(f\ \mathcal{W}\ g))) = (g \vee (f \wedge \bigcirc(f\ \mathcal{W}\ g)))$
  using *2* **by** *auto*
 show *?thesis* **using** *1 3* **by** *fastforce*
qed

lemma *WaitExclMid*:
$\vdash f\ \mathcal{W}\ g \vee f\ \mathcal{W}\ (\neg\ g)$
using *WaitExpand*
proof –
 have *1*: $\vdash f\ \mathcal{W}\ g = (g \vee f \wedge \textit{wnext}\ (f\ \mathcal{W}\ g))$
  by (*simp add*: *WaitExpand*)
 have *2*: $\vdash f\ \mathcal{W}\ (\neg\ g) = ((\neg\ g) \vee f \wedge \textit{wnext}\ (\ f\ \mathcal{W}\ (\neg\ g)))$
  by (*simp add*: *WaitExpand*)
 have *3*: $\vdash (f\ \mathcal{W}\ g \vee f\ \mathcal{W}\ (\neg\ g)) =$

($(g \vee f \wedge wnext\ (f\ \mathcal{W}\ g)) \vee ((\neg\ g) \vee f \wedge wnext\ (\ f\ \mathcal{W}\ (\neg\ g))))$
  using *1 2* **by** *fastforce*
 from *3* **show** *?thesis* **by** *fastforce*
qed

lemma *WaitleftZero*:
$\vdash \#True\ \mathcal{W}\ g$
by (*meson BoxGen BoxImpWait MP TrueW*)

lemma *WaitLeftDistOr*:
$\vdash f\ \mathcal{W}\ (g \vee h) = (f\ \mathcal{W}\ g \vee f\ \mathcal{W}\ h)$
proof –
 have *1*: $\vdash f\ \mathcal{W}\ (g \vee h) = (\ \Box\ f \vee f\ \mathcal{U}\ (g \vee h))$
  by (*simp add*: *wait-d-def*)
 have *2*: $\vdash (f\ \mathcal{W}\ g \vee f\ \mathcal{W}\ h) = ((\Box\ f \vee f\ \mathcal{U}\ g) \vee (\Box\ f \vee f\ \mathcal{U}\ h))$
  by (*simp add*: *wait-d-def*)
 have *3*: $\vdash f\ \mathcal{U}\ (g \vee h) = (f\ \mathcal{U}\ g \vee f\ \mathcal{U}\ h)$
  by (*simp add*: *UntilOrDist*)
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *WaitRightDistOr*:
$\vdash f\ \mathcal{W}\ h \vee g\ \mathcal{W}\ h \longrightarrow (f \vee g)\ \mathcal{W}\ h$
proof –
 have *0*:$\vdash \Box\ g \longrightarrow \Box\ (f \vee g)$
  by (*simp add*: *BoxImpBoxRule intI*)
 have *1*: $\vdash \Box f \longrightarrow \Box\ (f \vee g)$
  by (*simp add*: *BoxImpBoxRule intI*)
 have *11*: $\vdash \Box\ f \vee \Box\ g \longrightarrow \Box\ (f \vee g)$
  using *0 1 Prop02* **by** *blast*
 have *2*: $\vdash\ \ f\ \mathcal{W}\ h = (\Box\ f \vee f\ \mathcal{U}\ h)$
  by (*simp add*: *wait-d-def*)
 have *3*:$\vdash (f \vee g)\ \mathcal{W}\ h = (\Box\ (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
  by (*simp add*: *wait-d-def*)
 have *4*: $\vdash g\ \mathcal{W}\ h = (\Box g\ \ \vee g\ \mathcal{U}\ h)$
  by (*simp add*: *wait-d-def*)
 have *5*: $\vdash f\ \mathcal{U}\ h \vee g\ \mathcal{U}\ h \longrightarrow (f \vee g)\ \mathcal{U}\ h$
  using *UntilRightDistOr* **by** *simp*
 have *6*:$\vdash (f\ \mathcal{W}\ h \vee g\ \mathcal{W}\ h) = ((\Box\ f \vee \Box\ g) \vee (f\ \mathcal{U}\ h \vee g\ \mathcal{U}\ h))$
  using *2 4* **by** *fastforce*
 from *11 5 6 3* **show** *?thesis*
  by (*meson BoxImpWait Prop02 Prop11 UntilImpWait lift-imp-trans*)
qed

lemma *WaitOrRule*:
$\vdash f\ \mathcal{W}\ g = (f \vee g)\ \mathcal{W}\ g$
proof –
 have *1*: $\vdash f\ \mathcal{W}\ g \longrightarrow (f \vee g)\ \mathcal{W}\ g$
  by (*metis* (*no-types*, *lifting*) *Prop03 Prop10 UntilAbsorp-a WaitNotDistUntil int-iffD1 int-simps*(*14*) *int-simps*(*32*) *int-simps*(*33*) *inteq-reflection*)

have *2*: $\vdash (f \vee g)\ \mathcal{W}\ g \longrightarrow f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *Prop03 Prop10 WaitNotDistUntil int-iffD2 int-simps*(*14*) *int-simps*(*32*) *int-simps*(*33*) *inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *UntilOrRule*:
$\vdash f\ \mathcal{U}\ g = (f \vee g)\ \mathcal{U}\ g$
by (*metis UntilWait WaitOrRule inteq-reflection*)

lemma *WaitRule*:
$\vdash (\neg f)\ \mathcal{W}\ f$
by (*metis BoxGen BoxImpWait MP WaitOrRule int-eq-true int-simps*(*29*) *inteq-reflection*)

lemma *UntilRule*:
$\vdash (\neg f)\ \mathcal{U}\ f = \Diamond\ f$
using *DiamondEqvTrueUntil UntilOrRule inteq-reflection* **by** *fastforce*

lemma *WaitImpRule*:
$\vdash (f \longrightarrow g)\ \mathcal{W}\ f$
proof –
have *1*: $\vdash (f \longrightarrow g)\ \mathcal{W}\ f = ((f \longrightarrow g) \vee f)\ \mathcal{W}\ f$
by (*simp add*: *WaitOrRule*)
have *2*: $\vdash (f \longrightarrow g) \vee f$
by *auto*
have *3*: $\vdash ((f \longrightarrow g) \vee f)\ \mathcal{W}\ f = \#True\ \mathcal{W}\ f$
by (*metis 1 2 int-eq-true inteq-reflection*)
show *?thesis*
using *1 3 WaitleftZero* **by** *fastforce*
qed

lemma *DiamondUntilImpRule*:
$\vdash \Diamond\ f \longrightarrow (f \longrightarrow g)\ \mathcal{U}\ f$
using *UntilWait WaitImpRule* **by** *fastforce*

lemma *WaitNotDist*:
$\vdash (\neg (f\ \mathcal{W}\ g)) = (f \wedge \neg g)\ \mathcal{U}\ (\neg f \wedge \neg g)$
proof –
have *1*: $\vdash (\neg (f\ \mathcal{W}\ g)) = (\neg g)\ \mathcal{U}\ (\neg f \wedge \neg g)$
using *WaitNotDistUntil* **by** *blast*
have *2*: $\vdash (\neg g)\ \mathcal{U}\ (\neg f \wedge \neg g) = (\neg g \wedge \neg(\neg f \wedge \neg g))\ \mathcal{U}\ (\neg f \wedge \neg g)$
using *UntilAndRule* **by** *blast*
have *3*: $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) = (f \wedge \neg g)$
by *auto*
have *4*: $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g))\ \mathcal{U}\ (\neg f \wedge \neg g) = (f \wedge \neg g)\ \mathcal{U}\ (\neg f \wedge \neg g)$
using *3 inteq-reflection* **by** *force*
show *?thesis* **using** *1 2 4* **by** *fastforce*
qed

lemma *UntilNotDist*:

$\vdash (\neg (f\ \mathcal{U}\ g)) = (f \wedge \neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
proof –
have *1*: $\vdash (\neg (f\ \mathcal{U}\ g)) = (\neg\ g\ )\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
using *UntilNotDistWait* **by** *blast*
have *2*: $\vdash\ (\neg\ g\ )\ \mathcal{W}\ (\neg\ f \wedge \neg\ g) = (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g))\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
by (*simp add*: *WaitAndRule*)
have *3*: $\vdash (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g)) = (f \wedge \neg\ g)$
by *auto*
have *4*: $\vdash (\neg\ g \wedge \neg(\neg\ f \wedge \neg\ g))\ \mathcal{W}\ (\neg\ f \wedge \neg\ g) = (f \wedge \neg\ g)\ \mathcal{W}\ (\neg\ f \wedge \neg\ g)$
using *3 inteq-reflection* **by** *force*
show *?thesis* **using** *1 2 4* **by** *fastforce*
qed

lemma *UntilDuala*:
$\vdash (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g))) = g\ \mathcal{W}\ (f \wedge g)$
proof –
have *1*: $\vdash (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g))) = (\neg\ f \wedge \neg\ (\neg\ g))\ \mathcal{W}\ (\ \neg(\neg\ f) \wedge \neg\ (\neg\ g))$
using *UntilNotDist* **by** *blast*
have *2*: $\vdash (\neg\ f \wedge g)\ \mathcal{W}\ (\ f \wedge g) = g\ \mathcal{W}\ (\ f \wedge g)$
using *1 UntilNotDistWait int-eq* **by** *fastforce*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *UntilDualb*:
$\vdash (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g))) = (\neg\ f \wedge g)\ \mathcal{W}\ (f \wedge g)$
proof –
have *1*: $\vdash (\neg\ (\ (\neg\ f)\ \mathcal{U}\ (\neg\ g))) = (\neg\ f \wedge \neg\ (\neg\ g))\ \mathcal{W}\ (\ \neg(\neg\ f) \wedge \neg\ (\neg\ g))$
using *UntilNotDist* **by** *blast*
show *?thesis*
using *1* **by** *auto*
qed

lemma *WaitDuala*:
$\vdash (\neg\ (\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))) = g\ \mathcal{U}\ (f \wedge g)$
proof –
have *1*: $\vdash (\neg\ (\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))) = (\neg\ f \wedge \neg\ (\neg\ g))\ \mathcal{U}\ (\ \neg(\neg\ f) \wedge \neg\ (\neg\ g))$
using *WaitNotDist* **by** *blast*
have *2*: $\vdash (\neg\ f \wedge g)\ \mathcal{U}\ (\ f \wedge g) = g\ \mathcal{U}\ (\ f \wedge g)$
using *1 WaitNotDistUntil int-eq* **by** *fastforce*
show *?thesis*
using *1 2* **by** *fastforce*
qed

lemma *WaitDualb*:
$\vdash (\neg\ (\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))) = (\neg\ f \wedge g)\ \mathcal{U}\ (f \wedge g)$
proof –
have *1*: $\vdash (\neg\ (\ (\neg\ f)\ \mathcal{W}\ (\neg\ g))) = (\neg\ f \wedge \neg\ (\neg\ g))\ \mathcal{U}\ (\ \neg(\neg\ f) \wedge \neg\ (\neg\ g))$
using *WaitNotDist* **by** *blast*
show *?thesis* **using** *1* **by** *auto*

qed

lemma *WaitIdempotent*:
$\vdash f\ \mathcal{W}\ f = f$
by (*meson BoxElim Prop02 Prop12 UntilIdempotent UntilImpWait UntilIntro WaitUntilb int-iffD1 int-iffI lift-imp-trans*)

lemma *WaitRightZero*:
$\vdash f\ \mathcal{W}\ \#True$
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*:
$\vdash f\ \mathcal{W}\ g \longrightarrow f \vee g$
by (*metis Prop03 WaitIdempotent WaitLeftDistOr WaitOrRule inteq-reflection*)

lemma *BoxOrImpWait*:
$\vdash \Box(f \vee g) \longrightarrow f\ \mathcal{W}\ g$
using *BoxImpWait WaitOrRule* **by** *fastforce*

lemma *BoxImpImpWait*:
$\vdash \Box(\neg\ g \longrightarrow f) \longrightarrow f\ \mathcal{W}\ g$
proof –
have *1*: $\vdash (\neg\ g \longrightarrow f) = (f \vee g)$
by *auto*
have *2*: $\vdash \Box\ (\neg\ g \longrightarrow f) = \Box(f \vee g)$
using *1 BoxEqvBox* **by** *blast*
show *?thesis* **using** *2 BoxOrImpWait* **by** *fastforce*
qed

lemma *WaitInsertion*:
$\vdash g \longrightarrow f\ \mathcal{W}\ g$
by (*simp add*: *Prop05 UntilIntro wait-d-def*)

lemma *WaitFrameNext*:
$\vdash \Box\ f \longrightarrow (\bigcirc\ g \longrightarrow \bigcirc\ (f\ \mathcal{W}\ g))$
by (*simp add*: *NextImpNext Prop01 Prop05 Prop09 WaitInsertion*)

lemma *WaitFrameDiamond*:
$\vdash \Box\ f \longrightarrow (\Diamond\ g \longrightarrow \Diamond\ (f\ \mathcal{W}\ g))$
by (*simp add*: *DiamondImpDiamond Prop01 Prop05 Prop09 WaitInsertion*)

lemma *WaitFrameBox*:
$\vdash \Box\ f \longrightarrow (\Box\ g \longrightarrow \Box\ (f\ \mathcal{W}\ g))$
by (*meson BoxAndBoxImpBoxRule OrImpUntil Prop09 UntilImpWait lift-imp-trans*)

lemma *WaitInductiona*:
$\vdash \Box\ (f \longrightarrow (\bigcirc\ f \wedge g) \vee h) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ h)$
by (*simp add*: *UntilInduction-a wait-d-def*)

lemma *WaitInductionb*:
$\vdash \Box\ (f \longrightarrow \bigcirc\ f \vee g) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
by (*simp add*: *UntilInduction-b wait-d-def*)

lemma *WaitInductionc*:
$\vdash \Box(f \longrightarrow \bigcirc\ f) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
proof –
 have *1*: $\vdash (f \longrightarrow \bigcirc\ f) \longrightarrow\ (f \longrightarrow wnext\ f)$
  unfolding *wnext-d-def* **using** *NextImpNotNextNot*[*of f*] **by** *auto*
 have *2*: $\vdash \Box(f \longrightarrow \bigcirc\ f) \longrightarrow \Box\ (f \longrightarrow wnext\ f)$
  using *1 BoxImpBoxRule* **by** *blast*
 show *?thesis* **by** (*meson 2 BoxImpWait BoxInduct Prop09 lift-imp-trans*)
qed

lemma *WaitInductiond*:
$\vdash \Box\ (f \longrightarrow g \wedge \bigcirc\ f) \longrightarrow (f \longrightarrow f\ \mathcal{W}\ g)$
proof –
 have *1*: $\vdash (f \longrightarrow g \wedge \bigcirc\ f) \longrightarrow\ (f \longrightarrow\ wnext\ f)$
  unfolding *wnext-d-def* **using** *NextImpNotNextNot*[*of f*] **by** *auto*
 have *2*: $\vdash \Box(f \longrightarrow g \wedge \bigcirc\ f) \longrightarrow \Box\ (f \longrightarrow\ wnext\ f)$
  using *1 BoxImpBoxRule* **by** *blast*
 show *?thesis* **by** (*meson 2 BoxImpWait BoxInduct Prop09 lift-imp-trans*)
qed

lemma *WaitAbsorba*:
$\vdash (f \vee f\ \mathcal{W}\ g) = (f \vee g)$
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 *WaitAbsorbb*:
$\vdash (f\ \mathcal{W}\ g \vee g) = f\ \mathcal{W}\ g$
using *WaitInsertion*[*of g f*] **by** *auto*

lemma *WaitAbsorbc*:
$\vdash (f\ \mathcal{W}\ g \wedge g) = g$
using *WaitInsertion* **by** *fastforce*

lemma *WaitAbsorbd*:
$\vdash (f\ \mathcal{W}\ g \wedge (f \vee g)) = f\ \mathcal{W}\ g$
by (*meson Prop10 Prop11 WaitImpOr*)

lemma *WaitAbsorbe*:
$\vdash (f\ \mathcal{W}\ g \lor (f \land g)) = f\ \mathcal{W}\ g$
unfolding *wait-d-def*
using *UntilAbsorp-d* **by** *fastforce*

lemma *WaitLeftAbsorb*:
$\vdash f\ \mathcal{W}\ (f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *BoxEqvBoxBox UntilUntil WaitAbsorbBox WaitAbsorba WaitLeftDistOr inteq-reflection wait-d-def*)

lemma *WaitRightAbsorb*:
$\vdash (f\ \mathcal{W}\ g)\ \mathcal{W}\ g = f\ \mathcal{W}\ g$
by (*metis* (*no-types*, *lifting*) *LeftUntilAbsorp Prop10 WaitInsertion WaitNotDistUntil int-iffD1 int-iffI int-simps*(*32*) *inteq-reflection*)

lemma *WaitBox*:
$\vdash \Box\ f = f\ \mathcal{W}\ \#False$
by (*metis* (*no-types*, *lifting*) *BoxGen DiamondNotEqvNotBox UntilAbsorpAndDiamond UntilAbsorp-c int-eq-true int-simps*(*2*) *int-simps*(*25*) *inteq-reflection wait-d-def*)

lemma *WaitDiamond*:
$\vdash \Diamond\ f = (\neg(\ (\neg\ f)\ \mathcal{W}\ \#False))$
using *DiamondNotEqvNotBox WaitBox* **by** *fastforce*

lemma *WaitImp*:
$\vdash f\ \mathcal{W}\ g \longrightarrow \Box\ f \lor \Diamond\ g$
by (*metis Prop08 UntilImpDiamond WaitAbsorbb WaitImpOr WaitRightAbsorb int-eq wait-d-def*)

lemma *WaitRightUntilAbsorb*:
$\vdash f\ \mathcal{W}\ (f\ \mathcal{U}\ g) = f\ \mathcal{W}\ g$
by (*metis UntilUntil WaitOrRule inteq-reflection wait-d-def*)

lemma *WaitLeftUntilAbsorb*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{W}\ g = f\ \mathcal{U}\ g$
by (*metis Prop11 RightUntilAbsorp UntilAbsorp-b UntilImpWait WaitImpOr inteq-reflection*)

lemma *UntilRightWaitAbsorb*:
$\vdash f\ \mathcal{U}\ (f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
using *UntilImpWait UntilIntro WaitLeftAbsorb* **by** *fastforce*

lemma *UntilLeftWaitAbsorb*:
$\vdash (f\ \mathcal{W}\ g)\ \mathcal{U}\ g = f\ \mathcal{U}\ g$
by (*metis UntilWait WaitRightAbsorb inteq-reflection*)

lemma *WaitDiamondAbsorb*:
$\vdash (\Diamond\ g)\ \mathcal{W}\ g = \Diamond\ g$
by (*metis DiamondEqvTrueUntil WaitLeftUntilAbsorb inteq-reflection*)

lemma *WaitAndBoxAbsorb*:
$\vdash (\Box\ f \land f\ \mathcal{W}\ g) = \Box\ f$

by (*meson BoxImpWait NotDiamondNotEqvBox Prop04 Prop10*)

lemma *WaitOrBoxAbsorb*:
$\vdash (\square\ f \lor f\ \mathcal{W}\ g) = f\ \mathcal{W}\ g$
by (*metis UntilRightWaitAbsorb WaitLeftAbsorb inteq-reflection wait-d-def*)

lemma *WaitAndBoxImpBox*:
$\vdash f\ \mathcal{W}\ g \land \square\ (\neg\ g) \longrightarrow \square\ f$
by (*metis* (*no-types*, *opaque-lifting*) *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*: $\vdash (\square\ f \land \diamond\ g) \longrightarrow f\ \mathcal{U}\ g$
 using *UntilAndImp* **by** *blast*
 show *?thesis*
 using *1 2* **by** *fastforce*
qed

lemma *NotBoxAndWaitImpDiamond*:
$\vdash \neg(\ \square\ f) \land f\ \mathcal{W}\ g \longrightarrow \diamond\ g$
using *WaitImp* **by** *fastforce*

lemma *DiamondImpNotBoxOrUntil*:
$\vdash \diamond\ g \longrightarrow \neg\ (\square\ f) \lor f\ \mathcal{U}\ g$
proof –
 have *1*: $\vdash \diamond\ g \land \square\ f \longrightarrow f\ \mathcal{U}\ g$
 using *UntilAndImp* **by** *fastforce*
 show *?thesis* **using** *1* **by** *auto*
qed

lemma *WaitRightDistImp*:
$\vdash (f \longrightarrow g)\ \mathcal{W}\ h \longrightarrow (f\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ h)$
proof –
 have *0*: $\vdash \square(f \longrightarrow g) \land \square f \longrightarrow \square g$
 by (*simp add*: *BoxAndBoxImpBoxRule intI*)
 have *1*: $\vdash \square(f \longrightarrow g) \land \square f \longrightarrow \square g \lor g\ \mathcal{U}\ h$
 using *0* **by** *auto*
 have *2*: $\vdash \square(f \longrightarrow g) \land f\ \mathcal{U}\ h \longrightarrow (\ (f \longrightarrow g) \land f)\ \mathcal{U}\ (\ (f \longrightarrow g) \land h)$
 using *UntilAlwaysAndDist*[*of LIFT*(*f* $\longrightarrow$*g*) *f h*] **by** *auto*
 have *3*: $\vdash (f \longrightarrow g) \land f \longrightarrow g$
 by *auto*
 have *4*: $\vdash (f \longrightarrow g) \land h \longrightarrow h$
 by *auto*
 have *5*: $\vdash (\ (f \longrightarrow g) \land f)\ \mathcal{U}\ (\ (f \longrightarrow g) \land h) \longrightarrow \square g \lor g\ \mathcal{U}\ h$
 by (*simp add*: *3 4 Prop05 UntilImpUntil*)

have *6*: $\vdash \Box(f \longrightarrow g) \wedge f\ \mathcal{U}\ h \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
  using *2 5 lift-imp-trans* **by** *blast*
have *7*: $\vdash \Box(f \longrightarrow g) \longrightarrow \Box f \vee f\ \mathcal{U}\ h \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
  by (*simp add*: *1 6 Prop09 Prop12*)
have *8*: $\vdash \Box\ f \wedge (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow (f \wedge (f \longrightarrow g))\ \mathcal{U}\ (f \wedge h)$
  by (*simp add*: *UntilAlwaysAndDist*)
have *9*: $\vdash f \wedge (f \longrightarrow g) \longrightarrow g$
  by *auto*
have *10*: $\vdash f \wedge h \longrightarrow h$
  by *auto*
have *11*: $\vdash (f \wedge (f \longrightarrow g))\ \mathcal{U}\ (f \wedge h) \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
  by (*simp add*: *10 9 Prop05 UntilImpUntil*)
have *12*: $\vdash\ (f \longrightarrow g)\ \mathcal{U}\ h \wedge \Box f \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
  using *8 11* **by** *fastforce*
have *13*: $\vdash (f \longrightarrow g)\ \mathcal{U}\ h \wedge f\ \mathcal{U}\ h \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
  by (*metis 3 Prop05 UntilAndDist UntilImpUntil UntilIntro UntilUntil inteq-reflection*)
have *14*: $\vdash (f \longrightarrow g)\ \mathcal{U}\ h \longrightarrow \Box f \vee f\ \mathcal{U}\ h \longrightarrow \Box g \vee g\ \mathcal{U}\ h$
  by (*simp add*: *12 13 Prop09 Prop12*)
show *?thesis* **unfolding** *wait-d-def* **using** *7 14 Prop02* **by** *blast*
qed

lemma *WaitLeftMono*:
$\vdash \Box\ (f \longrightarrow g) \longrightarrow (f\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ h)$
by (*meson BoxImpWait WaitRightDistImp lift-imp-trans*)

lemma *WaitRightMono*:
$\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{W}\ f \longrightarrow h\ \mathcal{W}\ g)$
proof –
have *1*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{U}\ f \longrightarrow h\ \mathcal{U}\ g)$
  by (*simp add*: *UntilRightMono*)
have *2*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (\Box\ h \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
  by *auto*
have *3*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (h\ \mathcal{U}\ f \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
  using *1* **by** *auto*
have *4*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (\Box\ h \vee h\ \mathcal{U}\ f \longrightarrow \Box\ h \vee h\ \mathcal{U}\ g)$
  using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *wait-d-def*)
qed

lemma *WaitStrengthen*:
$\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
proof –
have *1*: $\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g)$
  by (*meson BoxAndBoxEqvBoxRule Prop01 Prop05 Prop11 WaitLeftMono lift-and-com lift-imp-trans*)
have *2*: $\vdash\ \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
  by (*meson BoxElim BoxImpBoxBox BoxImpBoxRule Prop12 WaitRightMono lift-imp-trans*)
have *3*: $\vdash \Box\ (\ (f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g) \wedge (\ h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
  using *1 2* **by** *fastforce*
have *4*: $\vdash (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ g) \wedge (\ h\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i) \longrightarrow (f\ \mathcal{W}\ g \longrightarrow h\ \mathcal{W}\ i)$
  by *auto*

from *3 4* **show** *?thesis* **by** *auto*
qed

lemma *WaitCatRule*:
$\vdash \Box(\ (f \longrightarrow g\ \mathcal{W}\ h) \wedge (h \longrightarrow g\ \mathcal{W}\ i)) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ i)$
proof –
have *1*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{W}\ \ h) \wedge (h \longrightarrow g\ \mathcal{W}\ \ i)) \longrightarrow \Box\ (f \longrightarrow g\ \mathcal{W}\ h)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *2*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{W}\ h) \wedge (h \longrightarrow g\ \mathcal{W}\ i)) \longrightarrow \Box\ (h \longrightarrow g\ \mathcal{W}\ i)$
by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)
have *3*: $\vdash \Box\ (h \longrightarrow g\ \mathcal{W}\ i) \longrightarrow \Box\ (\ g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ (g\ \mathcal{W}\ i))$
by (*metis BoxEqvBoxBox BoxImpBoxRule WaitRightMono inteq-reflection*)
have *4*: $\vdash \Box\ (\ g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ (g\ \mathcal{W}\ i)) \longrightarrow \Box(\ g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i)$
by (*metis BoxEqvBoxBox WaitLeftAbsorb int-iffD1 inteq-reflection*)
have *5*: $\vdash\ \ \Box\ (f \longrightarrow g\ \mathcal{W}\ h) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ h)$
by (*simp add*: *BoxElim*)
have *6*: $\vdash \Box(\ g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i) \longrightarrow (g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i)$
by (*simp add*: *BoxElim*)
have *7*: $\vdash (f \longrightarrow g\ \mathcal{W}\ h) \wedge (g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i) \longrightarrow (f \longrightarrow g\ \mathcal{W}\ i)$
by *auto*
have *8*: $\vdash \Box\ (\ (f \longrightarrow g\ \mathcal{W}\ h) \wedge (h \longrightarrow g\ \mathcal{W}\ i)) \longrightarrow$
$(f \longrightarrow g\ \mathcal{W}\ h) \wedge (g\ \mathcal{W}\ h \longrightarrow g\ \mathcal{W}\ i)$
using *1 2 3 4 5 6***by** *fastforce*
from *7 8* **show** *?thesis* **by** *auto*
qed

lemma *LeftUntilWaitImp*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{W}\ h \longrightarrow (f\ \mathcal{W}\ g)\ \mathcal{W}\ h$
by (*meson BoxGen MP UntilImpWait WaitLeftMono*)

lemma *RightWaitUntilImp*:
$\vdash f\ \mathcal{W}\ (g\ \mathcal{U}\ h) \longrightarrow f\ \mathcal{W}\ (g\ \mathcal{W}\ h)$
by (*meson BoxGen MP UntilImpWait WaitRightMono*)

lemma *RightUntilUntilImp*:
$\vdash f\ \mathcal{U}\ (g\ \mathcal{U}\ h) \longrightarrow f\ \mathcal{U}\ (g\ \mathcal{W}\ h)$
by (*meson BoxGen MP UntilImpWait UntilRightMono*)

lemma *LeftUntilUntilImp*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{U}\ h \longrightarrow (f\ \mathcal{W}\ g)\ \mathcal{U}\ h$
by (*simp add*: *UntilImpUntil UntilImpWait*)

lemma *LeftUntilOrStrengthen*:
$\vdash (f\ \mathcal{U}\ g)\ \mathcal{U}\ h \longrightarrow (f \vee g)\ \mathcal{U}\ h$
by (*simp add*: *UntilImpOr UntilImpUntil*)

lemma *LeftWaitOrStrengthen*:
$\vdash (f\ \mathcal{W}\ g)\ \mathcal{W}\ h \longrightarrow (f \vee g)\ \mathcal{W}\ h$
by (*meson BoxGen MP WaitImpOr WaitLeftMono*)

lemma *RightWaitOrStrengthen*:
$\vdash f\ \mathcal{W}\ (g\ \mathcal{W}\ h) \longrightarrow f\ \mathcal{W}\ (g \vee h)$
by (*meson BoxGen MP WaitImpOr WaitRightMono*)

lemma *BoxImpBoxOr*:
$\vdash \Box\ f \longrightarrow \Box(f \vee g)$
by (*metis BoxEqvBoxBox BoxImpBoxRule BoxImpWait Prop12 WaitAbsorbd inteq-reflection*)

lemma *RightWaitOrOrder*:
$\vdash f\ \mathcal{W}\ (g\ \mathcal{W}\ h) \longrightarrow (f \vee g)\ \mathcal{W}\ h$
proof –
have *1*: $\vdash f\ \mathcal{W}\ (g\ \mathcal{W}\ h) = (\Box\ f \vee f\ \mathcal{U}\ (\Box\ g \vee g\ \mathcal{U}\ h))$
by (*simp add*: *wait-d-def*)
have *2*: $\vdash (f \vee g)\ \mathcal{W}\ h = (\Box\ (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
by (*simp add*: *wait-d-def*)
have *3*: $\vdash \Box\ f \longrightarrow (\Box\ (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
using *BoxImpBoxOr* **by** *fastforce*
have *4*: $\vdash f\ \mathcal{U}\ (\Box\ g \vee g\ \mathcal{U}\ h) = (f\ \mathcal{U}\ (\Box\ g) \vee f\ \mathcal{U}\ (g\ \mathcal{U}\ h))$
using *UntilOrDist* **by** *blast*
have *5*: $\vdash f\ \mathcal{U}\ (g\ \mathcal{U}\ h) \longrightarrow (\Box\ (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
by (*simp add*: *Prop05 UntilRightOr*)
have *6*: $\vdash f\ \mathcal{U}\ (\Box\ g) \longrightarrow (\Box\ (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
by (*metis BoxImpBoxRule BoxImpWait UntilBoxImp UntilImpOr lift-imp-trans wait-d-def*)
have *7*: $\vdash (f\ \mathcal{U}\ (\Box\ g) \vee f\ \mathcal{U}\ (g\ \mathcal{U}\ h)) \longrightarrow (\Box\ (f \vee g) \vee (f \vee g)\ \mathcal{U}\ h)$
using *5 6* **by** *fastforce*
show *?thesis*
using *1 2 3 4 7* **by** *fastforce*
qed

lemma *RightWaitAndOrder*:
$\vdash f\ \mathcal{W}\ (g \wedge h) \longrightarrow f\ \mathcal{W}\ (g\ \mathcal{W}\ h)$
by (*metis Prop03 WaitAbsorbe 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*: $\vdash (\neg f \wedge \neg g)\ \mathcal{U}\ (f \vee g) \longrightarrow (\neg\ f \wedge \neg g)\ \mathcal{U}\ f\ \vee (\neg\ f \wedge \neg g)\ \mathcal{U}\ g$
by (*simp add*: *UntilOrDist int-iffD1*)
have *6*: $\vdash (\neg\ f \wedge \neg g)\ \mathcal{U}\ f \longrightarrow (\neg\ g)\ \mathcal{U}\ f$
by (*metis UntilAndRule int-iffD2 inteq-reflection lift-and-com*)
have *7*: $\vdash (\neg\ f \wedge \neg g)\ \mathcal{U}\ g \longrightarrow (\neg\ f)\ \mathcal{U}\ g$
by (*metis UntilAndRule int-iffD2*)

have *8*: $\vdash (\neg\, f \wedge \neg g)\ \mathcal{U}\ f\ \vee (\neg\, f \wedge \neg g)\ \mathcal{U}\ g \longrightarrow (\neg\, f)\ \mathcal{U}\ g \vee (\neg\, g)\ \mathcal{U}\ f$
  using *6 7* **by** *fastforce*
 have *9*: $\vdash\ \Diamond(f \vee g) \longrightarrow (\neg\, f)\ \mathcal{U}\ g \vee (\neg\, g)\ \mathcal{U}\ f$
  using *2 3 4 5 8* **by** *fastforce*
 show *?thesis* **using** *1 9* **by** *fastforce*
qed

lemma *WaitOrder*:
$\vdash (\neg\, f)\ \mathcal{W}\ g \vee (\neg\, g)\ \mathcal{W}\ f$
proof –
 have *1*: $\vdash (\neg\, f)\ \mathcal{W}\ g = (\Box\ (\neg\, f) \vee (\neg\, f)\ \mathcal{U}\ g)$
  by (*simp add*: *wait-d-def*)
 have *2*:$\vdash (\neg\, g)\ \mathcal{W}\ f = (\Box\ (\neg\, g) \vee (\neg\, g)\ \mathcal{U}\ f)$
  by (*simp add*: *wait-d-def*)
 have *3*: $\vdash ((\Box\ (\neg\, f) \vee (\neg\, f)\ \mathcal{U}\ g) \vee (\Box\ (\neg\, g) \vee (\neg\, g)\ \mathcal{U}\ f)) =$
   $(\ (\Box\ (\neg\, f) \vee \Box\ (\neg\, g)) \vee ((\neg\, f)\ \mathcal{U}\ g \vee (\neg\, g)\ \mathcal{U}\ f))$
  by *auto*
 have *4*: $\vdash ((\neg\, f)\ \mathcal{U}\ g \vee (\neg\, g)\ \mathcal{U}\ f) = \Diamond(f \vee g)$
  using *UntilOrder* **by** *blast*
 have *5*: $\vdash (\Box\ (\neg\, f) \vee \Box\ (\neg\, g)) = (\neg\ (\Diamond\ f) \vee \neg\ (\Diamond\ g))$
  by (*simp add*: *always-d-def*)
 have *6*: $\vdash \Diamond(f \vee g) = (\Diamond f \vee \Diamond\ g)$
  by (*simp add*: *ChopOrEqv sometimes-d-def*)
 have *7*: $\vdash (\Box\ (\neg\, f) \vee \Box\ (\neg\, g)) \vee \Diamond(f \vee g)$
  using *5 6* **by** *fastforce*
 show *?thesis*
  using *1 2 4 7* **by** *fastforce*
qed

lemma *WaitImpOrder*:
$\vdash f\ \mathcal{W}\ g \wedge (\neg\, g)\ \mathcal{W}\ h \longrightarrow f\ \mathcal{W}\ h$
proof –
 have *1*: $\vdash f\ \mathcal{W}\ g = (\Box f \vee f\ \mathcal{U}\ g)$
  by (*simp add*: *wait-d-def*)
 have *2*: $\vdash f\ \mathcal{W}\ h = (\Box\ f \vee f\ \mathcal{U}\ h)$
  by (*simp add*: *wait-d-def*)
 have *3*: $\vdash (\neg\, g)\ \mathcal{W}\ h = (\Box\ (\neg\, g) \vee (\neg\, g)\ \mathcal{U}\ h)$
  by (*simp add*: *wait-d-def*)
 have *4*: $\vdash \Box\ f \wedge (\Box\ (\neg\, g) \vee (\neg\, g)\ \mathcal{U}\ h) \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
  by *auto*
 have *5*: $\vdash (\Box f \vee f\ \mathcal{U}\ g) \wedge \Box\ (\neg\, g) \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
  using *1 WaitAndBoxImpBox* **by** *fastforce*
 have *6*: $\vdash f\ \mathcal{U}\ g \wedge (\neg\, g)\ \mathcal{U}\ h \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
  by (*simp add*: *Prop05 UntilNotImp*)
 have *7*: $\vdash \Box\ f \wedge (\neg\, g)\ \mathcal{U}\ h \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
  by *auto*
 have *8*: $\vdash (\Box f \vee f\ \mathcal{U}\ g) \wedge\ \ (\neg\, g)\ \mathcal{U}\ h \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
  using *6 7* **by** *fastforce*
 have *9*: $\vdash (\Box f \vee f\ \mathcal{U}\ g) \wedge (\Box\ (\neg\, g) \vee (\neg\, g)\ \mathcal{U}\ h) \longrightarrow (\Box\ f \vee f\ \mathcal{U}\ h)$
  using *5 8* **by** *fastforce*

show *?thesis* **by** (*simp add*: *9 wait-d-def*)
qed

end

13 Pi operator for infinite and finite ITL

theory *Pi*
imports *Until Omega*
begin

This theory introduces the Pi operator [10, 7]. The Pi operator is defined in terms of the filter operator. We prove the soundness of the rules and axiom system. The until operator from Until.thy is used as there is a striking similarity of the expressiveness of the until and the Pi operator [7].

13.1 Definitions

definition *sfxfilt* :: *'a nellist* ⇒ (*'a*:: *world*) *formula* ⇒ *'a nellist nellist*
where *sfxfilt s f* = (*nfilter* (λ σ. σ ⊨*f*) (*ndropns s*))

definition *pifilt* :: *'a nellist* ⇒ (*'a*:: *world*) *formula* ⇒ *'a nellist*
where *pifilt s f* = (*nmap* (λ*s*. *nnth s 0*) (*sfxfilt s f*))

definition *pi-d* :: (*'a*:: *world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *pi-d F G* ≡ λ*s*. ((∃ *i* ≤ *nlength s*. (*ndropn 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 $\rightleftharpoons$ *CONST upi-d*

13.2 Semantic Lemmas

lemma *sfxfilter-help*:
$(\exists\ ys \in nset\ (ndropns\ xs)\ .\ f\ ys) = (\exists\ i \leq nlength\ xs.\ f\ (ndropn\ i\ xs))$
using *in-nset-ndropns* **by** *auto*

lemma *pifiltinit-help*:
$(\exists\ y \in nset\ (xs).\ w\ (NNil\ y)\ ) = (\exists\ i \leq nlength\ xs.\ w\ (NNil\ (nnth\ xs\ i)))$
by (*metis in-nset-conv-nnth*)

lemma *sfxfilt-nnth*:
assumes $(\exists\ i \leq nlength\ \sigma.\ (ndropn\ i\ \sigma) \models f)$
$i \leq nlength\ (sfxfilt\ \sigma\ f)$
shows $(nnth\ (sfxfilt\ \sigma\ f)\ i) \models f$
using *assms* **by** (*metis in-nset-ndropns nkfilter-nnth-aa sfxfilt-def*)

lemma *pifilt-exists*:
assumes $(\exists\, i \leq nlength\ \sigma.\ f\ (ndropn\ i\ \sigma)\ )$
shows $(\exists\, i \leq nlength(sfxfilt\ \sigma\ f).\ (nnth\ (sfxfilt\ \sigma\ f)\ i) \models f)$
using *assms* **by** (*metis sfxfilt-nnth zero-enat-def zero-le*)

lemma *sfxfilt-pifilt-nlength*:
shows $nlength\ (pifilt\ \sigma\ f) = nlength\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))$
by (*simp add*: *pifilt-def*)

lemma *sfxfilt-pifilt-nnth*:
assumes $j \leq nlength\ (pifilt\ \sigma\ f)$
shows $nnth\ (pifilt\ \sigma\ f)\ j = nnth\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ j$
using *assms* **by** (*simp add*: *pifilt-def*)

lemma *sfxfilt-pifilt*:
shows $(nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ f)) = pifilt\ \sigma\ f$
by (*simp add*: *pifilt-def*)

lemma *sfxfilt-nlength-bound*:
assumes $(\exists\, i \leq nlength\ xs.\ f\ (ndropn\ i\ xs))$
shows $nlength\ (sfxfilt\ xs\ f) \leq nlength\ xs$
using *assms*
by (*metis length-nfilter-le ndropns-nlength sfxfilt-def*)

lemma *pifilt-nlength-bound*:
assumes $(\exists\, i \leq nlength\ \sigma.\ f\ (ndropn\ i\ \sigma))$
shows $nlength\ (pifilt\ \sigma\ f) \leq\ nlength\ \sigma$
using *assms* **by** (*simp add*: *pifilt-def sfxfilt-nlength-bound*)

lemma *sfxfilt-nlength-nnth-bound*:
assumes $(\exists\, i \leq nlength\ \sigma.\ f\ (ndropn\ i\ \sigma))$
$j \leq\ nlength\ (sfxfilt\ \sigma\ f)$

shows *nlength* (*nnth* (*sfxfilt* σ *f*) *j*) ≤ *nlength* σ
using *assms* **by** (*simp add*: *nfilter-ndropns-nnth-bound sfxfilt-def sfxfilter-help*)

lemma *sfxfilt-pifilt-nnth-ndropn*:
assumes (∃ *i* ≤ *nlength* σ. (*ndropn i* σ) ⊨ *f*)
j≤ *nlength* (*pifilt* σ *f*)
shows *nnth* (*sfxfilt* σ *f*) *j* = *ndropn* (*nnth* (*nkfilter f 0* (*ndropns* σ)) *j*) σ
proof −
have *0*: ∃ *x*∈*nset* (*ndropns* σ). *f x*
by (*simp add*: *assms*(*1*) *sfxfilter-help*)
have *1*: *nnth* (*sfxfilt* σ *f*) *j* = *nnth* (*nfilter f* (*ndropns* σ)) *j*
by (*simp add*: *sfxfilt-def*)
have *2*: *nnth* (*nfilter f* (*ndropns* σ)) *j* = *nnth* (*ndropns* σ) (*nnth* (*nkfilter f 0* (*ndropns* σ)) *j*)
using *nkfilter-nfilter*[*of ndropns* σ λ *s*. *s* ⊨ *f j 0*] *assms* **unfolding** *pifilt-def sfxfilt-def*
by (*simp add*: *0 nkfilter-nlength*)
have *30*: *enat* (*nnth* (*nkfilter f 0* (*ndropns* σ)) *j*) ≤ *nlength* σ
using *0 nkfilter-upperbound*[*of* (*ndropns* σ) *f j 0*] *assms* **unfolding** *pifilt-def sfxfilt-def*
by (*metis gen-nlength-def ndropns-nlength nkfilter-nlength nlength-code nlength-nmap*)
have *3*: *nnth* (*ndropns* σ) (*nnth* (*nkfilter f 0* (*ndropns* σ)) *j*) =
ndropn (*nnth* (*nkfilter f 0* (*ndropns* σ)) *j*) σ
using *ndropns-nnth*[*of* (*nnth* (*nkfilter f 0* (*ndropns* σ)) *j*)]
using *30* **by** *blast*
show *?thesis* **by** (*simp add*: *1 2 3*)
qed

lemma *pifilt-nnth*:
assumes (∃ *i*≤*nlength* σ. *f* (*ndropn i* σ))
i ≤ *nlength* (*pifilt* σ *f*)
shows (∃ *k* ≤ *nlength* σ. *nnth* (*pifilt* σ *f*) *i* = *nnth* σ *k*)
using *assms sfxfilt-pifilt-nnth-ndropn*[*of* σ *f i*]
by (*simp add*: *pifilt-def*)
(*metis enat-the-enat infinity-ileE linorder-le-cases ndropn-all ndropn-nfirst*)

lemma *sfxfilt-nlength-imp*:
assumes (∃ *i*≤*nlength* σ. *f* (*ndropn i* σ) ∧ *g* (*ndropn i* σ))
shows *nlength* (*sfxfilt* σ (*LIFT*(*f* ∧ *g*))) ≤ *nlength* (*sfxfilt* σ *f*)
proof −
have *1*: *nlength* (*sfxfilt* σ (*LIFT*(*f* ∧ *g*))) =
nlength (*nfilter* (λ*ys*. *f ys* ∧ *g ys*) (*ndropns* σ))
by (*simp add*: *sfxfilt-def*)
have *2*: *nlength* (*nfilter f* (*ndropns* σ)) = *nlength* (*sfxfilt* σ *f*)
by (*simp add*: *sfxfilt-def*)
have *3*: ∃ *x* ∈ *nset* (*ndropns* σ). *f x* ∧ *g x*
using *assms in-nset-ndropns* **by** *blast*
have *4*: *nlength* (*nfilter* (λ*x*. *f x* ∧ *g x*) (*ndropns* σ)) ≤ *nlength* (*nfilter f* (*ndropns* σ))
using *3 nfilter-nlength-imp*[*of ndropns* σ *f g*] **by** *auto*
show *?thesis* **using** *1 2 4* **by** *auto*
qed

lemma *pifilt-nlength-imp*:
assumes ($\exists$ *i*$\leq$*nlength* σ. *f* (*ndropn i* σ) $\wedge$ *g* (*ndropn i* σ))
shows *nlength* (*pifilt* σ (*LIFT*(*f* $\wedge$ *g*))) $\leq$ *nlength* (*pifilt* σ *f*)
using *assms*
by (*simp add*: *sfxfilt-nlength-imp pifilt-def*)

lemma *nellist-nset-sfxfilt* [*simp*]:
assumes ($\exists$ *i* $\leq$ *nlength xs*. *f* (*ndropn i xs*))
shows (*nset* (*sfxfilt xs f*)) = {*ys*. *ys* $\in$ *nset* (*ndropns xs*) $\wedge$ *f* (*ys*)}
using *assms*
proof −
have *1*: *nset* (*sfxfilt xs f*) = *nset* (*nfilter f* (*ndropns xs*))
by (*simp add*: *sfxfilt-def*)
have *2*: $\exists$ *x* $\in$ *nset* (*ndropns xs*). *f x*
using *assms* **using** *in-nset-ndropns* **by** *blast*
have *3*: *nset* (*nfilter f* (*ndropns xs*)) = {*ys*. *ys* $\in$ *nset* (*ndropns xs*) $\wedge$ *f ys*}
using *2 nset-nfilter*[*of ndropns xs f*] **by** *auto*
show *?thesis* **by** (*simp add*: *1 3*)
qed

lemma *subset-nfilter*:
assumes $\exists$ *x* $\in$ *nset xs*. *P x*
shows *nset*(*nfilter P xs*) $\leq$ *nset*(*nfilter* (λ*x*. *P x* $\vee$ *Q x*) *xs*)
proof −
have *1*: $\exists$ *x* $\in$ *nset xs*. *P x* $\vee$ *Q x*
using *assms* **by** *blast*
have *2*: *nset*(*nfilter* (λ*x*. *P x* $\vee$ *Q x*) *xs*) = {*x* $\in$ *nset xs*. *P x* $\vee$ *Q x*}
using *assms nset-nfilter*[*of xs* (λ*x*. *P x* $\vee$ *Q x*)] **by** *blast*
have *3*: *nset*(*nfilter P xs*) = {*x*$\in$ *nset xs*. *P x*}
using *assms nset-nfilter*[*of xs P*] **by** (*simp add*: *Collect-conj-eq*)
have *4*: {*x*$\in$ *nset xs*. *P x*} $\leq$ {*x* $\in$ *nset xs*. *P x* $\vee$ *Q x*}
by *auto*
show *?thesis* **by** (*simp add*: *2 3 4*)
qed

lemma *nellist-subset-sfxfilt* [*simp*]:
assumes ($\exists$ *i* $\leq$ *nlength xs*. *f* (*ndropn i xs*))
shows (*nset* (*sfxfilt xs f*)) $\leq$ (*nset* (*sfxfilt xs* (*LIFT*(*f* $\vee$ *g*))))
proof −
have *1*: $\exists$ *x* $\in$ *nset* (*ndropns xs*). *f x*
using *assms* **using** *in-nset-ndropns* **by** *blast*
have *2*: *nset* (*sfxfilt xs f*) = *nset* (*nfilter f* (*ndropns xs*))
by (*simp add*: *sfxfilt-def*)
have *3*: *nset* (*sfxfilt xs* (*LIFT*(*f* $\vee$ *g*))) = *nset* (*nfilter* (λ*x*. *f x* $\vee$ *g x*) (*ndropns xs*))
by (*simp add*: *sfxfilt-def*)
have *4*: *nset* (*nfilter f* (*ndropns xs*)) $\leq$ *nset* (*nfilter* (λ*x*. *f x* $\vee$ *g x*) (*ndropns xs*))
using *1 subset-nfilter*[*of ndropns xs f g*] **by** *auto*
show *?thesis* **using** *2 3 4* **by** *blast*
qed

lemma *nellist-nset-nmap*:
$(nset\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ xs)) = \{(nnth\ ys\ 0)\ |\ ys.\ ys \in nset\ xs\}$
by (*auto simp add*: *in-nset-conv-nnth nset-conv-nnth*)
(*metis nnth-nmap*)

lemma *nellist-nset-pifilt* [*simp*]:
assumes $(\exists\ i \leq nlength\ xs.\ f\ (ndropn\ i\ xs))$
shows $(nset\ (pifilt\ xs\ f)) = \{(nnth\ ys\ 0)|\ ys.\ ys \in nset(ndropns\ xs) \wedge f\ (ys)\}$
proof −
have *1*: $(nset\ (pifilt\ xs\ f)) = (nset\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ xs\ f)))$
by (*simp add*: *pifilt-def*)
have *2*: $(nset\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ xs\ f))) =$
$\{(nnth\ ys\ 0)\ |\ ys.\ ys \in nset\ (sfxfilt\ xs\ f)\}$
using *nellist-nset-nmap*[*of sfxfilt xs f*] **by** *auto*
have *3*: $\{(nnth\ ys\ 0)\ |\ ys.\ ys \in nset\ (sfxfilt\ xs\ f)\} =$
$\{(nnth\ ys\ 0)|\ ys.\ ys \in nset(ndropns\ xs) \wedge f\ (ys)\}$
using *assms* **by** *auto*
show *?thesis* **using** *1 2 3* **by** *auto*
qed

lemma *nellist-nnth-sfxfilt-in-nset*:
$x \in nset\ (sfxfilt\ \sigma\ (LIFT(f \vee g))) =$
$(\exists\ i \leq nlength\ (sfxfilt\ \sigma\ (LIFT(f \vee g))).\ x =(nnth\ (sfxfilt\ \sigma\ (LIFT(f \vee g)))\ i))$
by (*metis in-nset-conv-nnth*)

lemma *nfilter-nnth-or*:
assumes $\exists\ x \in nset\ xs.\ P\ x$
shows $\exists\ x \in nset(nfilter\ (\lambda x.\ P\ x \vee\ Q\ x)\ xs).\ P\ x$
proof −
have *0*: $\exists\ x \in nset\ xs.\ P\ x \vee\ Q\ x$
using *assms* **by** *auto*
have *1*: $\exists i \leq nlength(nfilter\ (\lambda x.\ P\ x \vee\ Q\ x)\ xs).\ P\ (nnth\ (nfilter\ P\ xs)\ i)$
using *assms nkfilter-nnth-aa*[*of xs* (*λx. P x*)] *nkfilter-nnth-aa*
by (*metis* (*mono-tags*, *lifting*) *le-cases le-zero-eq zero-enat-def*)
obtain *i* **where** *2*: $i \leq nlength(nfilter\ (\lambda x.\ P\ x \vee\ Q\ x)\ xs) \wedge\ P\ (nnth\ (nfilter\ P\ xs)\ i) \wedge$
$(nnth\ (nfilter\ P\ xs)\ i) \in nset\ (nfilter\ P\ xs)$
by (*metis 1 in-nset-conv-nnth le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have *3*: $nset\ (nfilter\ P\ xs) \leq nset(nfilter\ (\lambda x.\ P\ x \vee\ Q\ x)\ xs)$
using *assms subset-nfilter*[*of xs P Q*] **by** *auto*
have *4*: $(nnth\ (nfilter\ P\ xs)\ i) \in nset\ (nfilter\ P\ xs)$
using *2* **by** *auto*
have *5*: $(nnth\ (nfilter\ P\ xs)\ i) \in nset(nfilter\ (\lambda x.\ P\ x \vee\ Q\ x)\ xs)$
using *3 4* **by** *blast*
show *?thesis* **using** *2 5* **by** *blast*
qed

lemma *sfxfilt-nnth-or*:
assumes $(\exists\ i \leq nlength\ \sigma.\ (ndropn\ i\ \sigma) \models f)$
shows $(\exists\ i \leq nlength\ (sfxfilt\ \sigma\ (LIFT(f \vee g))).\ (nnth\ (sfxfilt\ \sigma\ (LIFT(f \vee g))\ )\ i) \models f)$
proof −

have *1*: $\exists\ x \in nset\ (ndropns\ \sigma).\ f\ x$
  using *assms* **by** (*simp add*: *sfxfilter-help*)
have *2*: $sfxfilt\ \sigma\ (LIFT(f \lor g)) = nfilter\ (\lambda\ x.\ f\ x \lor g\ x)\ (ndropns\ \sigma)$
  by (*simp add*: *sfxfilt-def*)
have *3*: $\exists\ x \in nset(nfilter\ (\lambda\ x.\ f\ x \lor g\ x)\ (ndropns\ \sigma)).\ f\ x$
  using *1 nfilter-nnth-or*[*of ndropns σ f g*] **by** *auto*
from *2 3* **show** *?thesis*
  by (*metis* (*full-types*) *in-nset-conv-nnth*)
qed

lemma *NotPiFalse*:
$\sigma \models \neg\ ((\#False)\ \Pi\ f)$
by (*simp add*: *pi-d-def*)

lemma *pifilt-true*:
$pifilt\ \sigma\ (LIFT(\#True)) = \sigma$
by (*simp add*: *pifilt-def sfxfilt-def nmap-first-ndropns*)

lemma *pifilt-state-help*:
$(\exists\ x \in nset\ (ndropns\ xs)\ .\ (LIFT(init\ w))\ x) = (\exists\ x \in nset\ xs.\ w\ ((NNil\ x)))$
proof (*auto simp add*: *init-defs*)
show $\bigwedge x.\ \ x \in nset\ (ndropns\ xs) \Longrightarrow w\ (NNil\ (nfirst\ x)) \Longrightarrow \exists x{\in}nset\ xs.\ w\ (NNil\ x)$
using *in-nset-ndropns* **by** (*metis in-nset-conv-nnth ndropn-nfirst*)
show $\bigwedge x.\ x \in nset\ xs \Longrightarrow w\ (NNil\ x) \Longrightarrow \exists x{\in}nset\ (ndropns\ xs).\ w\ (NNil\ (nfirst\ x))$
by (*metis in-nset-ndropns-nhd ndropn-0 ndropn-nfirst*)
qed

lemma *pifilt-init*:
shows $(pifilt\ xs\ (\lambda s.\ s \models init\ w)) = nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ xs$
proof (*simp add*: *init-defs pifilt-def sfxfilt-def*)
show $nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ (\lambda s.\ w\ (NNil\ (nfirst\ s)))\ (ndropns\ xs)) =$
    $nfilter\ (\lambda y.\ w\ (NNil\ y))\ xs$
proof −
have *1*: $\bigwedge x.\ ((\lambda y.\ w\ ((NNil\ y))) \circ (\lambda s.\ nnth\ s\ 0))\ x = (\lambda s.\ w\ (NNil\ (nfirst\ s)))\ x$
  by *simp* (*metis ndropn-0 ndropn-nfirst*)
have *2*: $((\lambda y.\ w\ ((NNil\ y))) \circ (\lambda s.\ nnth\ s\ 0)) = (\lambda s.\ w\ (NNil\ (nfirst\ s)))$
  using *1* **by** *blast*
have *4*: $nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ (\lambda s.\ w\ (NNil\ (nfirst\ s)))\ (ndropns\ xs)) =$
    $nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (ndropns\ xs))$
  by (*metis* (*mono-tags*, *lifting*) *1 nfilter-cong nfilter-nmap*)
have *5*: $(nmap\ (\lambda s.\ nnth\ s\ 0)\ (ndropns\ xs)) = xs$
  by (*simp add*: *nmap-first-ndropns*)
show *?thesis*
  by (*simp add*: *4 5*)
qed
qed

lemma *pifilt-init-a*:
shows $(pifilt\ xs\ (\lambda s.\ w\ (NNil\ (nnth\ s\ 0))\ )) = nfilter\ (\lambda y.\ w\ (NNil\ y))\ xs$
proof −

have *2*: (*pifilt xs* (λ*s. s* ⊨ *init w*)) = (*pifilt xs* (λ*s. w* (*NNil* (*nnth s 0*))))
  by (*metis* (*no-types*, *lifting*) *in-nset-ndropns init-defs ndropn-nfirst nfilter-cong nnth-zero-ndropn ntaken-0 pifilt-def sfxfilt-def*)
show *?thesis* **using** *pifilt-init 2* **by** (*metis*)
qed

lemma *pifilt-pifilt* :
assumes (∃ *i*≤*nlength xs. f* (*ndropn i xs*))
  (∃ *i*≤*nlength* (*pifilt xs f*). *w* (*NNil* (*nnth* (*ndropn i* (*pifilt xs f*)) *0*)))
shows (*pifilt* (*pifilt xs f*) (*LIFT*(*init w*))) = *pifilt xs* (*LIFT*(*f* ∧ *init w*))
proof −
have *1*: ∃ *i*≤*nlength* (*pifilt xs f*). (*LIFT*(*init w*)) (*ndropn i* (*pifilt xs f*))
  using *assms* **by** (*simp add*: *init-defs ndropn-nfirst*)
have *2*: (*pifilt* (*pifilt xs f*) (*LIFT*(*init w*))) = *nfilter* (λ*y. w* ((*NNil y*))) (*pifilt xs f*)
  using *1 pifilt-init*[*of* (*pifilt xs f*) *w*] **by** *auto*
have *3*: (*pifilt xs f*) = *nmap* (λ*s. nnth s 0*) (*sfxfilt xs f*)
  by (*simp add*: *assms sfxfilt-pifilt*)
have *4*: (*sfxfilt xs f*) = *nfilter* (λ *ys. f ys*) (*ndropns xs*)
  using *sfxfilt-def* **by** *blast*
have *5*: (*pifilt xs f*) = *nmap* (λ*s. nnth s 0*) (*nfilter* (λ *ys. f ys*) (*ndropns xs*))
  by (*simp add*: *3 4*)
have *6*: *nfilter* (λ*y. w* (*NNil y*)) (*pifilt xs f*) =
  nfilter (λ*y. w* (*NNil y*)) (*nmap* (λ*s. nnth s 0*) (*nfilter* (λ *ys. f ys*) (*ndropns xs*)))
  using *5* **by** *simp*
have *7*: *nfilter* (λ*y. w* (*NNil y*)) (*nmap* (λ*s. nnth s 0*) (*nfilter* (λ *ys. f ys*) (*ndropns xs*))) =
  nmap (λ*s. nnth s 0*) (*nfilter* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) (*nfilter* (λ *ys. f ys*) (*ndropns xs*)))
  using *assms 3 4* **by** (*simp add*: *nfilter-nmap pifiltinit-help*)
have *8*: ∃ *x*∈*nset* (*nfilter* (λ *ys. f ys*) (*ndropns xs*)). ((λ*y. w* ((*NNil y*)))∘(λ*s. nnth s 0*)) *x*
  using *assms 3 4 in-nset-conv-nnth*[*of -* (*nfilter f* (*ndropns xs*))]
  by *simp*
  (*metis in-nset-conv-nnth ndropn-0 ndropn-nfirst nlength-nmap nnth-nmap*)
have *9*: ∃ *x*∈*nset* (*ndropns xs*). (λ *ys. f ys*) *x*
  using *assms* **by** (*simp add*: *sfxfilter-help*)
have *10*: ∃ *x*∈*nset* (*ndropns xs*). ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) *x* ∧ (λ *ys. f ys*) *x*
  using *8 9*
  using *exist-conj*[*of f* (*ndropns xs*) ((λ*y. w* (*NNil y*)) ∘ (λ*s. nnth s 0*))] **by** *fastforce*
have *11*: *nfilter* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) (*nfilter* (λ *ys. f ys*) (*ndropns xs*)) =
  nfilter (λ *zs.* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) *zs* ∧ (λ *ys. f ys*) *zs*) (*ndropns xs*)
  using *nfilter-nfilter*[*of* (λ *ys. f ys*) (*ndropns xs*) ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*))]
  8 10 **by** *blast*
have *12*: ∃ *i*≤*nlength xs.* (λ *zs.* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) *zs* ∧ (λ *ys. f ys*) *zs*) (*ndropn i xs*)
  by (*metis 10 in-nset-ndropns*)
have *13*: (*nfilter* (λ *zs.* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) *zs* ∧ (λ *ys. f ys*) *zs*) (*ndropns xs*))
  = (*sfxfilt xs* (λ *zs.* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) *zs* ∧ (λ *ys. f ys*) *zs*))
  by (*simp add*: *sfxfilt-def*)
have *14*: ∃ *i*≤*nlength xs.* (λ *zs.* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) *zs* ∧ (λ *ys. f ys*) *zs*) (*ndropn i xs*)
  using *12* **by** *blast*
have *15*: *nmap* (λ*s. nnth s 0*)
  ((*sfxfilt xs* (λ *zs.* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) *zs* ∧ (λ *ys. f ys*) *zs*))) =
  pifilt xs (λ *zs.* ((λ*y. w* (*NNil y*))∘(λ*s. nnth s 0*)) *zs* ∧ (λ *ys. f ys*) *zs*)

using *14* **by** (*simp add*: *sfxfilt-pifilt*)
have *16*: $\bigwedge$*xs* . (λ *zs*. ((λ*y*. *w* (*NNil y*))$\circ$(λ*s*. *nnth s 0*)) *zs* $\wedge$ (λ *ys*. *f ys*) *zs*) *xs* =
(*LIFT*(*f* $\wedge$ *init w*)) *xs*
by (*simp add*: *init-defs*) (*metis ndropn-0 ndropn-nfirst*)
have *17*: *pifilt xs* (λ *zs*. ((λ*y*. *w* (*NNil y*))$\circ$(λ*s*. *nnth s 0*)) *zs* $\wedge$ (λ *ys*. *f ys*) *zs*) =
pifilt xs (*LIFT*(*f* $\wedge$ *init w*))
using *16* **by** *presburger*
show *?thesis*
using *11 13 15 17 2 3 4 7* **by** *auto*
qed

lemma *PiStatesem*:
($\sigma \models$ (*init w*) Π *f*) =
(($\exists$ *i*$\leq$*nlength* σ. *w* (*NNil* (*nnth* σ *i*))) $\wedge$ *f* (*nfilter* (λ*y*. *w* ((*NNil y*))) σ))
by (*simp add*: *pi-d-def init-defs ndropn-nfirst pifilt-init*)

13.3 Soundness of Axioms

13.3.1 PiK

lemma *PiKsem*:
$\sigma \models$ *f1* Π^u (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*f1* Π^u *f* $\longrightarrow$ *f1* Π^u *g*)
by (*simp add*: *upi-d-def init-defs pi-d-def*) *auto*

lemma *PiK*:
$\vdash$ *f1* Π^u (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*f1* Π^u *f* $\longrightarrow$ *f1* Π^u *g*)
using *PiKsem Valid-def* **by** *blast*

13.3.2 PiDc

lemma *PiDcsem*:
$\sigma \models$ *f* Π *g* $\longrightarrow$ *f* Π^u *g*
by (*simp add*: *upi-d-def init-defs pi-d-def*)

lemma *PiDc*:
$\vdash$ *f* Π *g* $\longrightarrow$ *f* Π^u *g*
using *PiDcsem Valid-def* **by** *blast*

13.3.3 PiN

lemma *PiN*:
assumes $\vdash$ *g*
shows $\vdash$ *f* Π^u *g*
using *assms* **by** (*simp add*: *Valid-def pi-d-def upi-d-def*)

13.3.4 PiTrueEqvDiamond

lemma *PiTrueEqvDiamond*:
$\vdash$ *f* Π #*True* = $\Diamond$ *f*
by (*simp add*: *Valid-def pi-d-def itl-defs*)

13.3.5 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*

13.3.6 UPiFalseEqvBoxNot:

lemma *UPiFalseEqvBoxNot*:
$\vdash f\ \Pi^u\ \#False = \Box\ (\neg f)$
by (*simp add*: *Valid-def upi-d-def pi-d-def itl-defs*)

13.3.7 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*: $\forall\, n \leq nlength\ \sigma.\ f1\ (ndropn\ n\ \sigma) = f2\ (ndropn\ n\ \sigma)$
   using *assms* **by** (*simp add*: *itl-defs*)
  have *2*: $(\sigma \models (f1\ \Pi\ g)) = ((\exists\, i \leq nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f1))$
   by (*simp add*: *pi-d-def*)
  have *3*: $(\exists\, i \leq nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) = (\exists\, i \leq nlength\ \sigma.\ f2\ (ndropn\ i\ \sigma))$
   using *1* **by** *blast*
  have *6*: $(\exists\, i \leq nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) \Longrightarrow$
     $\{\ (ndropn\ n\ \sigma) \mid n.\ n \leq nlength\ \sigma \wedge f1\ (ndropn\ n\ \sigma)\} =$
     $\{\ (ndropn\ n\ \sigma) \mid n.\ n \leq nlength\ \sigma \wedge f2\ (ndropn\ n\ \sigma)\}$
   using *1* **by** *blast*
  have *7*: $(\exists\, i \leq nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) \Longrightarrow (sfxfilt\ \sigma\ f1) = (sfxfilt\ \sigma\ f2)$
   by (*metis 1 in-nset-ndropns nfilter-cong sfxfilt-def*)
  have *8*: $((\exists\, i \leq nlength\ \sigma.\ f1\ (ndropn\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f1)) =$
     $((\exists\, i \leq nlength\ \sigma.\ f2\ (ndropn\ i\ \sigma)) \wedge g\ (pifilt\ \sigma\ f2))$
      by (*metis 3 7 pifilt-def*)
  show *?thesis*
   by (*simp add*: *8 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*)

13.3.8 PiDiamondImpDiamond

lemma *PiDiamondImpDiamondsem*:
 $\sigma \models f\ \Pi\ (\Diamond\ (init\ w)) \longrightarrow \Diamond\ (init\ w)$
by (*simp add*: *Valid-def pi-d-def itl-defs ndropn-nfirst*)
 (*metis pifilt-nnth*)

lemma *PiDiamondImp*:

$\vdash f \;\Pi\; (\Diamond\; (init\; w)) \longrightarrow \Diamond\; (init\; w)$
using *PiDiamondImpDiamondsem Valid-def* **by** *blast*

13.3.9 PiAssoc

lemma *PiAssocsem1*:
assumes $i \le nlength\ \sigma$
$f\ (ndropn\ i\ \sigma)$
$ia \le nlength\ (pifilt\ \sigma\ f)$
$w\ (NNil\ (nnth\ (pifilt\ \sigma\ f)\ ia))$
shows $\exists\, i{\le}nlength\ \sigma.\ f\ (ndropn\ i\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ i\ \sigma)\ 0))$
proof –
have *1*: $(nnth\ (pifilt\ \sigma\ f)\ ia) = (nnth\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ ia)$
using *assms(1) assms(2) assms(3) sfxfilt-pifilt-nnth* **by** *blast*
have *2*: $(nnth\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ ia) =$
$(\lambda s.\ nnth\ s\ 0)\ (nnth\ (sfxfilt\ \sigma\ f)\ ia)$
by (*metis assms(3) nlength-nmap nnth-nmap pifilt-def*)
have *3*: $f\ (nnth\ (sfxfilt\ \sigma\ f)\ ia)$
using *sfxfilt-nnth*
by (*metis assms(1) assms(2) assms(3) nlength-nmap pifilt-def*)
have *4*: $nnth\ (sfxfilt\ \sigma\ f)\ ia = ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ ia)\ \sigma$
using *sfxfilt-pifilt-nnth-ndropn assms(1) assms(2) assms(3)* **by** *blast*
have *5*: $w\ (NNil\ (nnth\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ ia)\ \sigma)\ \ 0))$
using *1 2 4 assms(4)* **by** *auto*
show *?thesis* **by** (*metis 3 4 5 enat-ile le-cases ndropn-all*)
qed

lemma *PiAssocsem2*:
assumes $i \le nlength\ \sigma$
$f\ (ndropn\ i\ \sigma)$
$w\ (NNil\ (nnth\ \sigma\ i))$
shows $\exists\, j{\le}nlength\ (pifilt\ \sigma\ f).\ w\ (NNil\ (nnth\ (pifilt\ \sigma\ f)\ j))$
proof –
have *1*: $\exists\, j{\le}nlength\ (sfxfilt\ \sigma\ f).\ f\ (nnth\ (sfxfilt\ \sigma\ f)\ j)$
using *assms pifilt-exists* **by** *blast*
have *2*: $(LIFT\ (init\ w))\ (ndropn\ i\ \sigma)$
by (*simp add: assms init-defs ndropn-nfirst*)
have *3*: $\exists\, j{\le}nlength\ (sfxfilt\ \sigma\ (LIFT(init\ w))).\ (LIFT(init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT(init\ w)))\ j)$
using *pifilt-exists 2 assms* **by** *blast*
have *4*: $(LIFT\ (f \wedge init\ w))\ (ndropn\ i\ \sigma)$
by (*simp add: 2 assms*)
have *5*: $\exists\, j{\le}nlength\ (sfxfilt\ \sigma\ (LIFT(f \wedge init\ w))).$
$(LIFT(f \wedge init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT(f \wedge init\ w)))\ j)$
using *pifilt-exists 4 assms* **by** *blast*
have *6*: $\exists\, i{\le}nlength\ \sigma.\ ndropn\ i\ \sigma \models f \wedge init\ w$
using *4 assms* **by** *blast*
have *7*: $\exists\, j{\le}nlength\ (sfxfilt\ \sigma\ (LIFT((f \wedge init\ w) \vee (f \wedge \neg\ (init\ w)))\ \ )).$
$(LIFT(f \wedge init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT((f \wedge init\ w) \vee (f \wedge \neg(init\ w)))\ ))\ j)$
using *6 sfxfilt-nnth-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* σ (*LIFT*(($f \wedge$ *init w*) $\vee$ ($f \wedge \neg$ (*init w*))))) =
    (*sfxfilt* σ f)
  using *8* **by** (*simp add*: *sfxfilt-def*)
have *10*: $\exists$ *j*$\leq$*nlength* (*sfxfilt* σ f).
    (*LIFT*($f \wedge$ *init w*)) (*nnth* (*sfxfilt* σ f) *j*)
  using *7 9* **by** *auto*
have *11*: *nlength* (*sfxfilt* σ f) = *nlength* (*pifilt* σ f)
  by (*simp add*: *pifilt-def*)
have *12*: $\exists$ *j*$\leq$*nlength* (*pifilt* σ f).
    (*LIFT*(*init w*)) (*nnth* (*sfxfilt* σ f) *j*)
  using *10 11* **by** *auto*
from *12 11* **show** *?thesis* **unfolding** *pifilt-def init-defs*
  by (*metis nlast-NNil nnth-nmap ntaken-0 ntaken-nlast*)
qed

lemma *PiAssocsema*:
((∃ *i*$\leq$*nlength* σ. f (*ndropn i* σ)) $\wedge$
  (∃ *i*$\leq$*nlength* (*pifilt* σ f). *w* (*NNil* (*nnth* (*ndropn i* (*pifilt* σ f)) *0*)))) =
 (∃ *i*$\leq$*nlength* σ. f (*ndropn i* σ) $\wedge$ *w* (*NNil* (*nnth* (*ndropn i* σ) *0*)))
using *PiAssocsem1*[*of* - σ f - *w*] *PiAssocsem2*[*of* - σ f *w*] **by** *fastforce*

lemma *PiAssocsemb*:
((∃ *i*$\leq$*nlength* σ. f (*ndropn i* σ)) $\wedge$
  (∃ *i*$\leq$*nlength* (*pifilt* σ f). (*LIFT*(*init w*)) (*ndropn i* (*pifilt* σ f)))) =
 (∃ *i*$\leq$*nlength* σ. (*LIFT*($f \wedge$ *init w*)) (*ndropn i* σ))
using *PiAssocsem1*[*of* - σ f - *w*] *PiAssocsem2*[*of* - σ f *w*]
by (*simp add*: *init-defs ndropn-nfirst*) *blast*

lemma *PiAssocsem*:
$\sigma \models f\ \Pi$ ((*init w*) Π *g*) = ($f \wedge$ (*init w*)) Π *g*
proof (*auto simp add*: *pi-d-def init-defs*)
fix *i*
fix *ia*
assume *a0*: *g* (*pifilt* (*pifilt* σ f) (*LIFT*(*init w*)))
assume *a1*: (*enat i*) $\leq$ *nlength* σ
assume *a2*: f (*ndropn i* σ)
assume *a3*: (*enat ia*) $\leq$ *nlength* (*pifilt* σ f)
assume *a4*: *w* (*NNil* (*nfirst* (*ndropn ia* (*pifilt* σ f))))
show ∃ *i*. *enat i* $\leq$ *nlength* σ $\wedge$ f (*ndropn i* σ) $\wedge$ *w* (*NNil* (*nfirst* (*ndropn i* σ)))
 using *a0 a1 a2 a3 a4 PiAssocsem1*
 by (*metis ndropn-nfirst nnth-zero-ndropn*)
next
fix *i*
fix *ia*
assume *a0*: *g* (*pifilt* (*pifilt* σ f) (*LIFT*(*init w*)))
assume *a1*: (*enat i*) $\leq$ *nlength* σ
assume *a2*: f (*ndropn i* σ)

assume *a3*: $(enat\ ia) \leq nlength\ (pifilt\ \sigma\ f)$
assume *a4*: $w\ (NNil\ (nfirst\ (ndropn\ ia\ (pifilt\ \sigma\ f))))$
show $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$
using *a0 a1 a2 a3 a4* **by** (*metis ndropn-nfirst nnth-zero-ndropn pifilt-pifilt*)
next
fix *i*
assume *a0*: $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$
assume *a1*: $(enat\ i) \leq nlength\ \sigma$
assume *a2*: $f\ (ndropn\ i\ \sigma)$
assume *a3*: $w\ (NNil\ (nfirst\ (ndropn\ i\ \sigma)))$
show $\exists\, i.\ enat\ i \leq nlength\ (pifilt\ \sigma\ f) \wedge w\ (NNil\ (nfirst\ (ndropn\ i\ (pifilt\ \sigma\ f))))$
using *a0 a1 a2 a3* **by** (*metis PiAssocsem2 ndropn-nfirst*)
next
fix *i*
assume *a0*: $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$
assume *a1*: $(enat\ i) \leq nlength\ \sigma$
assume *a2*: $f\ (ndropn\ i\ \sigma)$
assume *a3*: $w\ (NNil\ (nfirst\ (ndropn\ i\ \sigma)))$
show $g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w)))$
using *a0 a1 a2 a3* **by** (*metis PiAssocsem2 ndropn-nfirst nnth-zero-ndropn pifilt-pifilt*)
qed

lemma *PiAssoc*:
$\vdash f\ \Pi\ ((init\ w)\ \Pi\ \ g) = \ (f \wedge (init\ w))\ \Pi\ g$
using *PiAssocsem Valid-def* **by** *blast*

13.3.10 PiNotEqvDiamondAndNotPi

lemma *PiNotEqvDiamondAndNotPisem*:
$\sigma \models f\ \Pi\ (\neg\ g) = (\Diamond f \wedge \neg(f\ \Pi\ g))$
by (*simp add*: *pi-d-def itl-defs*) *blast*

lemma *PiNotEqvDiamondAndNotPi*:
$\vdash f\ \Pi\ (\neg\ g) = (\Diamond f \wedge \neg(f\ \Pi\ g))$
using *PiNotEqvDiamondAndNotPisem Valid-def* **by** *blast*

13.3.11 PiSChopDist

lemma *PiSChopDistsema*:
assumes $(\sigma \models (init\ w)\ \Pi\ (g \frown h))$
shows $(\sigma \models ((init\ w)\ \Pi\ g) \frown ((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
proof −
have *1*: $(\sigma \models (init\ w)\ \Pi\ (g \frown h))$
using *assms* **by** *auto*
have *2*: $((\exists\, i \leq nlength\ \sigma.\ (LIFT(init\ w))\ (ndropn\ i\ \sigma)) \wedge$
$((pifilt\ \sigma\ (LIFT(init\ w))) \models g \frown h)$
$)$
using *1* **by** (*simp add*: *pi-d-def*)
have *3*: $(\exists\, i \leq nlength\ \sigma.\ (LIFT(init\ w))\ (ndropn\ i\ \sigma))$
using *2* **by** *auto*
have *4*: $((pifilt\ \sigma\ (LIFT(init\ w))) \models g \frown h)$

using *2* **by** *auto*
have *5*: *nfilter* (*λy. w* ((*NNil y*))) *σ* ⊨ *g*⌢*h*
  using *pifilt-init* **by** (*metis 4*)
have *6*: (∃ *n*. (*enat n*)≤*nlength* (*nfilter* (*λy. w* (*NNil y*)) *σ*) ∧
  g (*ntaken n* (*nfilter* (*λy. w* (*NNil y*)) *σ*)) ∧
  h (*ndropn n* (*nfilter* (*λy. w* (*NNil y*)) *σ*)))
  using *5* **by** (*simp add*: *itl-defs*)
obtain *n* **where** *7*: (*enat n*)≤*nlength* (*nfilter* (*λy. w* (*NNil y*)) *σ*) ∧
  g (*ntaken n* (*nfilter* (*λy. w* (*NNil y*)) *σ*)) ∧
  h (*ndropn n* (*nfilter* (*λy. w* (*NNil y*)) *σ*))
  using *6* **by** *auto*
have *8*: ∃ *i*≤*nlength σ*. *w* (*NNil* (*nnth σ i*))
  using *3* **using** *init-defs PiStatesem assms* **by** *blast*
have *9*: ∃ *x* ∈ *nset σ*. *w* (*NNil x*)
  using *8* **by** (*simp add*: *pifiltinit-help*)
have *10*: (*ntaken n* (*nfilter* (*λy. w* (*NNil y*)) *σ*)) =
  (*nfilter* (*λy. w* (*NNil y*)) (*ntaken* ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) *σ*))
  by (*simp add*: *7 9 nfilter-nkfilter-ntaken-1 nkfilter-nlength*)
have *11*: (*ndropn n* (*nfilter* (*λy. w* (*NNil y*)) *σ*)) =
  (*nfilter* (*λy. w* (*NNil y*)) (*ndropn* ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) *σ*))
  by (*simp add*: *7 9 nfilter-nkfilter-ndropn-1 nkfilter-nlength*)
have *12*: *g* (*nfilter* (*λy. w* (*NNil y*)) (*ntaken* ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) *σ*))
  using *10 7* **by** *auto*
have *13*: *h* (*nfilter* (*λy. w* (*NNil y*)) (*ndropn* ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) *σ*))
  using *11 7* **by** *auto*
have *14*: ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) ≤ *nlength σ*
  using *7 9 nkfilter-upperbound*[*of σ* (*λy. w* (*NNil y*)) *- 0*]
  by (*metis gen-nlength-def nkfilter-nlength nlength-code*)
have *15*: *w* (*NNil* (*nnth* (*ndropn* ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) *σ*) *0*))
  using *7 9 nkfilter-nmap-nfilter*[*of σ λx. w* (*NNil x*)] *nkfilter-nnth-aa*[*of σ λx. w* (*NNil x*)]
  by *simp*
  (*metis* (*mono-tags*, *lifting*) *7 nkfilter-nlength nnth-nmap*)
have *16*: (∃ *i*. (*enat i*)≤*nlength* (*ntaken* ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) *σ*) ∧
  w (*NNil* (*nnth* (*ndropn i* (*ntaken* ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) *σ*)) *0*)))
  using *14 15*
  by (*metis le-cases min.orderE ndropn-0 ndropn-nfirst ntaken-nlength ntaken-nnth*)
have *17*: (∃ *i*. (*enat i*)≤*nlength* (*ndropn* ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*)) *σ*) ∧
  w (*NNil* (*nnth* (*ndropn* (*i* + ((*nnth* (*nkfilter* (*λy. w* (*NNil y*)) *0 σ*) *n*))) *σ*) *0*)))
  using *15* **by** (*metis add.left-neutral zero-enat-def zero-le*)
have *18*: (∃ *n*. (*enat n*)≤*nlength σ* ∧
  (∃ *i*. (*enat i*) ≤*nlength* (*ntaken n σ*) ∧ *w* (*NNil* (*nnth* (*ndropn i* (*ntaken n σ*)) *0*)) ∧
  g (*nfilter* (*λy. w* ((*NNil y*))) (*ntaken n σ*)) ∧
  w (*NNil* (*nnth* (*ndropn n σ*) *0*)) ∧
  (∃ *i*. (*enat i*) ≤*nlength* (*ndropn n σ*) ∧ *w* (*NNil* (*nnth* (*ndropn* (*i* + *n*) *σ*) *0*))) ∧
  h (*nfilter* (*λy. w* ((*NNil y*))) (*ndropn n σ*))))
  using *12 13 14 15 16 17* **by** *blast*
have *181*: (∃ *n*. (*enat n*)≤*nlength σ* ∧ (∃ *i*. (*enat i*)≤ *nlength* (*ntaken n σ*) ∧ *w* (*NNil* (*nnth* (*ntaken n σ*) *i*))))
  using *18* **by** *auto*
have *182*: (∃ *n*. (*enat n*)≤*nlength σ* ∧ (∃ *i*. (*enat i*)≤ *nlength* (*ndropn n σ*) ∧ *w* (*NNil* (*nnth σ* (*i* + *n*)))

))
  using *18* **by** *auto*
 have *19*: $(\exists n.\ (enat\ n) \leq nlength\ \sigma \wedge$
 $(\exists i.\ (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ i\ (ntaken\ n\ \sigma))\ 0)) \wedge$
 $g\ (pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w))\ ) \wedge$
 $w\ (NNil\ (nnth\ (ndropn\ n\ \sigma)\ 0)) \wedge$
 $(\exists i.\ (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ (i + n)\ \sigma)\ 0)) \wedge$
 $h\ (pifilt\ (ndropn\ n\ \sigma)\ (LIFT(init\ w))))))$
  using *18 pifilt-init*[*of - w*] **by** *auto*
 have *20*: $((\exists n.\ (enat\ n) \leq nlength\ \sigma \wedge$
 $(\exists i.\ (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ntaken\ n\ \sigma))) \wedge$
 $g\ (pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w))) \wedge$
 $(LIFT(init\ w))\ (ndropn\ n\ \sigma) \wedge$
 $(\exists i.\ (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ (i + n)\ \sigma))$
 $\wedge\ h\ (pifilt\ (ndropn\ n\ \sigma)\ (LIFT(init\ w)))))$
  using *19*
  by (*metis init-defs ndropn-nfirst nnth-zero-ndropn ntaken-0*)
 have *21*: $(\sigma \models ((init\ w)\ \Pi\ g)\frown((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
  using *20* **by** (*simp add*: *add.commute itl-defs ndropn-ndropn pi-d-def*)
 show *?thesis* **by** (*simp add*: *21*)
qed

lemma *PiSChopDistsemb*:
 assumes $(\sigma \models ((init\ w)\ \Pi\ g)\frown((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
 shows $(\sigma \models (init\ w)\ \Pi\ (g\frown h))$
proof −
 have *1*: $(\sigma \models ((init\ w)\ \Pi\ g)\frown((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
  using *assms* **by** *auto*
 have *2*: $(\exists n.\ (enat\ n) \leq nlength\ \sigma \wedge$
 $((ntaken\ n\ \sigma) \models ((init\ w)\ \Pi\ g)) \wedge$
 $((ndropn\ n\ \sigma) \models ((init\ w) \wedge ((init\ w)\ \Pi\ h))))$
  using *assms schop-defs* **by** *blast*
 obtain *n* **where** *3*: $(enat\ n) \leq nlength\ \sigma \wedge ((ntaken\ n\ \sigma) \models ((init\ w)\ \Pi\ g)) \wedge$
 $((ndropn\ n\ \sigma) \models ((init\ w) \wedge ((init\ w)\ \Pi\ h)))$
  using *2* **by** *auto*
 have *4*: $((\exists i.\ (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ntaken\ n\ \sigma))) \wedge$
 $((pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w))) \models g)$
)
  by (*meson 3 pi-d-def*)
 have *5*: $(\exists i.\ (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ntaken\ n\ \sigma)))$
  using *4* **by** *auto*
 have *6*: $g\ (pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w)))$
  using *4* **by** *auto*
 have *7*: $g\ (nfilter\ (\lambda y.\ w\ ((NNil\ y)))\ (ntaken\ n\ \sigma))$
  using *5 6 pifilt-init* **by** (*metis*)
 have *8*: $((\exists i.\ (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ndropn\ n\ \sigma))) \wedge$
 $((pifilt\ (ndropn\ n\ \sigma)\ (LIFT(init\ w))) \models h)$
)
  by (*metis 3 intensional-rews*(*3*) *pi-d-def*)
 have *9*: $(\exists i.\ (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ndropn\ n\ \sigma)))$

using *8* **by** *auto*
have *10*: h (*pifilt* (*ndropn* n σ) (*LIFT*(*init* w)))
using *8* **by** *auto*
have *11*: h (*nfilter* ($\lambda y.$ w ((*NNil* y))) (*ndropn* n σ))
using *10 9 pifilt-init* **by** *metis*
have *12*: ($\lambda y.$ w ((*NNil* y))) (*nlast* (*ntaken* n σ))
by (*metis 3 init-defs intensional-rews*(*3*) *ndropn-nfirst ntaken-0 ntaken-nlast*)
have *13*: $\exists$ $x \in$ *nset* σ. ($\lambda y.$ w ((*NNil* y))) x
using *12 3* **by** (*metis in-nset-conv-nnth ntaken-nlast*)
have *14*: $\exists$ $x \in$ *nset* (*ntaken* n σ) . ($\lambda y.$ w ((*NNil* y))) x
using *12* **using** *nfinite-ntaken nset-nlast* **by** *blast*
have *15*: $\exists$ $x \in$ *nset* (*ndropn* n σ) . ($\lambda y.$ w ((*NNil* y))) x
by (*metis 12 3 dual-order.order-iff-strict ndropn-Suc-conv-ndropn ndropn-nlast nellist.set-intros*(*1*) *nellist.set-intros*(*2*) *nfinite-ntaken ntaken-all ntaken-nlast the-enat.simps*)
have *16*: (*nfilter* ($\lambda y.$ w ((*NNil* y))) (*ntaken* n σ)) =
ntaken (*the-enat* (*nlength* (*nfilter* ($\lambda y.$ w ((*NNil* y))) (*ntaken* n σ))))
(*nfilter* ($\lambda y.$ w ((*NNil* y))) σ)
using *12 13 14 15 3 nfilter-chop1-ntaken*[*of* n σ ($\lambda y.$ w (*NNil* y))] **by** *auto*
have *17*: (*nfilter* ($\lambda y.$ w ((*NNil* y))) (*ndropn* n σ)) =
ndropn (*the-enat* (*nlength* (*nfilter* ($\lambda y.$ w ((*NNil* y))) (*ntaken* n σ))))
(*nfilter* ($\lambda y.$ w ((*NNil* y))) σ)
using *12 13 14 15 3 nfilter-chop1-ndropn*[*of* n σ ($\lambda y.$ w (*NNil* y))] **by** *auto*
have *18*: $\exists n.$ (*enat* n)$\leq$*nlength* (*nfilter* ($\lambda y.$ w (*NNil* y)) σ) $\wedge$
g (*ntaken* n (*nfilter* ($\lambda y.$ w (*NNil* y)) σ)) $\wedge$
h (*ndropn* n (*nfilter* ($\lambda y.$ w (*NNil* y)) σ))
by (*metis 11 16 17 7 enat-the-enat infinity-ileE le-cases ntaken-all*)
have *19*: *nfilter* ($\lambda y.$ w ((*NNil* y))) $\sigma \models g \frown h$
by (*simp add*: *18 schop-defs*)
have *20*: ((*pifilt* σ (*LIFT*(*init* w))) $\models g \frown h$)
by (*metis 19 pifilt-init*)
have *21*: ($\exists i.$ (*enat* i)$\leq$*nlength* σ $\wedge$ (*LIFT*(*init* w)) (*ndropn* i σ))
using *3* **by** *auto*
show *?thesis* **by** (*simp add*: *20 21 pi-d-def*)
qed

lemma *PiSChopDistsem*:
$\sigma \models$ (*init* w) Π ($g \frown h$) = ((*init* w) Π g)$\frown$((*init* w) $\wedge$ ((*init* w) Π h))
using *PiSChopDistsema PiSChopDistsemb unl-lift2* **by** *blast*

lemma *PiSChopDist*:
$\vdash$ (*init* w) Π ($g \frown h$) = ((*init* w) Π g)$\frown$((*init* w) $\wedge$ ((*init* w) Π h))
using *PiSChopDistsem Valid-def* **by** *blast*

13.3.12 PiProp

lemma *Pistate*:
($\sigma \models$ (*init* w) Π f) =
(($\exists$ $x \in$ *nset* σ. w (*NNil* x)) $\wedge$ ((*nfilter* ($\lambda y.$ w (*NNil* y)) σ) $\models f$))
proof –
have *1*: ($\sigma \models$ (*init* w) Π f) =

$((\exists\ i\leq\ nlength\ \sigma.\ \ w\ (NNil\ (nnth\ \sigma\ i))) \wedge ((pifilt\ \sigma\ (LIFT(init\ w))) \models f)\ )$
by (*auto simp add*: *pi-d-def init-defs ndropn-nfirst*)
have *2*: $(\exists\, i\leq nlength\ \sigma.\ (LIFT(init\ w))\ (ndropn\ i\ \sigma)) =$
$(\exists\ \ i\leq\ nlength\ \sigma.\ \ w\ (NNil\ (nnth\ \sigma\ i)))$
by (*auto simp add*: *init-defs ndropn-nfirst*)
have *3*: $(\exists\ \ i\leq\ nlength\ \sigma.\ \ w\ (NNil\ (nnth\ \sigma\ i))) \longrightarrow$
$(pifilt\ \sigma\ (LIFT(init\ w))) = (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)$
using *pifilt-init* **using** *2* **by** *blast*
have *4*: $(\exists\ \ i.\ (enat\ i)\leq\ nlength\ \sigma \wedge\ \ w\ (NNil\ (nnth\ \sigma\ i))) =(\exists\ \ x \in\ nset\ \sigma.\ w\ (NNil\ x))$
using *in-nset-conv-nnth* **by** *force*
show *?thesis*
using *1 3 4* **by** *auto*
qed

lemma *PiPropsem1a*:
$(\sigma \models f\ \Pi\ \$p) =$
$((\exists\, i.\ (enat\ i)\leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma)) \wedge p\ (nnth\ \sigma\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0))\ )$
using *sfxfilt-pifilt-nnth-ndropn*[*of* σ *f*]
by (*simp add*: *pi-d-def current-val-d-def pifilt-def*)
(*metis ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-order*(*1*))

lemma *PiPropsem2a*:
$(\sigma \models (\neg\ f)\ \mathcal{U}\ (f \wedge \$p)) =$
$(\exists\, k\leq nlength\ \sigma.\ f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall\, j<k.\ \neg\ f\ (ndropn\ j\ \sigma)))$
by (*simp add*: *until-d-def current-val-d-def ndropn-nfirst*)

lemma *PiPropsem3a*:
assumes $(\sigma \models f\ \Pi\ \$p)$
shows $(\sigma \models (\neg\ f)\ \mathcal{U}\ (f \wedge \$p))$
proof –
have *1*: $((\exists\, i.\ (enat\ i)\leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma)) \wedge$
$p\ (nnth\ \sigma\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ )\ )$
using *assms PiPropsem1a* **by** *auto*
have *2*: $\exists\ \ x \in\ nset(ndropns\ \sigma).\ f\ x$
using *1* **by** (*simp add*: *sfxfilter-help*)
have *3*: $\forall\ \ x \in\ nset(nkfilter\ f\ 0\ (ndropns\ \sigma)).\ f\ \ (nnth\ (ndropns\ \sigma)\ x)$
using *2 nkfilter-holds* **by** *fastforce*
have *31*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) \leq\ nlength\ \sigma$
by (*metis 1 2 dual-order.strict-trans2 enat-ord-simps*(*2*) *leI ndropns-nnth nkfilter-not-before*)
have *4*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \ \sigma)$
by (*metis 3 31 in-nset-ndropns in-nset-ndropns-nhd ndropn-0 ndropns-nnth zero-enat-def zero-le*)
have *5*: $(\forall\, j<\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0).\ \neg\ f\ (ndropn\ j\ \sigma))$
using *nkfilter-not-before*[*of ndropns* σ *f*] *2*
by (*simp add*: *31 less-imp-le ndropns-nnth order-less-le-subst2*)
have *6*: $(\exists\, k\leq nlength\ \sigma.\ f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall\, j<k.\ \neg\ f\ (ndropn\ j\ \sigma)))$
using *1 31 4 5* **by** *blast*
show *?thesis* **using** *6 PiPropsem2a* **by** *metis*
qed

lemma *PiPropsem3b*:

assumes $(\sigma \models (\neg\ f)\ \mathcal{U}\ (f \wedge \$p))$
shows $(\sigma \models f\ \Pi\ \$p)$
proof –
have *1*: $(\exists\, k \le nlength\ \sigma.\ f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall\, j<k.\ \neg\ f\ (ndropn\ j\ \sigma)))$
using *assms PiPropsem2a* **by** *auto*
obtain *k* **where** *2*: $k \le nlength\ \sigma \wedge f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall\, j<k.\ \neg\ f\ (ndropn\ j\ \sigma))$
using *1* **by** *auto*
have *3*: $(\exists\, i.\ (enat\ i) \le nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma))$
using *2* **by** *blast*
have *31*: $\exists\, x \in nset(ndropns\ \sigma).\ f\ x$
using *2 in-nset-ndropns* **by** *auto*
have *331*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) \le nlength\ \sigma$
by (*metis 31 gen-nlength-def ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def zero-le*)
have *32*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma)$
using *nkfilter-holds*[*of ndropns σ f 0*] *nkfilter-not-before*[*of ndropns σ f*]
by (*metis 2 31 ndropns-nfilter-nnth sfxfilt-def sfxfilt-pifilt-nnth-ndropn zero-enat-def zero-le*)
have *4*: $p\ (nnth\ \sigma\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ )$
by (*metis 2 31 32 linorder-neqE-nat ndropns-nnth nkfilter-not-before*)
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*

13.3.13 PiNext

lemma *PiNextsem1*:
$(\sigma \models f\ \Pi\ (\bigcirc\ g)) =$
$((\exists\, i \le nlength\ \sigma.\ f\ (ndropn\ i\ \sigma)) \wedge$
$0 < nlength\ (nfilter\ f\ (ndropns\ \sigma)) \wedge$
$g\ (ndropn\ (Suc\ 0)\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))$
using *zero-enat-def* **by** (*simp add*: *pi-d-def itl-defs pifilt-def sfxfilt-def*)

lemma *PiNextsem2*:
$(\sigma \models (\neg\ f)\ \mathcal{U}\ (f \wedge \bigcirc(f\ \Pi\ g))) =$
$(\exists\, k \le nlength\ \sigma.$
$f\ (ndropn\ k\ \sigma) \wedge$
$k < nlength\ \sigma \wedge$
$(\exists\, i \le nlength\ \sigma - Suc\ k.\ f\ (ndropn\ (Suc\ (i + k))\ \sigma)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ (Suc\ k)\ \sigma)))) \wedge (\forall\, j<k.\ \neg\ f\ (ndropn\ j\ \sigma)))$
by (*simp add*: *until-d-def itl-defs pi-d-def pifilt-def sfxfilt-def ndropn-ndropn add.commute*)
(*metis add.right-neutral antisym-conv2 enat.simps*(*3*) *enat-add-sub-same less-eqE zero-enat-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*: $((\exists\, i{\le}nlength\ \sigma.\ f\ (ndropn\ i\ \sigma)) \wedge 0 < nlength\ (nfilter\ f\ (ndropns\ \sigma)) \wedge$
$g\ (ndropn\ (Suc\ 0)\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))$
using *assms* **by** (*simp add*: *PiNextsem1*)
have *2*: $\exists\ x \in nset(ndropns\ \sigma).\ f\ x$
using *1* **by** (*simp add*: *sfxfilter-help*)
have *3*: $\forall\ x \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma)).\ f\ (nnth\ (ndropns\ \sigma)\ x)$
using *2 nkfilter-holds* **by** *fastforce*
have *31*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) \le nlength\ \sigma$
by (*metis 2 gen-nlength-def i0-lb ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def*)
have *4*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \ \sigma)$
by (*metis 3 31 i0-lb in-nset-conv-nnth ndropns-nnth zero-enat-def*)
have *41*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma))$
by (*metis 1 2 One-nat-def eSuc-enat ileI1 in-nset-conv-nnth nkfilter-nlength zero-enat-def*)
have *42*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) \le nlength\ (ndropns\ \sigma)$
by (*metis 2 41 gen-nlength-def in-nset-conv-nnth nkfilter-upperbound nlength-code*)
have *5*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1)\ \ \sigma)$
using *3 41 42* **by** (*metis ndropns-nlength ndropns-nnth*)
have *6*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) < (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1)$
by (*metis 1 2 One-nat-def ileI1 nidx-nkfilter-expand nkfilter-nlength one-eSuc one-enat-def*)
have *7*: $Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0) \le nlength\ \sigma$
by (*metis 42 6 Suc-leI dual-order.trans enat-ord-simps*(*1*) *ndropns-nlength*)
have *8*: $0 < nlength\ (nkfilter\ f\ 0\ (ndropns\ \sigma))$
using *1 2 nkfilter-nlength* **by** *fastforce*
have *9*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) \le nlength\ \sigma$
by (*metis 42 ndropns-nlength*)
have *10*: $(Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)) \le (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1)$
using *6 Suc-leI* **by** *blast*
have *11*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) =$
$(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) - (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)) +$
$(Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0))$
using *10* **by** *auto*
have *12*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) - (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)) \le$
$nlength\ \sigma - Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)$
using *9 diff-le-mono* **by** (*metis enat-minus-mono1 idiff-enat-enat*)
have *13*: $\exists\ i \le nlength\ \sigma - Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0).$
$(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 1) = (Suc\ (i + (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)))$
using *11 12* **by** *auto*
have *14*: $(\exists\, i{\le}nlength\ \sigma - Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0).$
$f\ (ndropn\ (Suc\ (i + (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)))\ \sigma))$
using *13 5* **by** *auto*
have *15*: $(ndropn\ (Suc\ 0)\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))) =$
$(nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0))\ \sigma))))$
using *2 8*
by (*metis One-nat-def ileI1 nfilter-ndropns-nmap nkfilter-nlength one-eSuc one-enat-def*)
have *16*: $g\ (nmap\ (\lambda s\ .\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (Suc\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0))\ \sigma))))$
using *1 15* **by** *auto*

have *17*: ∀ *j*< (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*). ¬ *f* (*ndropn j σ*)
  using *2 31 nkfilter-not-before*[*of* (*ndropns σ*) *f*]
  by (*metis enat-ord-simps*(*1*) *less-imp-le ndropns-nnth order.trans*)
have *18*: (∃ *k*≤*nlength σ*.
    f (*ndropn k σ*) ∧
    k < *nlength σ* ∧
    (∃ *i*≤*nlength σ* − *Suc k*. *f* (*ndropn* (*Suc* (*i* + *k*)) *σ*)) ∧
      g (*nmap* (*λs. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn* (*Suc k*) *σ*)))) ∧
      (∀ *j*<*k*. ¬ *f* (*ndropn j σ*)))
  using *14 16 17 4 7 Suc-ile-eq less-imp-le* **by** *blast*
show *?thesis* **using** *18* **by** (*simp add*: *PiNextsem2*)
qed

lemma *PiNextsem4*:
assumes (*σ* ⊨ (¬ *f*) 𝒰 (*f* ∧ ○(*f* Π *g*)))
shows (*σ* ⊨ *f* Π (○ *g*))
proof −
have *1*: (∃ *k*≤*nlength σ*.
    f (*ndropn k σ*) ∧
    k < *nlength σ* ∧
    (∃ *i*≤*nlength σ* − *Suc k*. *f* (*ndropn* (*Suc* (*i* + *k*)) *σ*)) ∧
      g (*nmap* (*λs. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn* (*Suc k*) *σ*)))) ∧
     (∀ *j*<*k*. ¬ *f* (*ndropn j σ*)))
  using *assms* **by** (*simp add*: *PiNextsem2*)
obtain *k* **where** *2*: *k*≤*nlength σ* ∧ *f* (*ndropn k σ*) ∧ *k* < *nlength σ* ∧ (∃ *i*≤*nlength σ* − *Suc k*.
    f (*ndropn* (*Suc* (*i* + *k*)) *σ*)) ∧
  g (*nmap* (*λs. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn* (*Suc k*) *σ*)))) ∧ (∀ *j*<*k*. ¬ *f* (*ndropn j σ*))
  using *1* **by** *auto*
have *3*: ∃ *x* ∈ *nset* (*ndropns σ*). *f x*
  using *1* **using** *in-nset-ndropns* **by** *auto*
have *4*: ∀ *x* ∈ *nset*(*nkfilter f 0* (*ndropns σ*)). *f* (*nnth* (*ndropns σ*) *x*)
  using *3 nkfilter-holds* **by** *fastforce*
have *41*: (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*) ≤ *nlength σ*
  by (*metis 3 gen-nlength-def le-cases le-zero-eq ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def*)
have *5*: *f* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*) *σ*)
  by (*metis 4 41 i0-lb in-nset-conv-nnth ndropns-nnth zero-enat-def*)
have *6*: *0* < *nlength* (*nfilter f* (*ndropns σ*))
  using *2 3 nfilter-nlength-zero-conv-a*[*of* (*ndropns σ*) *f*]
  by *simp*
    (*metis* (*no-types*, *lifting*) *add.commute eSuc-enat enat.simps*(*3*) *enat-add-sub-same enat-less-enat-plusI2 ileI1 iless-Suc-eq less-add-Suc2 less-eqE ndropn-Suc-conv-ndropn ndropn-nlength ndropns-nlength ndropns-nnth nlength-NCons*)
have *61*: (*nnth* (*nkfilter f 0* (*ndropns σ*)) *1*) ∈ *nset*(*nkfilter f 0* (*ndropns σ*))
  by (*metis 3 6 ileI1 in-nset-conv-nnth nkfilter-nlength one-eSuc one-enat-def*)
have *62*: (*nnth* (*nkfilter f 0* (*ndropns σ*)) *1*) ≤ *nlength* (*ndropns σ*)
  by (*metis 3 61 gen-nlength-def in-nset-conv-nnth nkfilter-upperbound nlength-code*)
have *7*: *f* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *1*) *σ*)
  using *4 61 62* **by** (*metis ndropns-nlength ndropns-nnth*)
have *8*: (∃ *i*≤*nlength σ*. *f* (*ndropn i σ*))

using *1* **by** *blast*
have *9*: *g* (*ndropn* (*Suc 0*) (*nmap* ($\lambda s.$ *nnth s 0*) (*nfilter f* (*ndropns* σ))))
using *2 3 5 6 nkfilter-not-before*[*of* (*ndropns* σ) *f*] *nfilter-ndropns-nmap*[*of* σ *f 0*]
by (*metis One-nat-def ileI1 ndropns-nnth not-less-iff-gr-or-eq one-eSuc one-enat-def*)
have *10*: (($\exists$ *i*. (*enat i*) $\leq$*nlength* σ $\wedge$ *f* (*ndropn i* σ)) $\wedge$
0 < *nlength* (*nfilter f* (*ndropns* σ)) $\wedge$
g (*ndropn* (*Suc 0*) (*nmap* ($\lambda s.$ *nnth s 0*) (*nfilter f* (*ndropns* σ)))))
using *6 8 9* **by** *blast*
show *?thesis* **using** *10*
by (*simp add*: *PiNextsem1*)
qed

lemma *PiNextsem*:
(σ $\models$ *f* Π ($\bigcirc$ *g*) = ($\neg$ *f*) $\mathcal{U}$ (*f* $\wedge$ $\bigcirc$(*f* Π *g*)))
using *PiNextsem3 PiNextsem4 unl-lift2* **by** *blast*

lemma *PiNext*:
$\vdash$ *f* Π ($\bigcirc$ *g*) = ($\neg$ *f*) $\mathcal{U}$ (*f* $\wedge$ $\bigcirc$(*f* Π *g*))
using *PiNextsem Valid-def* **by** *blast*

13.3.14 PiUntil

lemma *PiUntilDistsem1*:
(σ $\models$ *f* Π (*g* $\mathcal{U}$ *h*)) =
(($\exists$ *i*$\leq$*nlength* σ. *f* (*ndropn i* σ)) $\wedge$
($\exists$ *k*$\leq$*nlength* (*nfilter f* (*ndropns* σ)).
h (*ndropn k* (*nmap* ($\lambda s.$ *nnth s 0*) (*nfilter f* (*ndropns* σ)))) $\wedge$
($\forall$ *j*<*k*. *g* (*ndropn j* (*nmap* ($\lambda s.$ *nnth s 0*) (*nfilter f* (*ndropns* σ)))))))
by (*simp add*: *pi-d-def pifilt-def sfxfilt-def until-d-def*)

lemma *PiUntilDistsem2*:
(σ $\models$ (*f* Π *g*) $\mathcal{U}$ (*f* Π *h*)) =
($\exists$ *k*$\leq$*nlength* σ.
($\exists$ *i*$\leq$*nlength* σ $-$ *k*. *f* (*ndropn* (*i* + *k*) σ)) $\wedge$
h (*nmap* ($\lambda s.$ *nnth s 0*) (*nfilter f* (*ndropns* (*ndropn k* σ)))) $\wedge$
($\forall$ *j*<*k*. ($\exists$ *i*$\leq$*nlength* σ $-$ *j*. *f* (*ndropn* (*i* + *j*) σ)) $\wedge$
g (*nmap* ($\lambda s.$ *nnth s 0*) (*nfilter f* (*ndropns* (*ndropn j* σ))))))
by (*simp add*: *until-d-def pi-d-def pifilt-def sfxfilt-def ndropn-ndropn add.commute*)

lemma *cover*:
assumes ($\exists$ *i*<*k*. *ff*(*i*) <(*j*::*nat*) $\wedge$ *j*$\leq$ *ff*(*Suc i*))
($\forall$ *i*<*k*. *ff*(*i*)<*ff*(*Suc i*))
shows *ff*(*0*)<*j* $\wedge$ *j* $\leq$*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*: $(\exists\, i<k.\ ff(i) < (j::nat) \wedge j\leq ff(Suc\ i)) \vee (ff\ (k) < j \wedge j\leq ff(Suc\ k))$
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*: $(ff(0)<j \wedge j \leq ff(k)) \vee (ff\ (k) < j \wedge j\leq ff(Suc\ k))$
using *1 2 Suc.hyps* **by** *blast*
have *41*: $ff(0)<j$
using *2 4 Suc.hyps order-refl* **by** *force*
have *42*: $j \leq ff(Suc\ k)$
using *3 4* **by** *linarith*
have *5*: $ff(0)<j \wedge j \leq ff(Suc\ k)$
by (*simp add*: *41 42*)
show *?thesis*
by (*simp add*: *5*)
qed
qed

lemma *cover-a*:
assumes $(\forall\ j.$
$(j\leq ff\ 0) \vee (\exists\, i<k.\ ff(i) < (j::nat) \wedge j\leq ff(Suc\ i))$
$\longrightarrow gg\ j)$

$(\forall\, i<k.\ ff(i)<ff(Suc\ i))$
shows $(\forall\ j< ff\ k.\ gg\ j)$
proof −
have *1*: $(\forall\ j< ff\ 0.\ gg\ j)$
by (*simp add*: *assms*(*1*))
have *2*: $(\forall\ j.\ ff\ 0 < j \wedge j\leq ff\ k \longrightarrow gg\ j)$
proof
fix *j*
show $ff\ 0 < j \wedge j \leq ff\ k \longrightarrow gg\ j$
using *assms*
proof (*induct k arbitrary*: *j*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
proof −
have *21*: $(\forall\ j.\ (j\leq ff\ 0) \vee (\exists\, i<k.\ ff(i) < (j::nat) \wedge j\leq ff(Suc\ i))$
$\vee\ (ff\ k < j \wedge j \leq ff\ (Suc\ k)\quad) \longrightarrow gg\ j)$
using *Suc.prems*(*1*) *less-SucI* **by** *blast*
have *22*: $(\forall\, i<k.\ ff(i)<ff(Suc\ i))$
using *Suc.prems*(*2*) **by** *auto*
have *23*: $ff\ k < ff\ (Suc\ k)$
by (*simp add*: *Suc.prems*(*2*))
have *24*: $(\forall\ j.$

$(j \leq ff\ 0) \vee (ff\ 0 < j \wedge j \leq ff\ k)$
$\vee (ff\ k < j \wedge j \leq ff\ (Suc\ k)\quad)$
$\longrightarrow gg\ j)$
using *21 22 Suc.hyps* **by** *blast*
have *25*: $ff\ 0 < j \wedge j \leq ff\ (Suc\ k) \longrightarrow gg\ j$
using *24 not-less* **by** *blast*
show *?thesis* **using** *25* **by** *blast*
qed
qed
qed
show *?thesis*
by (*metis 2 assms*(*1*) *less-or-eq-imp-le linorder-neqE-nat*)
qed

lemma *PiUntilDistsem3*:
assumes $(\sigma \models f\ \Pi\ (g\ \mathcal{U}\ h))$
shows $(\sigma \models (\ f\ \Pi\ g\ )\ \mathcal{U}\ (\ f\ \Pi\ h\ )\ )$
proof –
have *1*: $((\exists\, i \leq nlength\ \sigma.\ f\ (ndropn\ i\ \sigma)) \wedge$
$(\exists\, k \leq nlength\ (nfilter\ f\ (ndropns\ \sigma)).$
$h\ (ndropn\ k\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))) \wedge$
$(\forall\, j<k.\ g\ (ndropn\ j\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))))$
using *assms PiUntilDistsem1* **by** *blast*
have *2*: $\exists\ x \in nset(ndropns\ \sigma).\ f\ x$
using *1* **by** (*simp add*: *sfxfilter-help*)
have *3*: $\forall\ x \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma)).\ f\ (nnth\ (ndropns\ \sigma)\ x)$
using *2 nkfilter-holds* **by** *fastforce*
have *4*: $(\exists\, k \leq nlength\ (nfilter\ f\ (ndropns\ \sigma)).$
$h\ (ndropn\ k\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))) \wedge$
$(\forall\, j<k.\ g\ (ndropn\ j\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma))))))$
using *1* **by** *auto*
obtain *k* **where** *5*: $k \leq nlength\ (nfilter\ f\ (ndropns\ \sigma)) \wedge$
$h\ (ndropn\ k\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))) \wedge$
$(\forall\, j<k.\ g\ (ndropn\ j\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))$
using *4* **by** *auto*
have *51*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k) \leq nlength\ \sigma$
by (*metis* (*mono-tags*, *lifting*) *2 5 gen-nlength-def ndropns-nlength nkfilter-nlength nkfilter-upperbound nlength-code*)
have *6*: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k)\ \ \sigma)$
using *2 3 5*
by (*metis 1 ndropns-nfilter-nnth nlength-nmap pifilt-def sfxfilt-def sfxfilt-pifilt-nnth-ndropn*)
have *7*: $k=0 \longrightarrow$
$(\exists\, i.\ (enat\ i) \leq nlength\ \sigma - k \wedge f\ (ndropn\ (i + k)\ \sigma)) \wedge$
$h\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ \ (ndropns\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall\, j<k.\ (\exists\, i \leq nlength\ \sigma - j.\ f\ (ndropn\ (i + j)\ \sigma)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma)))))$
using *1 5* **by** *auto*
have *71*: $k>0 \longrightarrow (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k\ -1)) \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma))$
by (*metis 2 5 One-nat-def Suc-ile-eq Suc-pred in-nset-conv-nnth less-imp-le nkfilter-nlength*)
have *72*: $k>0 \longrightarrow enat\ (k - 1) \leq nlength\ (nkfilter\ f\ 0\ (ndropns\ \sigma))$

by (*metis 2 5 One-nat-def Suc-ile-eq Suc-pred nkfilter-nlength order-less-imp-le*)
have *8*: *k>0 ⟶ f* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k −1*)) *σ*)
using *3 2 71 72 nkfilter-upperbound*[*of* (*ndropns σ*) *f k−1 0*]
by (*metis add-0 ndropns-nlength ndropns-nnth zero-enat-def*)
have *9*: *k>0 ⟶* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k −1*)) < (*nnth* (*nkfilter f 0* (*ndropns σ*)) *k*)
by (*metis 2 5 One-nat-def Suc-pred nkfilter-mono nkfilter-nlength*)
have *10*: *k>0 ⟶* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k −1*)) ≤ *nlength σ*
by (*metis 2 71 gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound nlength-code*)
have *11*: *k>0 ⟶* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *k*) ≤ *nlength σ*
using *nkfilter-upperbound*[*of ndropns σ f k 0*]
using *51* **by** *blast*
have *12*: *k>0 ⟶*
h (*nmap* (*λs. nnth s 0*)
(*nfilter f* (*ndropns* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *k*) *σ*))))
using *nfilter-nkfilter-ndropn*[*of f* (*ndropns σ*) *k*]
ndropns-nfilter-ndropn-a[*of k f σ*]
by (*metis 2 5 ndropn-nmap*)
have *121*: *k>0 ⟶*
h (*nmap* (*λs. nnth s 0*)
(*nfilter f* (*ndropns* (*ndropn* (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*))) *σ*))))
by (*metis 2 5 One-nat-def Suc-pred nfilter-ndropns-nmap*)
have *13*: *k>0 ⟶* (∃ *i≤nlength σ −* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *k*).
f (*ndropn* (*i* + (*nnth* (*nkfilter f 0* (*ndropns σ*)) *k*)) *σ*))
using *6* **by** (*metis add.left-neutral zero-enat-def zero-le*)
have *130*: *k>0 ⟶* (∃ *i. i*+ (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*))) ≤*nlength σ* ∧
(*ndropn* (*i*+(*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*)))) *σ*) =
(*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *k*) *σ*))
using *9* **by** (*metis 11 Suc-leI diff-add*)
have *131*: *k>0 ⟶* (∃ *i. i*+ (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*)))≤*nlength σ* ∧
f (*ndropn* (*i* + (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*)))) *σ*))
using *10 6 9 11 130* **by** *auto*
have *132*: *k>0 ⟶* (∃ *i≤nlength σ −* (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*))).
f (*ndropn* (*i* + (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*)))) *σ*))
using *131*
by (*metis add-diff-cancel-right′ enat-minus-mono1 idiff-enat-enat*)
have *14*: *k>0 ⟶* (∀ *j<* (*nnth* (*nkfilter f 0* (*ndropns σ*)) *k*).
(∃ *i. i+j≤ nlength σ* ∧ *f* (*ndropn* (*i* + *j*) *σ*)))
using *131* **by** (*metis 11 6 diff-add less-imp-le plus-enat-simps*(*1*))
have *141*: *k>0 ⟶* (∀ *j<* (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*))).
(∃ *i. i+j≤nlength σ* ∧ *f* (*ndropn* (*i* + *j*) *σ*)))
using *14 9* **by** *auto*
have *142*: *k>0 ⟶* (∀ *j<* (*Suc* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*k−1*))).
(∃ *i≤nlength σ −j*. *f* (*ndropn* (*i* + *j*) *σ*)))
using *141* **by** (*metis add.commute enat.simps*(*3*) *enat-add-sub-same enat-minus-mono1*)
have *15*: (∀ *j<k. g* (*ndropn j* (*nmap* (*λs. nnth s 0*) (*nfilter f* (*ndropns σ*)))))
using *5* **by** *blast*
have *151*: (∀ *j<k. g* (*nmap* (*λs. nnth s 0*) (*ndropn j* (*nfilter f* (*ndropns σ*)))))
by (*metis 15 ndropn-nmap*)
have *152*: (∀ *j<k.* (*ndropn j* (*nfilter f* (*ndropns σ*))) =

$(nfilter\ f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ (ndropns\ \sigma))))$
using *nfilter-nkfilter-ndropn-1*[*of ndropns σ f*]
by (*simp add*: *2 5 less-imp-le nkfilter-nlength order-less-le-subst2*)
have *16*: $(\forall j<k.\ g\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ (ndropns\ \sigma)))))$
using *151 152 nfilter-nkfilter-ndropn* **by** *simp*
have *1610*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k) \leq nlength(ndropns\ \sigma)$
by (*simp add*: *51 ndropns-nlength*)
have *1611*: $(\forall j<k.\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j) < \ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ k))$
by (*simp add*: *2 5 nidx-nkfilter-gr nkfilter-nlength*)
have *1612*: $(\forall j<k.\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j) \leq nlength(ndropns\ \sigma))$
using *1610 1611* **by** (*meson enat-ord-simps*(*2*) *less-imp-le order-less-le-subst2*)
have *161*: $(\forall j<k.\ (\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ (ndropns\ \sigma))) =$
$(\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ \sigma))))$
using *1612* **by** (*simp add*: *ndropn-ndropns*)
have *17*: $(\forall j<k.\ g\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ j)\ \sigma)))))$
using *16 161* **by** *auto*
have *18*: $k>0 \longrightarrow$
$g\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (k-1))\ \sigma))))$
using *17* **by** *simp*
have *19*: $k>0 \longrightarrow$
$g\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma))))$
using *17* **by** *blast*
have *20*: $k>0 \longrightarrow (nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma))) =$
$(nfilter\ f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ (ndropns\ \sigma)))$
using *161* **by** *auto*
have *200*: $nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0 \leq nlength\ (ndropns\ \sigma)$
using *1610 1612* **by** *blast*
have *21*: $k>0 \longrightarrow (\forall\ \ j\leq (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0).$
$(\ (ndropns\ (ndropn\ j\ \sigma))) =$
$(\ (ndropn\ j\ (ndropns\ \sigma))))$
using *200 ndropn-ndropns* **by** (*simp add*: *ndropn-ndropns order-subst2*)
have *22*: $k>0 \longrightarrow (\forall\ \ j\leq (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma)\ )\ 0).$
$(nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma))) =$
$(nfilter\ f\ (ndropn\ j\ (ndropns\ \sigma))))$
using *21* **by** *auto*
have *23*: $\ k>0 \longrightarrow$
$(\forall\ \ j\leq (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma)\ )\ 0).$
$(nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma)))) =$
$(nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma))))$
)
by (*simp add*: *2 22 nfilter-ndropns-nmap-help-0*)
have *24*: $k>0 \longrightarrow$
$(\forall\ \ j\leq (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma)\ )\ 0).$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)\ \sigma)))) =$

g (*nmap* (*λs. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn j σ*))))
)
using *23* **by** *auto*
have *241*: *k>0* ⟶ (∀ *j*≤ (*nnth* (*nkfilter f 0* (*ndropns σ*)) *0*).
g (*nmap* (*λs. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn j σ*)))))
using *19 24* **by** *blast*
have *25*: *k>0* ⟶ (∀ *i* < *k* −*1*.
(∀ *l*. *l*≤ (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*Suc i*)) ∧
(*nnth* (*nkfilter f 0* (*ndropns σ*)) *i*) < *l* ⟶
(*nfilter f* (*ndropn* (*nnth*(*nkfilter f 0* (*ndropns σ*)) (*Suc i*)) (*ndropns σ*)) =
(*nfilter f* (*ndropn l* (*ndropns σ*))))))
proof *auto*
fix *i*
fix *l*
assume *a0*: *0* < *k*
assume *a1*: *i* < *k* − *Suc 0*
assume *a2*: *l* ≤ *nnth* (*nkfilter f 0* (*ndropns σ*)) (*Suc i*)
assume *a3*: *nnth* (*nkfilter f 0* (*ndropns σ*)) *i* < *l*
show *nfilter f* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*Suc i*)) (*ndropns σ*)) =
nfilter f (*ndropn l* (*ndropns σ*))
proof −
have *251*: *k=1* ⟹
nfilter f (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*Suc i*)) (*ndropns σ*)) =
nfilter f (*ndropn l* (*ndropns σ*))
using *a1* **by** *force*
have *2511*: *1* < *k* ⟹ *enat* (*Suc i*) ≤ *nlength* (*nfilter f* (*ndropns σ*))
by (*metis* (*mono-tags*, *opaque-lifting*) *5 One-nat-def Suc-diff-1 Suc-mono a0 a1*
enat-ord-simps(*1*) *less-imp-le order-subst2*)
then have *252*: *1*<*k* ⟹
nfilter f (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*Suc i*)) (*ndropns σ*)) =
nfilter f (*ndropn l* (*ndropns σ*))
using *2 5 a1 nfilter-ndropns-nmap-help-j*[*of ndropns σ f l i*] *a2 a3* **by** *fastforce*
show *?thesis*
using *252 a1* **by** *fastforce*
qed
qed
have *261*: *k>0* ⟶ (∀ *i* < *k* −*1*.
(∀ *l*. *l*≤ (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*Suc i*)) ∧
(*nnth* (*nkfilter f 0* (*ndropns σ*)) *i*) < *l* ⟶
l ≤ *nlength* (*ndropns σ*)))
using *1612* **by** (*meson diff-less enat-ord-simps*(*1*) *less-one less-trans-Suc order-subst2*)
have *262*: *k>0* ⟶ (∀ *i* < *k* −*1*.
(∀ *l*. *l*≤ (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*Suc i*)) ∧
(*nnth* (*nkfilter f 0* (*ndropns σ*)) *i*) < *l* ⟶
(*ndropn l* (*ndropns σ*)) = (*ndropns* (*ndropn l σ*))))
using *261* **by** (*simp add*: *ndropn-ndropns*)
have *26*: *k>0* ⟶ (∀ *i* < *k* −*1*.
(∀ *l*. *l*≤ (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*Suc i*)) ∧
(*nnth* (*nkfilter f 0* (*ndropns σ*)) *i*) < *l* ⟶
(*nmap* (*λs. nnth s 0*)

(*nfilter f* (*ndropn* (*nnth*(*nkfilter f 0* (*ndropns* σ)) (*Suc i*)) (*ndropns* σ))))=
(*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn l* σ))))))
using *25 262* **by** *auto*
have *27*: *k*>*0* $\longrightarrow$ ($\forall$ *i* < *k* −*1*.
($\forall$ *l*. *l*$\leq$ (*nnth* (*nkfilter f 0* (*ndropns* σ)) (*Suc i*)) $\wedge$
(*nnth* (*nkfilter f 0* (*ndropns* σ)) *i*) < *l* $\longrightarrow$
g (*nmap* (λ*s. nnth s 0*)
(*nfilter f*
(*ndropn* (*nnth*(*nkfilter f 0* (*ndropns* σ)) (*Suc i*)) (*ndropns* σ))))))
by (*simp add*: *16*)
have *28*: *k*>*0* $\longrightarrow$ ($\forall$ *i* < *k* −*1*.
($\forall$ *l*. *l*$\leq$ (*nnth* (*nkfilter f 0* (*ndropns* σ)) (*Suc i*)) $\wedge$
(*nnth* (*nkfilter f 0* (*ndropns* σ)) *i*) < *l* $\longrightarrow$
g (*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn l* σ))))))
using *25 262 27* **by** *auto*
have *281*: *k*>*0* $\longrightarrow$
($\forall$*j*.
(*j* $\leq$ (*nnth* (*nkfilter f 0* (*ndropns* σ)) *0*) $\vee$
($\exists$ *i*. *i* < *k* −*1* $\wedge$
j$\leq$ (*nnth* (*nkfilter f 0* (*ndropns* σ)) (*Suc i*)) $\wedge$
(*nnth* (*nkfilter f 0* (*ndropns* σ)) *i*) < *j*))
$\longrightarrow$ *g* (*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn j* σ)))))
using *241 28* **by** *blast*
have *282*: *k*>*0* $\longrightarrow$ ($\forall$*i*<*k* − *1*.
nnth (*nkfilter f 0* (*ndropns* σ)) *i* < *nnth* (*nkfilter f 0* (*ndropns* σ)) (*Suc i*))
using *nidx-nkfilter-expand*[*of ndropns* σ *f - 0*]
by (*metis* (*full-types*) *2 72 Suc-ile-eq enat-ord-simps*(*2*) *order-less-le-subst2*)
have *29*: *k*>*0* $\longrightarrow$ ($\forall$*j*< (*Suc* (*nnth* (*nkfilter f 0* (*ndropns* σ)) (*k*−*1*))).
g (*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn j* σ)))))
using *281 282 cover-a*[*of* λ *x*. *nnth* (*nkfilter f 0* (*ndropns* σ)) *x k*−*1*
(λ *j*. *g* (*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn j* σ)))))]
using *18 less-antisym* **by** *blast*
have *30*: *k*>*0* $\longrightarrow$ ($\exists$*k*$\leq$*nlength* σ.
($\exists$*i*$\leq$*nlength* σ − *k*. *f* (*ndropn* (*i* + *k*) σ)) $\wedge$
h (*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn k* σ)))) $\wedge$
($\forall$*j*<*k*. ($\exists$*i*$\leq$*nlength* σ − *j*. *f* (*ndropn* (*i* + *j*) σ)) $\wedge$
g (*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn j* σ))))))
using *10 11 121 132 142 29 9* **by** *simp*
(*metis add-Suc-right antisym-conv2 eSuc-enat enat-ord-simps*(*1*) *ileI1 leD*)
have *31*: ($\exists$*k*$\leq$*nlength* σ.
($\exists$*i*$\leq$*nlength* σ − *k*. *f* (*ndropn* (*i* + *k*) σ)) $\wedge$
h (*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn k* σ)))) $\wedge$
($\forall$*j*<*k*. ($\exists$*i*$\leq$*nlength* σ − *j*. *f* (*ndropn* (*i* + *j*) σ)) $\wedge$
g (*nmap* (λ*s. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn j* σ))))))
using *30 7* **by** (*metis not-gr-zero zero-enat-def zero-le*)
show *?thesis* **using** *31* **by** (*simp add*: *PiUntilDistsem2*)
qed

lemma *PiUntilDistsem4*:
assumes ($\sigma \models$ (*f* Π *g*) $\mathcal{U}$ (*f* Π *h*))

shows $(\sigma \models f\ \Pi\ (g\ \mathcal{U}\ h))$
proof –
have *1*: $(\exists k \le nlength\ \sigma.$
$(\exists i \le nlength\ \sigma - k.\ f\ (ndropn\ (i + k)\ \sigma)) \wedge$
$h\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall j<k.$
$(\exists i \le nlength\ \sigma - j.\ f\ (ndropn\ (i + j)\ \sigma)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma))))))$
using *assms* **by** (*simp add*: *PiUntilDistsem2*)
obtain k **where** *2*: $k \le nlength\ \sigma\ \wedge$
$(\exists i \le nlength\ \sigma - k.\ f\ (ndropn\ (i + k)\ \sigma)) \wedge$
$h\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ k\ \sigma)))) \wedge$
$(\forall j<k.$
$(\exists i \le nlength\ \sigma - j.\ f\ (ndropn\ (i + j)\ \sigma)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ (ndropn\ j\ \sigma)))))$
using *1* **by** *auto*
have *3*: $(\exists i \le nlength\ \sigma - k.\ f\ (ndropn\ (i + k)\ \sigma))$
using *2* **by** *auto*
have *31*: $(\exists i \le\ nlength\ (ndropns\ (ndropn\ k\ \sigma)).\ f\ (nnth\ (ndropns\ (ndropn\ k\ \sigma))\ i))$
by (*metis 3 add.commute ndropn-ndropn ndropn-nlength ndropns-nlength ndropns-nnth*)
have *4*: $k \le nlength\ \sigma$
using *2* **by** *auto*
obtain i **where** *5*: $(enat\ i) \le nlength\ (ndropns\ (ndropn\ k\ \sigma)) \wedge f\ (nnth\ (ndropns\ (ndropn\ k\ \sigma))\ i) \wedge$
$i = (LEAST\ na.\ enat\ na \le nlength\ (ndropns\ (ndropn\ k\ \sigma)) \wedge$
$f\ (nnth\ (ndropns\ (ndropn\ k\ \sigma))\ na))$
by (*metis* (*no-types*, *lifting*) *31 LeastI-ex*)
have *51*: $i = (LEAST\ na.\ enat\ na \le nlength\ (ndropns\ (ndropn\ k\ \sigma)) \wedge f\ (nnth\ (ndropns\ (ndropn\ k\ \sigma))$
$na))$
using *5* **by** *auto*
have *60*: $(enat\ i) \le\ nlength\ \sigma - k$
by (*metis 5 ndropn-nlength ndropns-nlength*)
have *601*: $k+i \le nlength\ \sigma$
by (*metis 4 60 dual-order.eq-iff enat.simps*(*3*) *enat-add-sub-same enat-less-enat-plusI2 less-eqE less-imp-le order.not-eq-order-implies-strict plus-enat-simps*(*1*))
have *6*: $i+k \le nlength\ \sigma$
by (*metis 601 add.commute*)
have *61*: $\exists x \in nset\ (ndropns\ (ndropn\ k\ \sigma)).\ f\ x$
using *31 exists-Pred-nnth-nset* **by** *blast*
have *62*: $\exists x \in nset\ (ndropns\ (ndropn\ (k+i)\ \sigma)).\ f\ x$
by (*metis 5 in-nset-conv-nnth ndropn-ndropn ndropn-ndropns nnth-zero-ndropn zero-enat-def zero-le*)
have *7*: $(\exists i \le nlength\ \sigma.\ f\ (ndropn\ i\ \sigma))$
by (*metis 5 6 add.commute ndropn-ndropn ndropns-nlength ndropns-nnth*)
have *71*: $\exists x \in nset\ (ndropns\ \sigma).\ f\ x$
using *5 6* **using** *in-nset-ndropns* **using** *7* **by** *blast*
have *72*: $\exists\ x \in nset(nkfilter\ f\ 0\ (ndropns\ \sigma)).\ f\ (nnth\ (ndropns\ \sigma)\ x)$
using *71 nkfilter-holds-b*[*of* $(ndropns\ \sigma)$ *f*] *sfxfilter-help*[*of* σ]
by (*metis 71 Nat.add-0-right ndropns-nlength ndropns-nnth*)
have *722*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0) \le nlength\ (ndropns\ (ndropn\ k\ \sigma))$
by (*metis 61 gen-nlength-def nkfilter-upperbound nlength-code zero-enat-def zero-le*)
have *723*: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0) \le nlength\ (\ (ndropn\ k\ \sigma))$

by (*metis 722 ndropns-nlength*)
have *73*: *f* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) (*ndropn k σ*))
using *nkfilter-holds*[*of ndropns* (*ndropn k σ*) *f 0*]
by (*metis* (*no-types*, *lifting*) *61 722 diff-zero ndropns-nfilter-nnth ndropns-nlength ndropns-nnth nkfilter-nfilter zero-enat-def zero-le*)
have *74*: *f* (*ndropn* ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) *σ*)
using *73* **by** (*simp add*: *add.commute ndropn-ndropn*)
have *75*: *f* (*ndropn k* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) *σ*))
by (*simp add*: *74 ndropn-ndropn*)
have *76*: (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) = *nleast f* (*ndropns* (*ndropn k σ*))
by (*simp add*: *61 nkfilter-nleast*)
have *77*: *nleast f* (*ndropns* (*ndropn k σ*)) = (*LEAST na. enat na* ≤ *nlength* (*ndropns* (*ndropn k σ*)) ∧ *f* (*nnth* (*ndropns* (*ndropn k σ*)) *na*))
using *61 nleast-conv*[*of ndropns* (*ndropn k σ*) *f*] **by** *auto*
have *78*: *nleast f* (*ndropns* (*ndropn k σ*)) = *i*
using *5 77* **by** *blast*
have *8*: *h* (*nmap* (*λs. nnth s 0*) (*nfilter f* (*ndropns* (*ndropn k σ*))))
using *2* **by** *auto*
have *10*: *nfirst* (*nkfilter f k* (*ndropn k* (*ndropns σ*))) = ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*)
by (*metis 2 61 diff-add ndropn-0 ndropn-ndropns ndropn-nfirst ndropns-nlength nkfilter-lowerbound nkfilter-nnth-n-zero zero-enat-def zero-le*)
have *101*: *nfirst* (*nkfilter f k* (*ndropn k* (*ndropns σ*))) = *k*+ *i*
by (*simp add*: *10 76 78*)
have *90*: *f* (*nlast* (*ntaken* ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) (*ndropns σ*)))
by (*metis 6 74 76 78 ndropns-nnth ntaken-nlast*)
have *91*: ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) ≤ *nlength* (*ndropns σ*)
using *nkfilter-upperbound*[*of* (*ndropns* (*ndropn k σ*)) *f 0 0*] *ndropn-nlength*[*of k σ*] *ndropns-nlength*[*of ndropn k σ*]
by (*simp add*: *6 76 78 ndropns-nlength*)
have *92*: ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) = ((*nnth* (*nkfilter f k* (*ndropns* (*ndropn k σ*))) *0*))
by (*simp add*: *61 nkfilter-nleast*)
let *?kf* = (*the-enat* ((*nlength*(*nfilter f* (*ntaken* ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) (*ndropns σ*))))))
let *?nkf* = (*the-enat* ((*nlength*(*nkfilter f 0* (*ntaken* ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) (*ndropns σ*))))))
let *?pkf* = (*the-enat* ((*nlength* (*nkfilter f 0* (*ntaken k* (*ndropns σ*))))))
have *93*: *?kf* = *?nkf*
by (*metis 90 nfinite-ntaken nkfilter-nlength nset-nlast*)
have *94*: ((*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)+*k*) ≤ *nnth* (*nkfilter f 0* (*ndropns σ*)) *?nkf*
by (*metis 74 90 91 add.left-neutral eq-iff ndropn-nfirst ndropns-nlength ndropns-nnth nkfilter-chop1-ndropn nkfilter-nfirst*)
have *95*: (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) = *0* ⟹ *k* = *nnth* (*nkfilter f 0* (*ndropns σ*)) *?nkf*
by (*metis 10 90 91 add-cancel-left-left ndropn-nfirst nkfilter-chop1-ndropn*)
have *98*: *0*< *?nkf* ∧ *0*< (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*) ⟹ *nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*) < *k*
proof −
assume *b*: *0*< *?nkf* ∧ *0*< (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k σ*))) *0*)
have *980*: (*nnth* (*nkfilter f 0* (*ndropns σ*)) (*?nkf* −*1*)) ∈ *nset* (*nkfilter f 0* (*ndropns σ*))

by (*metis in-nset-conv-nnth le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have *981*: *f* (*nnth* (*ndropns* σ) (*nnth* (*nkfilter f 0* (*ndropns* σ)) (*?nkf* $-$*1*)))
using *nkfilter-holds*[*of* (*ndropns* σ) *f* (*nnth* (*nkfilter f 0* (*ndropns* σ)) (*?nkf* $-$*1*)) *0*]
using *71 980* **by** *auto*
have *982*: $\neg$ *f* (*ndropn k* σ)
by (*metis 10 4 add-cancel-left-left b gr-implies-not0 ndropn-nfirst ndropns-nnth nkfilter-nfirst*)
have *984*: *f* (*ndropn* (*nnth* (*nkfilter f 0* (*ndropns* σ)) (*?nkf* $-$*1*)) σ)
using *71 980 981 ndropns-nnth*[*of* (*nnth* (*nkfilter f 0* (*ndropns* σ)) (*?nkf* $-$*1*)) σ]
by (*metis gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound nlength-code*)
have *985*: $\bigwedge$*j. k* $\leq$*j* $\Longrightarrow$ *j*< *k*+*i* $\Longrightarrow$ $\neg$ *f* (*ndropn j* σ)
proof –
fix *j*
assume *a0*: *k*$\leq$*j*
assume *a1*: *j*<*k*+*i*
show $\neg$ *f* (*ndropn j* σ)
proof –
have *9851*: $\exists$ *x*$\in$*nset* (*ndropn k* (*ndropns* σ)). *f x*
by (*simp add*: *4 61 ndropn-ndropns ndropns-nlength*)
have *9852*: *enat k* $\leq$ *nlength* (*ndropns* σ)
by (*simp add*: *4 ndropns-nlength*)
have *9853*: *k* $\leq$ *j*
by (*simp add*: *a0*)
have *9854*: *j* < *nnth* (*nkfilter f k* (*ndropn k* (*ndropns* σ))) *0*
using *10 101 92 9852 a1 ndropn-ndropns* **by** *fastforce*
have *9855*: $\neg$ *f* (*nnth* (*ndropns* σ) *j*)
using *9851 9852 9853 9854 nkfilter-n-not-before*[*of k ndropns* σ *f j*] **by** *auto*
show *?thesis*
by (*metis 10 101 6 76 78 9855 a1 enat-ord-simps*(*2*) *less-imp-le ndropns-nnth order-less-le-subst2*)
qed
qed
have *986*: *ntaken ?nkf* (*nkfilter f 0* (*ndropns* σ)) =
nkfilter f 0 (*ntaken* (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k* σ))) *0* + *k*) (*ndropns* σ))
using *nkfilter-chop1-ntaken*[*of* (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k* σ))) *0* + *k*) (*ndropns* σ) *f 0*]
using *90 91* **by** *blast*
have *987*: $\neg$ *enat* (*?nkf*) $\leq$ *nlength* (*nkfilter f 0* (*ndropns* σ)) $\vee$
nnth (*nkfilter f 0* (*ndropns* σ)) (*?nkf* $-$ *1*) < *nnth* (*nkfilter f 0* (*ndropns* σ)) (*?nkf*)
using *71* **by** (*meson b diff-less less-one nidx-nkfilter-gr*)
have *988*: *nlast* (*nkfilter f 0* (*ntaken* (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k* σ))) *0* + *k*) (*ndropns* σ)))
=
0 + (*nnth* (*nkfilter f 0* (*ndropns* (*ndropn k* σ))) *0* + *k*)
by (*simp add*: *90 91 nkfilter-nlast*)
have *989*: *nnth* (*nkfilter f 0* (*ndropns* σ)) (*?nkf*) = *nnth* (*nkfilter f 0* (*ndropns* (*ndropn k* σ))) *0* + *k*
using *986*
by (*metis 988 add.left-neutral ntaken-nlast*)
have *990*: *nnth* (*nkfilter f 0* (*ndropns* σ)) (*?nkf*) = *nfirst* (*nkfilter f k* (*ndropn k* (*ndropns* σ)))
using *10 989* **by** *fastforce*
have *991*: *nnth* (*nkfilter f 0* (*ndropns* σ)) (*?nkf*) = *k* + *i*
using *10 101 989* **by** *presburger*

```
    show ?thesis
      by (metis 984 985 986 987 991 enat-the-enat infinity-ileE le-cases not-le-imp-less ntaken-all)
  qed
have 100:  h (nmap (λs. nnth s 0) (ndropn ?kf (nfilter f (ndropns σ))))
  using ndropns-nfilter-ndropn-a[of - f ndropn k σ ]
      nfilter-chop1-ndropn[of ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ) f  ]
    101 61 76 78 8 90 91
  by simp
    (metis 10 ndropn-0 ndropn-ndropn ndropn-ndropns zero-enat-def zero-le)
have 11: h (ndropn ?kf (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))
  by (simp add: 100 ndropn-nmap)
have 111: ?kf ≤ nlength (nfilter f (ndropns σ))
  by (metis 90 enat-ile le-cases nfilter-chop1-ntaken ntaken-all the-enat.simps)
have 112: nfinite (nfilter f (ntaken ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ)))
  by (metis 90 91 nfilter-chop1-ntaken nfinite-ntaken)
have 13: (∀ j<k.
          (∃ x ∈ nset(ndropns (ndropn j σ)). f x) ∧
          g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
  by (metis 2 add.commute in-nset-ndropns ndropn-ndropn ndropn-nlength)
have 151: (∀ jj<?kf.
           (ndropn jj (nfilter f (ndropns σ)))  =
           nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) jj) σ))
          )   using nfilter-nkfilter-ndropn
  by (simp add: 111 71 less-imp-le ndropns-nfilter-ndropn-a order-less-le-subst2)
have 152: (⋀ jj. (enat jj)<?kf ⟹ (nnth (nkfilter f 0 (ndropns σ)) jj) < k )
  proof −
   fix jj
   assume a: (enat jj) < ?kf
   show nnth (nkfilter f 0 (ndropns σ)) jj < k
    proof −
     have 1522: (enat jj) ≤ ?nkf −1
       using 93 a by auto
     have 1523: nnth (nkfilter f 0 (ndropns σ)) (?nkf −1) < nnth (nkfilter f 0 (ndropns σ)) ?nkf
       by (metis 111 71 93 a diff-less gr-implies-not-zero less-numeral-extra(1)
           nidx-nkfilter-gr nkfilter-nlength not-gr-zero zero-enat-def)
     have 15230: enat (?nkf − 1) ≤ nlength (nkfilter f 0 (ndropns σ))
       by (metis 111 71 93 One-nat-def Suc-ile-eq Suc-pred a less-imp-le nkfilter-nlength
            not-gr-zero not-less-zero zero-enat-def)
     have 1524: nnth (nkfilter f 0 (ndropns σ)) jj ≤ nnth (nkfilter f 0 (ndropns σ)) (?nkf−1)
       using 1522 1523 71 93 15230
         nidx-less-eq[of (nkfilter f 0 (ndropns σ)) (jj) ?nkf −1]
         nidx-nkfilter[of (ndropns σ) f 0]
       using enat-ord-simps(1) by blast
     have 1525:  k ≤  nnth (nkfilter f 0 (ndropns σ)) ?nkf
       using 94 add-leE by blast
     have 1526: (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) = 0 ⟹
                nnth (nkfilter f 0 (ndropns σ)) jj < k
       using 1523 1524 95 by linarith
     have 1527: 0<(nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) ⟹
               nnth (nkfilter f 0 (ndropns σ)) (?nkf −1) < k
```

using *93 98 a* **by** *auto*
show *?thesis* **using** *1524 1526 1527 dual-order.strict-trans2* **by** *blast*
qed
qed
have *153*: $(\forall jj < ?kf.$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)$
$(nfilter\ f\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ jj)\ \sigma))\ )$
$))$
using *112 13 152 nfinite-nlength-enat* **by** *force*
have *1530*: $(\forall jj < ?kf.$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (ndropn\ jj\ \ (nfilter\ f\ (ndropns\ \sigma)))))$
by (*simp add*: *151 153*)
have *1531*: $(\forall jj < ?kf.$
$g\ \ (ndropn\ jj\ \ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))$
by (*simp add*: *1530 ndropn-nmap*)
have *154*: $(\ (\exists i.\ (enat\ i) \le nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma)) \wedge$
$(\exists kk.\ (enat\ kk) \le nlength\ (nfilter\ f\ (ndropns\ \sigma)) \wedge$
$h\ (ndropn\ kk\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))) \wedge$
$(\forall jj < kk.\ g\ (ndropn\ jj\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ \sigma)))))))$
using *11 111 1531 7* **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*

13.3.15 PiChopstar

lemma *wnextboxnotstatesem*:
assumes $k \le nlength\ \sigma$
shows $(\forall\ \ j \le nlength\ \sigma.\ k < j \longrightarrow \neg\ (\lambda y.\ w\ (NNil\ y))\ (nnth\ \sigma\ j)) =$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma)$

using *assms*
proof (*auto simp add*: *itl-defs ndropn-nfirst*)
show $\bigwedge n.\ enat\ k \le nlength\ \sigma \Longrightarrow$
$\forall j.\ enat\ j \le nlength\ \sigma \longrightarrow k < j \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ j)) \Longrightarrow$
$nlength\ \sigma - enat\ k \ne enat\ 0 \Longrightarrow$
$enat\ n \le nlength\ \sigma - enat\ k - enat\ (Suc\ 0) \Longrightarrow w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + n)))) \Longrightarrow False$
by (*metis add-Suc-right assms diff-self-eq-0 dual-order.order-iff-strict*
enat-ord-simps(*2*) *idiff-enat-enat less-add-same-cancel1 linorder-le-cases*
nfinite-nlength-enat nfinite-ntaken not-le-imp-less ntaken-all ntaken-nlast
zero-enat-def zero-less-Suc)
show $\bigwedge j.\ enat\ k \le nlength\ \sigma \Longrightarrow enat\ j \le nlength\ \sigma \Longrightarrow k < j \Longrightarrow w\ (NNil\ (nnth\ \sigma\ j)) \Longrightarrow$
$nlength\ \sigma - enat\ k = enat\ 0 \Longrightarrow False$

by (*metis add.right-neutral enat.inject enat-add-sub-same enat-ord-code*(*4*) *leD less-eqE min.strict-order-iff min-enat-simps*(*1*) *zero-enat-def*)
show $\bigwedge j.\ enat\ k \le nlength\ \sigma \Longrightarrow$
$enat\ j \le nlength\ \sigma \Longrightarrow$
$k < j \Longrightarrow w\ (NNil\ (nnth\ \sigma\ j)) \Longrightarrow$
$\forall n.\ enat\ n \le nlength\ \sigma - enat\ k - enat\ (Suc\ 0) \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + n)))) \Longrightarrow False$
proof −
fix *j*
assume *a0*: $enat\ k \le nlength\ \sigma$
assume *a1*: $enat\ j \le nlength\ \sigma$
assume *a2*: $k < j$
assume *a3*: $w\ (NNil\ (nnth\ \sigma\ j))$
assume *a4*: $\forall n.\ enat\ n \le nlength\ \sigma - enat\ k - enat\ (Suc\ 0) \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + n))))$
show *False*
proof −
have *1*: $j = (Suc\ (k + (j - (Suc\ k))))$
using *a2* **by** *auto*
have *2*: $\forall n.\ enat\ n + 1 + k \le nlength\ \sigma \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + n))))$
by *auto*
(*metis One-nat-def a4 add.commute enat.simps*(*3*) *enat-add-sub-same enat-minus-mono1 one-enat-def*)
have *3*: $(j - (Suc\ k)) + 1 + k \le nlength\ \sigma$
using *1 a1* **by** *simp*
have *4*: $\neg\ w\ (NNil\ (nnth\ \sigma\ (Suc\ (k + (j - (Suc\ k))))))$
using *2 3 eSuc-enat plus-1-eSuc*(*2*) **by** *auto*
from *1 4 a3* **show** *?thesis* **by** *auto*
qed
qed
qed

lemma *NotStateUntilStateAndsem*:
$(\sigma \models (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge f)\ ) =$
$(\exists k.\ (enat\ k) \le nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge f\ (ndropn\ k\ \sigma) \wedge (\forall j<k.\ \neg\ w\ (NNil\ (nnth\ \sigma\ j))))$

by (*auto simp add*: *until-d-def init-defs ndropn-nfirst*)

lemma *StateUntilEqvWPrevChopsem*:
$\sigma \models (init\ w)\ \mathcal{U}\ f = (wprev\ (\Box\ (init\ w)))\frown f$
proof (*auto simp add*: *until-d-def itl-defs ntaken-nnth ndropn-nfirst min-absorb1*)
show $\bigwedge k.\ enat\ k \le nlength\ \sigma \Longrightarrow$
$f\ (ndropn\ k\ \sigma) \Longrightarrow$
$\forall j<k.\ w\ (NNil\ (nnth\ \sigma\ j)) \Longrightarrow$
$\exists n.\ enat\ n \le nlength\ \sigma \wedge$
$(min\ (enat\ n)\ (nlength\ \sigma) = enat\ 0 \vee$
$(\forall na.\ na \le the\text{-}enat\ (min\ (enat\ (n - Suc\ 0))\ (epred\ (nlength\ \sigma))) \wedge na \le n \wedge enat\ na \le nlength\ \sigma \longrightarrow$
$w\ (NNil\ (nnth\ \sigma\ na)))) \wedge$
$f\ (ndropn\ n\ \sigma)$
by (*metis One-nat-def Suc-pred epred-enat epred-min le-imp-less-Suc min.orderE not-gr-zero the-enat.simps*)

next
  fix n
  assume $a0$: $enat\ n \leq nlength\ \sigma$
  assume $a1$: $f\ (ndropn\ n\ \sigma)$
  assume $a2$: $\forall na.\ na \leq the\text{-}enat\ (min\ (enat\ (n - Suc\ 0))\ (epred\ (nlength\ \sigma))) \wedge na \leq n \wedge$
  $enat\ na \leq nlength\ \sigma \longrightarrow w\ (NNil\ (nnth\ \sigma\ na))$
  show $\exists k.\ enat\ k \leq nlength\ \sigma \wedge f\ (ndropn\ k\ \sigma) \wedge (\forall j<k.\ w\ (NNil\ (nnth\ \sigma\ j)))$
  proof –
  have 1: $enat\ n \leq nlength\ \sigma \wedge f\ (ndropn\ n\ \sigma)$
   using $a0\ a1$ **by** $auto$
  have 2: $(\forall na.\ na \leq the\text{-}enat\ (min\ (enat\ (n - Suc\ 0))\ (epred\ (nlength\ \sigma))) \wedge na \leq n \wedge$
  $enat\ na \leq nlength\ \sigma \longrightarrow w\ (NNil\ (nnth\ \sigma\ na))) \longrightarrow$
  $(\forall j<n.\ w\ (NNil\ (nnth\ \sigma\ j)))$
   by ($auto\ simp\ add$: $min\text{-}def$)
   ($simp\ add$: $a0\ less\text{-}imp\text{-}le\ order\text{-}less\text{-}le\text{-}subst2$,
   $metis\ One\text{-}nat\text{-}def\ a0\ epred\text{-}enat\ epred\text{-}min\ min.absorb\text{-}iff2$)
  have 3: $(\forall j<n.\ w\ (NNil\ (nnth\ \sigma\ j)))$
   using $2\ a2$ **by** $blast$
  show $?thesis$
  using $1\ 3$ **by** $blast$
 qed
qed

lemma $StateUntilEqvWPrevChop$:
$\vdash (init\ w)\ \mathcal{U}\ f = (wprev\ (\Box\ (init\ w)))\frown f$
using $StateUntilEqvWPrevChopsem\ Valid\text{-}def$ **by** $blast$

lemma $UntilChopDist$:
$\vdash (init\ w)\ \mathcal{U}\ (g\frown h) = (\ (init\ w)\ \mathcal{U}\ g)\ \frown h$
using $StateUntilEqvWPrevChop[of\ w\ ]$
by ($metis\ SChopAssoc\ inteq\text{-}reflection$)

lemma $PiEmptysem$:
$\sigma \models (init\ w)\ \Pi\ empty = (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w))))$
proof –
have 1: $(\sigma \models (init\ w)\ \Pi\ empty) =$
 $((\exists x\in nset\ \sigma.\ w\ (NNil\ x)) \wedge nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) = 0)$
 by ($simp\ add$: $Pistate\ itl\text{-}defs\ zero\text{-}enat\text{-}def$)
have 2: $((\exists x\in nset\ \sigma.\ w\ (NNil\ x)) \wedge nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) = 0) =$
 $(\exists\ k \leq nlength\ \sigma\ .\ (\lambda y.\ w\ (NNil\ y))\ (nnth\ \sigma\ k) \wedge$
 $(\forall\ j.\ j<k \longrightarrow \neg\ (\lambda y.\ w\ (NNil\ y))\ (nnth\ \sigma\ j)) \wedge$
 $(\forall\ j \leq nlength\ \sigma.\ k<j \longrightarrow \neg\ (\lambda y.\ w\ (NNil\ y))\ (nnth\ \sigma\ j)))$
 by ($simp\ add$: $nfilter\text{-}nlength\text{-}zero\text{-}conv\text{-}2$)
have 3: $(\exists\ k \leq nlength\ \sigma\ .\ (\lambda y.\ w\ (NNil\ y))\ (nnth\ \sigma\ k) \wedge$
 $(\forall\ j.\ j<k \longrightarrow \neg\ (\lambda y.\ w\ (NNil\ y))\ (nnth\ \sigma\ j)) \wedge$
 $(\forall\ j \leq nlength\ \sigma.\ k<j \longrightarrow \neg\ (\lambda y.\ w\ (NNil\ y))\ (nnth\ \sigma\ j))) =$
 $(\exists k\leq nlength\ \sigma.$
 $w\ (NNil\ (nnth\ \sigma\ k)) \wedge$
 $(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma) \wedge$
 $(\forall j<k.\ \neg\ w\ (NNil\ (nnth\ \sigma\ j))))$ (**is** $?lhs = ?rhs$)

proof
assume *a*: *?lhs*
show *?rhs* **using** *a wnextboxnotstatesem* **by** *auto*
next
assume *b*: *?rhs*
show *?lhs*
proof −
obtain *k* **where** *1*:
$enat\ k \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma) \wedge (\forall j<k.\ \neg\ w\ (NNil\ (nnth\ \sigma\ j)))$
using *b* **by** *auto*
have *2*: $enat\ k \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge (\forall j<k.\ \neg\ w\ (NNil\ (nnth\ \sigma\ j))) \wedge$
$(\forall j.\ enat\ j \leq nlength\ \sigma \longrightarrow k < j \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ j)))$
using *1* *wnextboxnotstatesem*[*of k σ w*]
proof −
have $\forall x0.\ (x0 < k \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ x0))) = (\neg\ x0 < k \vee \neg\ w\ (NNil\ (nnth\ \sigma\ x0)))$
by *meson*
then have *f1*: $enat\ k \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma) \wedge (\forall n.\ \neg\ n < k \vee \neg\ w\ (NNil\ (nnth\ \sigma\ n)))$
by (*metis* ‹$enat\ k \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma) \wedge (\forall j<k.\ \neg\ w\ (NNil\ (nnth\ \sigma\ j)))$›)
have $(enat\ k \leq nlength\ \sigma \longrightarrow (\forall n.\ enat\ n \leq nlength\ \sigma \wedge k < n \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ n))) =$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma)) =$
$(\neg\ enat\ k \leq nlength\ \sigma \vee (\forall n.\ (\neg\ enat\ n \leq nlength\ \sigma \vee \neg\ k < n) \vee \neg\ w\ (NNil\ (nnth\ \sigma\ n))) =$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma))$
by *blast*
then have $\neg\ enat\ k \leq nlength\ \sigma \vee (\forall n.\ (\neg\ enat\ n \leq nlength\ \sigma \vee \neg\ k < n) \vee \neg\ w\ (NNil\ (nnth\ \sigma$
$n))) =$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma)$
by (*metis* ‹$enat\ k \leq nlength\ \sigma \Longrightarrow (\forall j.\ enat\ j \leq nlength\ \sigma \longrightarrow k < j \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ j)))$
$=$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma)$›)
then show *?thesis* **using** *f1* **by** *presburger*
qed
show *?thesis* **using** *2* **by** *blast*
qed
qed
have *4*: $(\exists k \leq nlength\ \sigma.$
$w\ (NNil\ (nnth\ \sigma\ k)) \wedge$
$(LIFT(wnext\ (\Box\ (init\ (\neg\ w)))))\ (ndropn\ k\ \sigma) \wedge$
$(\forall j<k.\ \neg\ w\ (NNil\ (nnth\ \sigma\ j)))) =$
$(\sigma \models (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w)))))$
by (*simp add*: *NotStateUntilStateAndsem*)
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *PiEmpty*:
$\vdash (init\ w)\ \Pi\ empty = (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w))))$
using *PiEmptysem Valid-def* **by** *blast*

lemma *StatePiBoxStatesem*:
$\sigma \models (init\ w)\ \Pi\ f = (init\ w)\ \Pi\ (f \wedge \Box\ (init\ w))$
proof –
have *1*: $(\sigma \models (init\ w)\ \Pi\ f) =$
$((\exists\ x \in nset\ \sigma.\ w\ (NNil\ x)) \wedge\ (\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \models\ f))$
by (*metis Pistate*)
have *2*: $(\sigma \models (init\ w)\ \Pi\ (f \wedge \Box\ (init\ w))) =$
$((\exists\ x \in nset\ \sigma.\ w\ (NNil\ x)) \wedge\ (\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \models f \wedge \Box\ (init\ w)))$
by (*metis Pistate*)
have *3*: $(\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \models f \wedge \Box\ (init\ w))$
$= (f\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \wedge$
$(\forall\ n \le nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma).\ w\ (NNil\ (nnth\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)\ n))))$
by (*simp add*: *itl-defs ndropn-nfirst*)
have *4*: $((\exists\ x \in nset\ \sigma.\ w\ (NNil\ x)) \wedge (f\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma) \wedge$
$(\forall\ n \le nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma).\ w\ (NNil\ (nnth\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma)\ n))))) =$
$((\exists\ x \in nset\ \sigma.\ w\ (NNil\ x)) \wedge f\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ \sigma))$
by (*meson nkfilter-nnth-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*:
$\vdash (\ (init\ (\neg\ w))\ \mathcal{U}\ (\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(wprev\ (\Box\ (init\ (\neg\ w))))\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )$
using *StateUntilEqvWPrevChop* **by** *blast*

lemma *StatePiUntilsem2*:
$(\sigma \models (wprev\ (\Box\ (init\ (\neg\ w))))\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ )) =$
$(\sigma \models ((wprev\ (\Box\ (init\ (\neg\ w))))\frown\ ((init\ w) \wedge empty))\ \frown\ ((init\ w) \wedge (init\ w)\ \Pi\ f\ )\ )$
by (*auto simp add*: *schop-defs init-defs empty-defs zero-enat-def ndropn-nfirst*)
(*metis* (*no-types*, *lifting*) *add.right-neutral dual-order.refl enat.simps*(*3*) *enat-add-sub-same min.orderE min.orderE nfinite-ntaken nnth-nlast ntaken-nlast ntaken-nlength the-enat.simps zero-enat-def*)

lemma *StatePiUntil2*:
$\vdash (wprev\ (\Box\ (init\ (\neg\ w))))\frown(\ (init\ w) \wedge (init\ w)\ \Pi\ f\ ) =$
$(\ ((wprev\ (\Box\ (init\ (\neg\ w))))\frown\ ((init\ w) \wedge empty))\ \frown\ ((init\ w) \wedge (init\ w)\ \Pi\ f\ )\ )$
by (*simp add*: *StatePiUntilsem2 Valid-def*)

lemma *StatePiUntil3*:
$\vdash ((wprev\ (\Box\ (init\ (\neg\ w))))\frown\ ((init\ w) \wedge empty)) =$
$(\ ((init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w)))))\ \frown\ ((init\ w) \wedge empty))$
proof –
have *1*: $\vdash (wprev\ (\Box\ (init\ (\neg\ w))))\frown\ ((init\ w) \wedge empty) =$
$(init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge empty)$
by (*meson Prop11 StateUntilEqvWPrevChop*)

have *2*: $\vdash$ ((*init w*) $\wedge$ *empty*) = ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$ *w*))))$\frown$((*init w*) $\wedge$ *empty*)
by (*auto simp add*: *Valid-def itl-defs zero-enat-def ndropn-nfirst*)
(*metis ndropn-0 ndropn-nfirst* ,
metis add.right-neutral add-diff-inverse-nat enat.simps(*3*) *enat-add-sub-same enat-ord-simps*(*2*)
iless-Suc-eq less-eqE min.absorb2 min-def ntaken-all one-eSuc one-enat-def order-refl
plus-1-eq-Suc zero-enat-def zero-le)
show *?thesis* **by** (*metis 1 2 UntilChopDist inteq-reflection*)
qed

lemma *StatePiUntilsem4*:
(σ $\models$ ((*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$ *w*))))) $\frown$ ((*init w*) $\wedge$ *empty*)) =
(σ $\models$ ((*init w*) Π *empty*)$\frown$ ((*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*))))) $\frown$ ((*init w*) $\wedge$ *empty*) =
(((*init w*) Π *empty*)$\frown$ ((*init w*) $\wedge$ *empty*))
by (*simp add*: *StatePiUntilsem4 Valid-def*)

lemma *StatePiUntilsem*:
σ $\models$ (*init w*) Π *f* = (*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ (*init w*) Π *f*)
proof –
have *2*: (σ $\models$ (*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ (*init w*) Π *f*)) =
(σ $\models$ (*wprev* ($\Box$ (*init* ($\neg$ *w*))))$\frown$((*init w*) $\wedge$ (*init w*) Π *f*))
using *StateUntilEqvWPrevChopsem*[*of LIFT*($\neg$ *w*) *LIFT*((*init w*) $\wedge$ (*init w*) Π *f*) σ]
by *simp*
have *7*: (σ $\models$ (*wprev* ($\Box$ (*init* ($\neg$ *w*))))$\frown$((*init w*) $\wedge$ (*init w*) Π *f*)) =
(σ $\models$ (((*init w*) Π *empty*)$\frown$ ((*init w*) $\wedge$ *empty*))$\frown$((*init w*) $\wedge$ (*init w*) Π *f*))
by (*metis PiEmpty StatePiUntil2 StatePiUntil3 inteq-reflection*)
have *8*: (σ $\models$ (((*init w*) Π *empty*)$\frown$ ((*init w*) $\wedge$ *empty*))$\frown$((*init w*) $\wedge$ (*init w*) Π *f*)) =
(σ $\models$ (((*init w*) Π *empty*))$\frown$((*init w*) $\wedge$ (*init w*) Π *f*))
by (*auto simp add*: *schop-defs init-defs empty-defs zero-enat-def ndropn-nfirst*)
(*metis* (*no-types*, *opaque-lifting*) *enat-0-iff*(*2*) *min.absorb-iff1 ndropn-all ndropn-nlength*
nle-le nlength-NNil ntaken-nlast ntaken-nlength ntaken-ntaken)
have *9*: (σ $\models$ (((*init w*) Π *empty*))$\frown$((*init w*) $\wedge$ (*init w*) Π *f*)) =
(σ $\models$ (*init w*) Π (*empty*$\frown$*f*))
using *PiSChopDistsema PiSChopDistsemb* **by** *blast*
have *10*: (σ $\models$ (*init w*) Π (*empty*$\frown$*f*)) = (σ $\models$ (*init w*) Π *f*)
by (*metis EmptySChop inteq-reflection*)
show *?thesis*
by (*simp add*: *10 2 7 8 9*)
qed

lemma *StatePiUntil*:
$\vdash$ (*init w*) Π *f* = (*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ (*init w*) Π *f*)
using *StatePiUntilsem* **by** *blast*

lemma *StateAndPiEmpty*:
$\vdash$ ((*init w*) $\wedge$ (*init w*) Π *empty*) = (*w* $\wedge$ *empty*) $\frown$ (*wnext* ($\Box$ (*init* ($\neg$ *w*))))
proof –

have *1*: $\vdash$ ((*init w*) $\wedge$ (*init w*) Π *empty*) =
((*init w*) $\wedge$ (*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$ *w*))))))
using *PiEmpty* **by** *fastforce*
have *2*: $\vdash$ ((*init w*) $\wedge$ (*init* ($\neg$ *w*)) $\mathcal{U}$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$ *w*))))))
= ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$ *w*)))))
by (*auto simp add*: *until-d-def Valid-def init-defs ndropn-nfirst*)
(*metis ndropn-0 ndropn-nfirst neq0-conv*,
metis gr-implies-not-zero ndropn-0 ndropn-nfirst zero-enat-def zero-le)
have *3*: $\vdash$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$ *w*))))) = (*w* $\wedge$ *empty*) $\frown$ (*wnext* ($\Box$ (*init* ($\neg$ *w*))))
proof –
have $\bigwedge$*p pa*. $\vdash$((*p*::'*a nellist* $\Rightarrow$ *bool*) $\wedge$ *empty*) $\frown$ *pa* = (*init p* $\wedge$ *pa*)
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop inteq-reflection*)
then show *?thesis*
by (*simp add*: *Prop11*)
qed
show *?thesis*
by (*metis 1 2 3 inteq-reflection*)
qed

lemma *PiFPowerExpandsem*:
(σ $\models$ ($\exists$ *k*. (*init w*) Π (*fpower f k*))) =
(σ $\models$ (*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*f*$\frown$(*fpower f* (*k*)))))
proof –
have *1*: (σ $\models$ ($\exists$ *k*. (*init w*) Π (*fpower f k*))) =
($\exists$ *k*. (σ $\models$ (*init w*) Π (*fpower f k*)))
by *simp*
have *2*: ($\exists$ *k*. (σ $\models$ (*init w*) Π (*fpower f k*))) =
((σ $\models$ (*init w*) Π (*fpower f 0*)) $\vee$ ($\exists$ *k*. *1* $\leq$ *k* $\wedge$ (σ $\models$ (*init w*) Π (*fpower f* (*k*)))))
by (*metis One-nat-def diff-Suc-1 le-SucE le-add1 plus-1-eq-Suc*)
have *3*: (σ $\models$ (*init w*) Π (*fpower f 0*)) = (σ $\models$ (*init w*) Π *empty*)
by (*simp add*: *itl-def fpower-d-def*)
have *4*: ($\exists$ *k*. *1* $\leq$ *k* $\wedge$ (σ $\models$ (*init w*) Π (*fpower f* (*k*)))) =
($\exists$ *k*. (σ $\models$ (*init w*) Π (*fpower f* (*Suc k*))))
by (*metis le-add1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
have *5*: ($\exists$ *k*. (σ $\models$ (*init w*) Π (*fpower f* (*Suc k*)))) =
(σ $\models$ ($\exists$ *k*. (*init w*) Π (*fpower f* (*Suc k*))))
by *simp*
have *6*: ((σ $\models$ (*init w*) Π *empty*) $\vee$ (σ $\models$ ($\exists$ *k*. (*init w*) Π (*fpower f* (*Suc k*))))) =
(σ $\models$ (*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*f*$\frown$*fpower f* (*k*))))
unfolding *fpower-d-def* **by** (*auto simp add*: *schop-d-def*)
show *?thesis* **using** *1 2 3 4 5 6* **by** *blast*
qed

lemma *PiFPowerExpandsem1*:
$\forall$ σ. σ $\models$ ($\exists$ *k*. (*init w*) Π (*fpower f k*)) =
((*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*fpower f* (*Suc k*))))
proof
fix σ
show σ $\models$ ($\exists$ *k*. (*init w*) Π (*fpower f k*)) =
((*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*fpower f* (*Suc k*))))

proof –
have *1*: ($\sigma \models$ ($\exists k$. (*init w*) Π (*fpower f k*)) =
((*init w*) Π *empty* $\vee$ ($\exists k$. (*init w*) Π (*fpower f* (*Suc k*)))))
= (($\sigma \models$ ($\exists k$. (*init w*) Π (*fpower f k*))) =
($\sigma \models$ (*init w*) Π *empty* $\vee$ ($\exists k$. (*init w*) Π (*fpower f* (*Suc k*)))))

by *auto*
have *2*: (($\sigma \models$ ($\exists k$. (*init w*) Π (*fpower f k*))) =
($\sigma \models$ (*init w*) Π *empty* $\vee$ ($\exists k$. (*init w*) Π (*fpower f* (*Suc k*)))))
using *PiFPowerExpandsem*[*of w f* σ]
unfolding *fpower-d-def* **by** (*auto simp add*: *schop-d-def*)
show *?thesis* **using** *1 2* **by** *blast*
qed
qed

lemma *PiFPowerExpand*:
$\vdash$ ($\exists k$. (*init w*) Π (*fpower f k*)) =
((*init w*) Π *empty* $\vee$ ($\exists k$. (*init w*) Π (*fpower f* (*Suc k*))))
using *PiFPowerExpandsem1*[*of w f*] **by** (*auto simp add*: *Valid-def PiFPowerExpandsem1*)

lemma *exists-expand-sem*:
($\sigma \models$ ($\exists k$. (*fpower* ((*init w*) Π *f* $\wedge$ *fin w*) *k*))) =
(($\sigma \models$ (*fpower* ((*init w*) Π *f* $\wedge$ *fin w*) *0*)) $\vee$
($\sigma \models$ ($\exists k$. (*fpower* ((*init w*) Π *f* $\wedge$ *fin w*) (*Suc k*)))))
by (*metis* (*no-types*, *lifting*) *not0-implies-Suc unl-Rex*)

lemma *exists-expand*:
$\vdash$ ($\exists k$. (*fpower* ((*init w*) Π *f* $\wedge$ *fin w*) *k*)) =
((*fpower* ((*init w*) Π *f* $\wedge$ *fin w*) *0*) $\vee$ ($\exists k$. (*fpower* ((*init w*) Π *f* $\wedge$ *fin w*) (*Suc k*))))
using *exists-expand-sem Valid-def* **by** *fastforce*

13.3.16 TruePiEqv

lemma *TruePiEqvsem*:
$\sigma \models$ #*True* Π *f* = *f*
by (*simp add*: *pi-d-def pifilt-true*) (*metis zero-enat-def zero-le*)

lemma *TruePiEqv*:
$\vdash$ (#*True*) Π *f* = *f*
using *TruePiEqvsem* **by** (*auto simp add*: *Valid-def*)

13.3.17 BoxImpEqvPi

lemma *BoxImpEqvPi*:
$\vdash \Box$ *f* $\longrightarrow$ *g* = *f* Π *g*
proof (*simp add*: *Valid-def itl-defs pi-d-def pifilt-def sfxfilt-def*)
show $\forall w$. ($\forall n$. *enat n* $\leq$ *nlength w* $\longrightarrow$ *f* (*ndropn n w*)) $\longrightarrow$
g w = (($\exists i$. *enat i* $\leq$ *nlength w* $\wedge$ *f* (*ndropn i w*)) $\wedge$
g (*nmap* (λs. *nnth s 0*) (*nfilter f* (*ndropns w*))))
proof
fix *w*

show $(\forall\, n.\ enat\ n \leq nlength\ w \longrightarrow f\ (ndropn\ n\ w)) \longrightarrow$
$g\ w = ((\exists\, i.\ enat\ i \leq nlength\ w \wedge f\ (ndropn\ i\ w)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ w))))$
proof
assume *a0*: $\forall\, n.\ enat\ n \leq nlength\ w \longrightarrow f\ (ndropn\ n\ w)$
show $g\ w = ((\exists\, i.\ enat\ i \leq nlength\ w \wedge f\ (ndropn\ i\ w)) \wedge$
$g\ (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ w))))$
proof –
have *1*: $(nfilter\ f\ (ndropns\ w)) = (ndropns\ w)$
by (*metis* (*mono-tags*, *lifting*) *a0 ile0-eq in-nset-ndropns le-cases nfilter-id-conv zero-enat-def*)
have *2*: $w = (nmap\ (\lambda s.\ nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ w)))$
by (*simp add*: *1 nmap-first-ndropns*)
show *?thesis* **by** (*metis 1 a0 nmap-first-ndropns zero-enat-def zero-le*)
qed
qed
qed
qed

13.3.18 PiEqvDiamondUPi

lemma *PiEqvDiamondUPi*:
$\vdash f\ \Pi\ g = (\ \Diamond\ f \wedge f\ \Pi^u\ g)$
by (*simp add*: *Valid-def upi-d-def itl-defs pi-d-def*,*blast*)

13.3.19 PiEqvUntilPi

lemma *PiEqvUntilPi*:
$\vdash (init\ w)\ \Pi\ g = (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w)\ \Pi\ g)$
by (*metis StatePiUntil UntilUntil int-eq*)

13.3.20 UPiEqvBoxOrPi

lemma *UPiEqvBoxOrPi*:
$\vdash f\ \Pi^u\ g = (\Box\ (\neg\ f) \vee f\ \Pi\ g)$
by (*simp add*: *Valid-def upi-d-def itl-defs pi-d-def*,*blast*)

13.4 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*:
$\vdash f1\ \Pi^u\ f \wedge f1\ \Pi\ (\neg\ g) \longrightarrow f1\ \Pi\ (f \wedge \neg g)$
proof –
have *1*: $\vdash (\neg(f \longrightarrow g)) = (f \wedge \neg g)$
by *fastforce*
have *2*: $\vdash (\neg\ (f1\ \Pi^u\ (f \longrightarrow g))) = f1\ \Pi\ (\neg(f \longrightarrow g))$
by (*simp add*: *NotPiEqvNotUPi int-iffD1 int-iffD2 int-iffI*)
have *3*: $\vdash\ \neg(\ f1\ \Pi^u\ f \longrightarrow f1\ \Pi^u\ g\ ) \longrightarrow \neg\ (f1\ \Pi^u\ (f \longrightarrow g))$
by (*simp add*: *PiK*)
have *4*: $\vdash (\neg(\ f1\ \Pi^u\ f \longrightarrow f1\ \Pi^u\ g\ )\ ) = (f1\ \Pi^u\ f \wedge f1\ \Pi\ (\neg\ g))$
using *NotPiEqvNotUPi*[*of f1 g*] **by** *fastforce*
have *5*: $\vdash f1\ \Pi\ (\neg(f \longrightarrow g)) = f1\ \Pi\ (f \wedge \neg g)$
using *1* **by** (*simp add*: *PiEqvRule*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *UPiAndPiImpPiAndA*:
$\vdash f1\ \Pi^u\ f \wedge f1\ \Pi\ g \longrightarrow f1\ \Pi\ (f \wedge g)$
using *UPiAndPiImpPiAnd*[*of f1 f LIFT*($\neg g$)] **by** *fastforce*

lemma *PiAndPiImpPiAnd*:
$\vdash f1\ \Pi\ f \wedge f1\ \Pi\ g \longrightarrow f1\ \Pi\ (f \wedge g)$
proof –
have *1*: $\vdash\ f1\ \Pi^u\ f \wedge f1\ \Pi\ g \longrightarrow f1\ \Pi\ (f \wedge g)$
using *UPiAndPiImpPiAndA* **by** *fastforce*
have *2*: $\vdash f1\ \Pi\ f \longrightarrow f1\ \Pi^u\ f$
using *PiDc* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *PiAnd*:
$\vdash f\ \Pi\ (g1 \wedge g2) = (f\ \Pi\ g1 \wedge f\ \Pi\ g2)$
proof –
have *1*: $\vdash f\ \Pi\ (g1 \wedge g2) \longrightarrow f\ \Pi\ g1$
by (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)
have *2*: $\vdash f\ \Pi\ (g1 \wedge g2) \longrightarrow f\ \Pi\ g2$
by (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)
have *3*: $\vdash f\ \Pi\ (g1 \wedge g2) \longrightarrow f\ \Pi\ g1 \wedge f\ \Pi\ g2$
using *1 2* **by** *fastforce*
have *4*: $\vdash f\ \Pi\ g1 \wedge f\ \Pi\ g2 \longrightarrow f\ \Pi\ (g1 \wedge g2)$
by (*simp add*: *PiAndPiImpPiAnd*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *UPiAnd*:
$\vdash f\ \Pi^u\ (g1 \wedge g2) = (\ f\ \Pi^u\ g1 \wedge f\ \Pi^u\ g2\ )$

proof −
have *1*: $\vdash f\ \Pi\ (\neg g1 \lor \neg\ g2) = (f\ \Pi\ (\neg g1) \lor f\ \Pi\ (\neg\ g2))$
by (*simp add*: *PiOr*)
have *2*: $\vdash (\neg(f\ \Pi\ (\neg g1 \lor \neg\ g2))) = (\neg(f\ \Pi\ (\neg g1) \lor f\ \Pi\ (\neg\ g2)))$
using *1* **by** *fastforce*
have *3*: $\vdash (\neg(f\ \Pi\ (\neg g1 \lor \neg\ g2))) = f\ \Pi^u\ (\neg(\neg g1 \lor \neg\ g2))$
by (*meson NotUPiEqvNotPi Prop11*)
have *4*: $\vdash (\neg(\neg g1 \lor \neg\ g2)) = (g1 \land g2)$
by *fastforce*
have *5*: $\vdash f\ \Pi^u\ (\neg(\neg g1 \lor \neg\ g2)) = f\ \Pi^u\ (g1 \land g2)$
using *4* **by** (*simp add*: *UPiEqvRule*)
have *6*: $\vdash (\neg(f\ \Pi\ (\neg g1) \lor f\ \Pi\ (\neg\ g2))) = (\neg(f\ \Pi\ (\neg g1)) \land \neg(f\ \Pi(\neg g2)))$
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*: $\vdash (\neg(f\ \Pi\ (\neg g1)) \land \neg(f\ \Pi(\neg g2))) = (f\ \Pi^u\ g1 \land f\ \Pi^u\ g2)$
using *6 7 8* **by** *fastforce*
show *?thesis* **by** (*metis 2 3 5 6 9 inteq-reflection*)
qed

lemma *UPiOr*:
$\vdash f\ \Pi^u\ (g1 \lor g2) = (\ f\ \Pi^u\ g1 \lor f\ \Pi^u\ g2)$
proof −
have *1*: $\vdash f\ \Pi\ (\neg g1 \land \neg g2) = (f\ \Pi\ (\neg g1) \land f\ \Pi\ (\neg g2))$
by (*simp add*: *PiAnd*)
have *2*: $\vdash (\neg(f\ \Pi\ (\neg g1 \land \neg g2)))\ = (\neg(f\ \Pi\ (\neg g1) \land f\ \Pi\ (\neg g2)))$
using *1* **by** *fastforce*
have *3*: $\vdash (\neg(f\ \Pi\ (\neg g1 \land \neg g2))) = f\ \Pi^u\ (\neg(\neg g1 \land \neg\ g2))$
by (*meson NotUPiEqvNotPi Prop11*)
have *4*: $\vdash (\neg(\neg g1 \land \neg\ g2)) = (g1 \lor g2)$
by *fastforce*
have *5*: $\vdash f\ \Pi^u\ (\neg(\neg g1 \land \neg\ g2)) = f\ \Pi^u\ (g1 \lor g2)$
using *4 UPiEqvRule* **by** *blast*
have *6*: $\vdash (\neg(f\ \Pi\ (\neg g1) \land f\ \Pi\ (\neg g2))) = (\neg(f\ \Pi\ (\neg g1)) \lor \neg(f\ \Pi\ (\neg g2)))$
by *fastforce*
have *7*: $\vdash (\neg(f\ \Pi\ (\neg g1))) = f\ \Pi^u\ g1$
by (*simp add*: *upi-d-def*)
have *8*: $\vdash (\neg(f\ \Pi\ (\neg g2))) = f\ \Pi^u\ g2$
by (*simp add*: *upi-d-def*)
have *9*: $\vdash (\neg(f\ \Pi\ (\neg g1)) \lor \neg(f\ \Pi\ (\neg g2))) = (\ f\ \Pi^u\ g1 \lor f\ \Pi^u\ g2)$
using *7 8* **by** *fastforce*
show *?thesis*
by (*metis 2 3 4 6 9 inteq-reflection*)
qed

lemma *UpiAndImp*:
$\vdash f\ \Pi^u\ (g1 \longrightarrow g2) \land f\ \Pi\ g1 \longrightarrow f\ \Pi\ g2$

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 integ-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*:
$\vdash (init\ w)\ \Pi\ f = (wprev\ (\Box\ (init\ (\neg\ w))))\frown\ ((init\ w) \wedge (init\ w)\ \Pi\ f)$
  using *StatePiUntil StatePiUntil1* **by** *fastforce*

lemma *EmptyAndSChopAndMoreEqvAndSChop*:
$\vdash (w \wedge empty)\frown(f \wedge more) = ((w \wedge empty)\frown f \wedge more)$
proof –
 have *1*: $\vdash (w \wedge empty)\frown(f \wedge more) \longrightarrow (w \wedge empty)\frown f$
  by (*simp add*: *SChopAndA*)
 have *2*: $\vdash (w \wedge empty)\frown(f \wedge more) \longrightarrow more$
  by (*meson SChopAndB SChopMoreImpMore lift-imp-trans*)
 have *3*: $\vdash((w \wedge empty)\frown f \wedge more) \longrightarrow (w \wedge empty)\frown(f \wedge more)$
  by (*metis* (*no-types*, *opaque-lifting*) *InitAndEmptyEqvAndEmpty Prop11 Prop12 StateAndEmptySChop int-simps*(*1*) *inteq-reflection*)
 show *?thesis*
  by (*simp add*: *1 2 3 Prop12 int-iffI*)

qed

lemma *PiInfImpInf*:
$\vdash$ *f* Π *inf* $\longrightarrow$ *inf*
unfolding *Valid-def pi-d-def init-defs infinite-defs*
by *auto*
(*metis enat-ile nfinite-conv-nlength-enat pifilt-nlength-bound*)

lemma *DiamondWPrevBoxSChop*:
$\vdash \Diamond$ (*init w*) = (*wprev* ($\Box$ (*init* ($\neg$ *w*))))$\frown$ (*init w*)
by (*metis Initprop*(*2*) *StateUntilEqvWPrevChop UntilRule inteq-reflection*)

lemma *PiChopDist1*:
$\vdash$ ((*init w*) Π (*f*;*g*) $\vee$ ((*init w*) Π (*f* $\wedge$ *finite*) $\wedge$ *inf*)) =
(((*init w*) Π *f*);((*init w*) $\wedge$ (*init w*) Π *g*))
proof –
have *1*: $\vdash$ *f*;*g* = (*f*$\frown$*g* $\vee$ (*f* $\wedge$ *inf*))
by (*simp add*: *ChopSChopdef*)
have *2*: $\vdash$ (*init w*) Π (*f*;*g*) = ((*init w*) Π (*f*$\frown$*g*) $\vee$ (*init w*) Π (*f* $\wedge$ *inf*))
by (*metis 1 PiOr int-eq*)
have *3*: $\vdash$ (*init w*) Π (*f*$\frown$*g*) = ((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*init w*) Π *g*)
by (*simp add*: *PiSChopDist*)
have *4*: $\vdash$ ((*init w*) Π *f*);((*init w*) $\wedge$ (*init w*) Π *g*) =
(((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*init w*) Π *g*) $\vee$ ((*init w*) Π *f* $\wedge$ *inf*))
by (*simp add*: *ChopSChopdef*)
have *41*: $\vdash$ *f* = (*f* $\wedge$ (*finite* $\vee$ *inf*))
unfolding *finite-d-def* **by** *fastforce*
have *5*: $\vdash$ (*init w*) Π *f* = (*init w*) Π (*f* $\wedge$ (*finite* $\vee$ *inf*))
by (*simp add*: *41 PiEqvRule*)
have *6*: $\vdash$ ((*init w*) Π *f* $\wedge$ *inf*) =
(((*init w*) Π (*f* $\wedge$ *finite*) $\wedge$ *inf*) $\vee$ ((*init w*) Π (*f* $\wedge$ *inf*) $\wedge$ *inf*))
by (*metis AndInfEqvChopFalse OrChopEqvRule OrFiniteInf PiOr inteq-reflection*)
have *7*: $\vdash$ ((*init w*) Π (*f* $\wedge$ *inf*) $\wedge$ *inf*) = ((*init w*) Π (*f* $\wedge$ *inf*))
by (*metis EmptySChop PiImpRule PiInfImpInf Prop01 Prop04 Prop05 Prop10 int-iffD1 lift-imp-trans*)
show *?thesis*
using *2 3 4 6 7* **by** *fastforce*
qed

lemma *PiChopDist2*:
$\vdash$ ((*init w*) Π (*f*;*g*)) =
(((*init w*) Π (*f* $\wedge$ *finite*))$\frown$((*init w*) $\wedge$ (*init w*) Π *g*) $\vee$ ((*init w*) Π (*f* $\wedge$ *inf*) $\wedge$ *inf*))
proof –
have *1*: $\vdash$ (*init w*) Π (*f*;*g*) = ((*init w*) Π (*f*$\frown$*g*) $\vee$ (*init w*) Π (*f* $\wedge$ *inf*))
by (*metis ChopSChopdef PiOr inteq-reflection*)
have *2*: $\vdash$ (*init w*) Π (*f*$\frown$*g*) = ((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*init w*) Π *g*)

by (*simp add*: *PiSChopDist*)
have *3*: ⊢ ((*init w*) Π *f*)⁀((*init w*) ∧ (*init w*) Π *g*) =
((*init w*) Π *f* ∧ *finite*);((*init w*) ∧ (*init w*) Π *g*)
by (*simp add*: *schop-d-def*)
have *31*: ⊢ *f* = ((*f* ∧ *finite*) ∨ (*f* ∧ *inf*))
unfolding *finite-d-def* **by** *fastforce*
have *32*: ⊢ (*init w*) Π *f* = (*init w*) Π ((*f* ∧ *finite*) ∨ (*f* ∧ *inf*))
using *31 PiEqvRule* **by** *blast*
have *33*: ⊢ (*init w*) Π ((*f* ∧ *finite*) ∨ (*f* ∧ *inf*)) = ((*init w*) Π (*f* ∧ *finite*) ∨ (*init w*) Π (*f* ∧ *inf*))
by (*simp add*: *PiOr*)
have *34*: ⊢ ¬((*init w*) Π (*f* ∧ *inf*) ∧ *finite*)
unfolding *finite-d-def* **using** *PiAnd*[*of LIFT init w f LIFT inf*] *PiInfImpInf*[*of LIFT init w*]
by *fastforce*
have *4*: ⊢ ((*init w*) Π *f* ∧ *finite*) = ((*init w*) Π (*f* ∧ *finite*) ∧ *finite*)
using *32 33 34* **by** *fastforce*
have *5*: ⊢ (*init w*) Π (*f* ∧ *inf*) = ((*init w*) Π (*f* ∧ *inf*) ∧ *inf*)
by (*metis PiAnd PiInfImpInf Prop10 Prop12 int-eq int-iffD2*)
show *?thesis*
by (*metis 1 2 4 5 int-eq schop-d-def*)
qed

lemma *InfEqvNotForallNotLen*:
⊢ *inf* = (∀ *n*. ¬ (*len n*))
proof −
have *1*: ⊢ *finite* = (∃ *n*. *len n*)
by (*simp add*: *Finite-exist-len*)
have *2*: ⊢ (¬ *finite*) = (¬(∃ *n*. *len n*))
using *1*
by (*metis NotEqvYieldsMore int-eq*)
have *3*: ⊢ (¬(∃ *n*. *len n*)) = (∀ *n*. ¬ (*len n*))
by *fastforce*
show *?thesis* **using** *2 3* **by** (*metis InfEqvNotFinite int-eq*)
qed

lemma *PiEx*:
⊢ *f* Π (∃ *n*. *g n*) = (∃ *n*. *f* Π (*g n*))
unfolding *Valid-def* **by** (*auto simp add*: *pi-d-def*)

lemma *PiAll*:
⊢ *f* Π (∀ *n*. *g n*) = (∀ *n*. *f* Π (*g n*))
unfolding *Valid-def* **by** (*auto simp add*: *pi-d-def*)

lemma *PiLenSuc*:
⊢ (*init w*) Π (*len* (*Suc n*)) = ((*init w*) Π *skip*)⁀((*init w*) ∧ (*init w*) Π (*len n*))
proof −
have *1*: ⊢ *len* (*Suc n*) = *skip*⁀(*len n*)
by (*metis NextSChopdef len-d-def next-d-def wpow-Suc*)

have *2*: $\vdash$ (*init w*) Π (*skip*$\frown$(*len n*)) = ((*init w*) Π *skip*)$\frown$((*init w*) $\wedge$ (*init w*) Π (*len n*))
by (*simp add*: *PiSChopDist*)
show *?thesis* **by** (*metis 1 2 int-eq*)
qed

lemma *PiImpDiamond*:
$\vdash$ *f* Π *g* $\longrightarrow$ $\Diamond$ *f*
by (*meson PiEqvDiamondUPi Prop12 int-iffD1*)

lemma *PiMPA*:
assumes $\vdash$ *f* Π *g1*
$\vdash$ *f* Π (*g1* $\longrightarrow$ *g2*)
shows $\vdash$ *f* Π *g2*
using *assms*
by (*meson MP PiDc Prop09 UpiAndImp*)

lemma *PiMPB*:
assumes $\vdash$ *f* Π *g1*
$\vdash$ *g1* $\longrightarrow$ *g2*
shows $\vdash$ *f* Π *g2*
using *assms*
by (*metis MP PiAnd Prop10 Prop12 int-iffD2 inteq-reflection*)

lemma *PiMPBC*:
assumes $\vdash$ *f1* $\longrightarrow$ *f2* Π *g1*
$\vdash$ *g1* $\longrightarrow$ *g2*
shows $\vdash$ *f1* $\longrightarrow$ *f2* Π *g2*
using *assms*
by (*meson PiImpRule lift-imp-trans*)

lemma *SlideInitSFin*:
$\vdash$ *f*$\frown$((*init w*) $\wedge$ *g*) = (*f* $\wedge$ *sfin w*)$\frown$*g*
proof –
have *1*: $\vdash$ *sfin* (*init w*) = *sfin* (*w*)
by (*metis InitAndEmptyEqvAndEmpty SFinEqvTrueSChopAndEmpty int-eq*)
show *?thesis*
by (*metis 1 AndSFinSChopEqvStateAndSChop inteq-reflection*)
qed

lemma *UPiSChop*:
$\vdash$ (*init w*) Π^u (*f*$\frown$*g*) = ($\Box$(*init* ($\neg$*w*)) $\vee$ ((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π *g*))
proof –
have *1*: $\vdash$ (*init w*) Π^u (*f*$\frown$*g*) = ($\Box$ (*init* ($\neg$*w*)) $\vee$ (*init w*) Π (*f*$\frown$*g*))
by (*metis Initprop*(*2*) *UPiEqvBoxOrPi int-eq*)

have *2*: $\vdash$ (*init w*) Π ($f \frown g$) = ((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*init w*) Π *g*)
by (*simp add*: *PiSChopDist*)
have *3*: $\vdash$ ((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*init w*) Π *g*) =
((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π *g*)
by (*simp add*: *SlideInitSFin*)
show *?thesis*
using *1 2 3* **by** (*meson Prop06*)
qed

lemma *PiUPiSChop*:
$\vdash$ ((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π *g*) = ((*init w*) Π^u *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π^u *g*)
proof –
have *1*: $\vdash$ ((*init w*) Π *f*) = ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u *f*)
by (*simp add*: *PiEqvDiamondUPi*)
have *2*: $\vdash$ ((*init w*) Π *f* $\wedge$ *sfin w*) = ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u *f* $\wedge$ *sfin w*)
using *1* **by** *auto*
have *3*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ *sfin w*) = *sfin w*
by (*metis InitAndEmptyEqvAndEmpty Prop10 SChopAndA SFinEqvTrueSChopAndEmpty TrueSChopE-qvDiamond*
inteq-reflection lift-and-com)
have *4*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u *f* $\wedge$ *sfin w*) = ((*init w*) Π^u *f* $\wedge$ *sfin w*)
using *3* **by** *auto*
have *5*: $\vdash$ ((*init w*) Π *g*) = ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u *g*)
by (*simp add*: *PiEqvDiamondUPi*)
have *6*: $\vdash$ (*init w* $\wedge$ (*init w*) Π *g*) = (*init w* $\wedge$ $\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u *g*)
using *5* **by** *auto*
have *7*: $\vdash$ (*init w* $\wedge$ $\Diamond$ (*init w*)) = *init w*
by (*meson NowImpDiamond Prop10 Prop11*)
have *8*: $\vdash$ (*init w* $\wedge$ $\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u *g*) = (*init w* $\wedge$ (*init w*) Π^u *g*)
using *7* **by** *auto*
show *?thesis*
by (*metis 2 4 6 8 SlideInitSFin inteq-reflection*)
qed

lemma *PiFiniteAbsorb*:
$\vdash$ (*g* Π (*f* $\wedge$ *finite*) $\wedge$ *finite*) = (*g* Π *f* $\wedge$ *finite*)
proof –
have *1*: $\vdash$ (*g* Π (*f* $\wedge$ *finite*) $\wedge$ *finite*) $\longrightarrow$ (*g* Π *f* $\wedge$ *finite*)
by (*metis FiniteImp PiAnd PiAndPiImpPiAnd Prop01 Prop05 Prop12 inteq-reflection lift-and-com*)
have *31*: $\vdash$ *f* = ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*))
unfolding *finite-d-def* **by** *fastforce*
have *32*: $\vdash$ *g* Π *f* = *g* Π ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*))
using *31 PiEqvRule* **by** *blast*
have *33*: $\vdash$ *g* Π ((*f* $\wedge$ *finite*) $\vee$ (*f* $\wedge$ *inf*)) = (*g* Π (*f* $\wedge$ *finite*) $\vee$ *g* Π (*f* $\wedge$ *inf*))
by (*simp add*: *PiOr*)
have *34*: $\vdash$ $\neg$(*g* Π (*f* $\wedge$ *inf*) $\wedge$ *finite*)

unfolding *finite-d-def* **using** *PiAnd*[*of g f LIFT inf*] *PiInfImpInf*[*of g*]
by *fastforce*
have *4*: ⊢ (g Π f ∧ *finite*) = (g Π (f ∧ *finite*) ∧ *finite*)
using *32 33 34* **by** *fastforce*
have *2*: ⊢ (g Π f ∧ *finite*) ⟶ (g Π (f ∧ *finite*) ∧ *finite*)
by (*simp add*: *4 int-iffD1*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *PiInfAbsorb*:
⊢ (g Π (f ∧ *inf*) ∧ *inf*) = (g Π (f ∧ *inf*))
proof −
have *1*: ⊢ (g Π (f ∧ *inf*) ∧ *inf*) ⟶ (g Π (f ∧ *inf*))
by *fastforce*
have *2*: ⊢ (g Π (f ∧ *inf*)) ⟶ (g Π (f ∧ *inf*) ∧ *inf*)
by (*metis PiAnd PiInfImpInf Prop10 Prop12 int-iffD1 lift-imp-trans*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *PiMoreImpMore*:
⊢ (g Π *more*) ⟶ *more*
unfolding *Valid-def pi-d-def itl-defs pifilt-def sfxfilt-def*
by *auto*
(*metis enat-min-eq-0-iff length-nfilter-le min-def ndropns-nlength*)

lemma *PiMoreAbsorb*:
⊢ (g Π (f ∧ *more*) ∧ *more*) = (g Π (f ∧ *more*))
proof −
have *1*: ⊢ (g Π (f ∧ *more*) ∧ *more*) ⟶ (g Π (f ∧ *more*))
by *auto*
have *2*: ⊢ (g Π (f ∧ *more*)) ⟶ (g Π (f ∧ *more*) ∧ *more*)
by (*metis PiMoreImpMore PiAnd Prop10 Prop12 int-iffD1 inteq-reflection*)
show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *EmptyImpPiEmpty*:
⊢ (g Π f ∧ *empty*) ⟶ (g Π *empty*)
unfolding *Valid-def pi-d-def itl-defs pifilt-def sfxfilt-def*
by *auto*
(*metis ile0-eq length-nfilter-le ndropns-nlength*)

lemma *PiEmptyAbsorb*:
⊢ (g Π (f ∧ *empty*) ∧ *empty*) = (g Π f ∧ *empty*)
proof −
have *1*: ⊢ (g Π (f ∧ *empty*) ∧ *empty*) ⟶ (g Π f ∧ *empty*)
using *PiAnd* **by** *fastforce*

have *2*: $\vdash$ (g Π f $\wedge$ *empty*) $\longrightarrow$ (g Π (f $\wedge$ *empty*) $\wedge$ *empty*)
  using *EmptyImpPiEmpty PiAnd* **by** *fastforce*
 show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *SlideInitSFinVar1*:
$\vdash$ (*init w*) Π (((*F* $\wedge$ *more*) $\wedge$ *finite*);*X*) =
    (((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);((*init w*) Π *X*)
proof –
 have *1*: $\vdash$ (*init w*) Π (((*F* $\wedge$ *more*) $\wedge$ *finite*);*X*) =
        ((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π *X*)
  by (*metis PiSChopDist schop-d-def*)
 have *2*: $\vdash$ ((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π *X*) =
        (((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);((*init w*) Π *X*)
  using *SlideInitSFin*[*of LIFT* (*init w*) Π (*F* $\wedge$ *more*) *w LIFT* (*init w*) Π *X*] **unfolding** *schop-d-def*
  by *blast*
 show *?thesis* **using** *1 2* **by** (*metis int-eq*)
qed

lemma *UPiAbsorp*:
$\vdash$ (((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);((*init w*) Π^u *X*) =
   (((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);((*init w*) Π *X*)
proof –
 have *0*: $\vdash$ ((*init w*) Π^u *X*) = ($\Box$ (*init* ($\neg$*w*)) $\vee$ (*init w*) Π *X*)
 by (*metis Initprop*(*2*) *UPiEqvBoxOrPi inteq-reflection*)
 have *1*: $\vdash$ (((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);((*init w*) Π^u *X*) =
        (((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);($\Box$ (*init* ($\neg$*w*)) $\vee$ (*init w*) Π *X*)
        using *0 RightChopEqvChop* **by** *blast*
 have *2*: $\vdash$ (((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);($\Box$ (*init* ($\neg$*w*)) $\vee$ (*init w*) Π *X*) =
        (((*init w*) Π (*F* $\wedge$ *more*)) $\wedge$ *finite*);((*init w*) $\wedge$ ($\Box$ (*init* ($\neg$*w*)) $\vee$ (*init w*) Π *X*))
  using *SlideInitSFin*[*of LIFT* (*init w*) Π (*F* $\wedge$ *more*) *w LIFT* ($\Box$ (*init* ($\neg$*w*)) $\vee$ (*init w*) Π *X*)]
  unfolding *schop-d-def*
  by (*metis inteq-reflection*)
 have *3*: $\vdash$ ((*init w*) $\wedge$ ($\Box$ (*init* ($\neg$*w*)) $\vee$ (*init w*) Π *X*)) =
        ((*init w*) $\wedge$ ((*init w*) Π *X*))
  using *BoxElim Initprop*(*2*) **by** *fastforce*
 have *4*: $\vdash$ (((*init w*) Π (*F* $\wedge$ *more*)) $\wedge$ *finite*);((*init w*) $\wedge$ ($\Box$ (*init* ($\neg$*w*)) $\vee$ (*init w*) Π *X*)) =
        (((*init w*) Π (*F* $\wedge$ *more*)) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π *X*)
   using *3 RightChopEqvChop* **by** *blast*
 have *5*: $\vdash$ (((*init w*) Π (*F* $\wedge$ *more*)) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π *X*) =
        (((*init w*) Π (*F* $\wedge$ *more*) $\wedge$ *sfin w*) $\wedge$ *finite*);((*init w*) Π *X*)
   using *SlideInitSFin*[*of LIFT* ((*init w*) Π (*F* $\wedge$ *more*)) *w LIFT* (*init w*) Π *X*]
    unfolding *schop-d-def*
    by *auto*
 show *?thesis*
 by (*metis 1 2 4 5 inteq-reflection*)

qed

lemma *PiStateFinite*:
$\vdash$ (*init w*) Π *finite* = $\Diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))
proof –
have *1*: $\vdash$ (*init w*) Π *finite* = (*init w*) Π (#*True*$\frown$*empty*)
by (*metis DiamondEmptyEqvFinite DiamondSChopdef PiEqvRule int-eq*)
have *2*: $\vdash$ (*init w*) Π (#*True*$\frown$*empty*) = ((*init w*) Π #*True*) $\frown$((*init w*) $\wedge$ (*init w*) Π *empty*)
by (*simp add*: *PiSChopDist*)
have *3*: $\vdash$ ((*init w*) $\wedge$ (*init w*) Π *empty*) = ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$*w*))))
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop StateAndPiEmpty inteq-reflection*)
have *4*: $\vdash$ (*init w*) Π #*True* = $\Diamond$ (*init w*)
by (*simp add*: *PiTrueEqvDiamond*)
have *5*: $\vdash$ ((*init w*) Π #*True*) $\frown$((*init w*) $\wedge$ (*init w*) Π *empty*) =
($\Diamond$ (*init w*))$\frown$ ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$*w*))))
by (*simp add*: *3 4 SChopEqvSChop*)
have *6*: $\vdash$ ($\Diamond$ (*init w*))$\frown$ ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$*w*)))) =
($\Diamond$ (*init w*) $\wedge$ *sfin w*)$\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*))))
by (*simp add*: *SlideInitSFin*)
have *7*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ *sfin w*) = *sfin w*
by (*metis DiamondSChopdef InitAndEmptyEqvAndEmpty Prop10 SChopAndA SFinEqvTrueSChopAn-dEmpty*
inteq-reflection lift-and-com)
have *8*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ *sfin w*)$\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*)))) =
(*sfin w*) $\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*))))
using *7 LeftSChopEqvSChop* **by** *blast*
have *9*: $\vdash$ (*sfin w*) $\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*)))) = #*True* $\frown$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))
by (*metis SlideInitSFin int-eq int-simps*(*17*))
show *?thesis*
by (*metis 1 2 3 4 6 7 9 TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *PiStateInf*:
$\vdash$ (*init w*) Π *inf* = ($\Diamond$ (*init w*) $\wedge$ $\Box$ ((*init w*) $\longrightarrow$ $\bigcirc$ ($\Diamond$ (*init w*))))
proof –
have *1*: $\vdash$ (*init w*) Π *inf* = (*init w*) Π ($\neg$ *finite*)
by (*simp add*: *InfEqvNotFinite PiEqvRule*)
have *2*: $\vdash$ (*init w*) Π ($\neg$ *finite*) = ($\neg$ ((*init w*) Π^u *finite*))
by (*simp add*: *NotPiEqvNotUPi*)
have *3*: $\vdash$ ((*init w*) Π^u *finite*) = ($\Box$ (*init* ($\neg$ *w*)) $\vee$ (*init w*) Π *finite*)
by (*metis Initprop*(*2*) *UPiEqvBoxOrPi inteq-reflection*)
have *4*: $\vdash$ ($\Box$ (*init* ($\neg$ *w*)) $\vee$ (*init w*) Π *finite*) =
($\Box$ (*init* ($\neg$ *w*)) $\vee$ $\Diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*))))))
using *PiStateFinite* **by** *fastforce*
have *5*: $\vdash$ ($\neg$ ((*init w*) Π^u *finite*)) =
($\neg$ ($\Box$ (*init* ($\neg$ *w*)) $\vee$ $\Diamond$ ((*init w*) $\wedge$ (*wnext* ($\Box$ (*init* ($\neg$*w*)))))))
using *3 4* **by** *fastforce*

have *6*: $\vdash (\neg (\Box (init\ (\neg\ w)) \vee \Diamond ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w))))))) =$
$(\ \neg(\Box (init\ (\neg\ w))) \wedge \neg(\Diamond ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w)))))))$
by *force*
have *7*: $\vdash (\neg(\Box (init\ (\neg\ w)))) = \Diamond (init\ w)$
unfolding *always-d-def*
by (*metis Initprop*(*2*) *int-simps*(*4*) *inteq-reflection*)
have *8*: $\vdash (\neg(\Diamond ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w))))))) =$
$\Box(\ \neg ((init\ (w)) \wedge (wnext\ (\Box (init\ (\neg w))))))$
by (*simp add*: *always-d-def*)
have *9*: $\vdash (\ \neg ((init\ (w)) \wedge (wnext\ (\Box (init\ (\neg w)))))) =$
$(\neg(init\ (\ w)) \vee \neg (wnext\ (\Box (init\ (\neg w)))))$
by *force*
have *10*: $\vdash (\neg(init\ (\ w))) = (init\ (\neg\ w))$
by (*simp add*: *Initprop*(*2*))
have *11*: $\vdash (\neg (wnext\ (\Box (init\ (\neg w))))) = \bigcirc(\neg ((\Box (init\ (\neg w)))))$
by (*simp add*: *wnext-d-def*)
have *12*: $\vdash \bigcirc(\neg ((\Box (init\ (\neg w))))) = \bigcirc (\Diamond (init\ w))$
by (*simp add*: *7 NextEqvNext*)
have *13*: $\vdash (\neg(init\ (\ w)) \vee \neg (wnext\ (\Box (init\ (\neg w))))) = (\ (init\ \ w) \longrightarrow \bigcirc (\Diamond (init\ w)))$
using *10 11 12* **by** *fastforce*
have *14*: $\vdash \Box(\ \neg ((init\ (w)) \wedge (wnext\ (\Box (init\ (\neg w)))))) =$
$\Box\ (\ (init\ w) \longrightarrow \bigcirc (\Diamond (init\ w)))$
by (*metis 13 8 9 inteq-reflection*)
have *15*: $\vdash (\ \neg(\Box (init\ (\neg\ w))) \wedge \neg(\Diamond ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w))))))) =$
$(\ \Diamond (init\ w) \wedge \Box\ (\ (init\ \ w) \longrightarrow \bigcirc (\Diamond (init\ w))))$
by (*metis 13 6 7 8 9 inteq-reflection*)
show *?thesis*
by (*metis 1 15 2 5 6 int-eq*)
qed

lemma *UPiPiState*:
$\vdash (init\ w)\ \Pi^u\ (\ (init\ w)\ \Pi\ f) = (init\ w)\ \Pi^u\ f$
proof −
have *1*: $\vdash (init\ w)\ \Pi^u\ (\ (init\ w)\ \Pi\ f) = (\Box(init\ (\neg w)) \vee (init\ w)\ \Pi\ (\ (init\ w)\ \Pi\ f))$
by (*metis Initprop*(*2*) *UPiEqvBoxOrPi int-eq*)
have *2*: $\vdash (init\ w)\ \Pi\ (\ (init\ w)\ \Pi\ f) =\ (\ (init\ w)\ \Pi\ f)$
by (*metis PiAssoc Prop10 StateEqvBi StateImpBi inteq-reflection*)
have *3*: $\vdash (\Box(init\ (\neg w)) \vee (init\ w)\ \Pi\ (\ (init\ w)\ \Pi\ f)) =$
$(\Box(init\ (\neg w)) \vee\ (\ (init\ w)\ \Pi\ f))$
using *2* **by** *auto*
have *4*: $\vdash (\Box(init\ (\neg w)) \vee\ (\ (init\ w)\ \Pi\ f)) = (init\ w)\ \Pi^u\ f$
by (*metis Initprop*(*2*) *UPiEqvBoxOrPi int-eq*)
show *?thesis*
by (*metis 1 3 4 int-eq*)
qed

lemma *PiUPiState*:

$\vdash$ (*init w*) Π ((*init w*) Π^u *f*) = (*init w*) Π *f*
proof –
 have *1*: $\vdash$ (*init w*) Π ((*init w*) Π^u *f*) = ($\Diamond$(*init w*) $\wedge$ (*init w*) Π^u ((*init w*) Π^u *f*))
 by (*simp add*: *PiEqvDiamondUPi*)
 have *2*: $\vdash$ (*init w*) Π^u ((*init w*) Π^u *f*) = ((*init w*) Π^u *f*)
 by (*metis* (*no-types*, *lifting*) *NotUPiEqvNotPi UPiPiState inteq-reflection upi-d-def*)
 have *3*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ (*init w*) Π^u ((*init w*) Π^u *f*)) =
 ($\Diamond$(*init w*) $\wedge$ ((*init w*) Π^u *f*))
 using *2* **by** *auto*
 have *4*: $\vdash$ ($\Diamond$(*init w*) $\wedge$ ((*init w*) Π^u *f*)) = (*init w*) Π *f*
 by (*meson PiEqvDiamondUPi Prop11*)
 show *?thesis*
 by (*metis 1 3 4 int-eq*)
qed

lemma *PiStateFiniteAndFinite*:
$\vdash$ ((*init w*) Π *finite* $\wedge$ *finite*) = ($\Diamond$ (*init w*) $\wedge$ *finite*)
proof –
 have *1*: $\vdash$ ((*init w*) Π *finite* $\wedge$ *finite*) $\longrightarrow$ ($\Diamond$ (*init w*) $\wedge$ *finite*)
 using *PiImpDiamond* **by** *fastforce*
 have *2*: $\vdash$ ($\Diamond$ (*init w*) $\wedge$ *finite*) $\longrightarrow$ ((*init w*) Π *finite* $\wedge$ *finite*)
 by (*metis PiFiniteAbsorb PiTrueEqvDiamond TrueW int-simps*(*13*) *int-simps*(*17*) *inteq-reflection*)
 show *?thesis*
 using *1 2 int-iffI* **by** *blast*
qed

lemma *PiStateState*:
$\vdash$ (*init w*) Π (*init w*) = (*init w*) Π *#True*
by (*metis BoxElim PiImpDiamond PiMPBC PiTrueEqvDiamond Prop11 StatePiBoxState int-simps*(*17*) *inteq-reflection*)

lemma *StateAndPiState*:
$\vdash$ ((*init w*) $\wedge$ (*init w*) Π (*empty* $\wedge$ *w*)) = ((*init w*) $\wedge$ *wnext* ($\Box$ (*init* ($\neg$*w*))))
proof –
 have *2*: $\vdash$ ((*init w*) $\wedge$ *init w* Π *empty*) = (*init w* $\wedge$ *wnext* ($\Box$ (*init* ($\neg$ *w*))))
 by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop StateAndPiEmpty inteq-reflection*)
 have *5*: $\vdash$ (*init w*) Π *#True* = $\Diamond$ (*init w*)
 by (*simp add*: *PiTrueEqvDiamond*)
 show *?thesis*
 by (*metis 2 5 InitAndEmptyEqvAndEmpty PiAnd PiImpDiamond PiStateState Prop10 inteq-reflection lift-and-com*)
qed

lemma *PiStateFiniteSfin*:
$\vdash$ ((*init w*) Π *finite* $\wedge$ *sfin w*) = *sfin w*
proof –
 have *1*: $\vdash$ ((*init w*) Π *finite* $\wedge$ *sfin w*) =

$(((init\ w)\ \Pi\ finite \wedge finite) \wedge sfin\ w)$
by (*metis AndSFinEqvSChopAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty*
inteq-reflection lift-and-com)
have *2*: $\vdash (((init\ w)\ \Pi\ finite \wedge finite) \wedge sfin\ w) = ((\Diamond\ (init\ w) \wedge finite) \wedge sfin\ w)$
using *PiStateFiniteAndFinite* **by** *fastforce*
have *3*: $\vdash ((\Diamond\ (init\ w) \wedge finite) \wedge sfin\ w) = ((\Diamond\ (init\ w)) \wedge sfin\ w)$
by (*metis AndSFinEqvSChopAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty*
inteq-reflection lift-and-com)
have *4*: $\vdash ((\Diamond\ (init\ w)) \wedge sfin\ w) = sfin\ w$
by (*metis 3 InitAndEmptyEqvAndEmpty Prop11 Prop12 SChopAndA SFinEqvTrueSChopAndEmpty*
TrueSChopEqvDiamond inteq-reflection)
show *?thesis*
by (*metis 1 2 3 4 inteq-reflection*)
qed

lemma *PiStateSFin*:
$\vdash (init\ w)\ \Pi\ (sfin\ w) = (\ sfin\ w);(\ wnext\ (\Box\ (init\ (\neg w))))$
proof –
have *1*: $\vdash (sfin\ w) = finite;(empty \wedge w)$
by (*metis ChopAndCommute SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection sometimes-d-def*)
have *2*: $\vdash (init\ w)\ \Pi\ (finite;(empty \wedge w)) =$
$((init\ w)\ \Pi\ finite \wedge finite);((init\ w) \wedge (init\ w)\ \Pi\ (empty \wedge w))$
using *PiSChopDist*[*of w LIFT finite LIFT* (*empty* ∧ *w*)] **unfolding** *schop-d-def*
by *auto*
have *3*: $\vdash ((init\ w)\ \Pi\ finite \wedge finite);((init\ w) \wedge (init\ w)\ \Pi\ (empty \wedge w)) =$
$((init\ w)\ \Pi\ finite \wedge finite);((init\ w) \wedge wnext\ (\Box\ (init\ (\neg w))))$
using *RightChopEqvChop StateAndPiState* **by** *blast*
have *4*: $\vdash (sfin\ w \wedge finite) = sfin\ w$
by (*metis 1 ChopAndA DiamondEmptyEqvFinite Prop10 inteq-reflection sometimes-d-def*)
have *5*: $\vdash ((init\ w)\ \Pi\ finite \wedge finite);((init\ w) \wedge wnext\ (\Box\ (init\ (\neg w)))) =$
$((init\ w)\ \Pi\ finite \wedge sfin\ w);(\ wnext\ (\Box\ (init\ (\neg w))))$
using *SlideInitSFin*[*of LIFT* (*init w*) Π *finite w LIFT wnext* (□ (*init* (¬*w*)))] *4*
unfolding *schop-d-def*
by (*metis Prop10 Prop12 int-eq int-iffD2 lift-and-com*)
have *6*: $\vdash ((init\ w)\ \Pi\ finite \wedge sfin\ w);(\ wnext\ (\Box\ (init\ (\neg w)))) = (\ sfin\ w);(\ wnext\ (\Box\ (init\ (\neg w))))$
using *LeftChopEqvChop PiStateFiniteSfin* **by** *blast*
show *?thesis*
by (*metis 1 2 3 5 6 int-eq*)
qed

13.5 SChopstar and Pi

lemma *PiChopstarhelp2a*:
$\vdash (w \wedge empty)\frown(fpower\ (\ (f\frown(w \wedge empty)) \wedge more)\ k) =$
$(fpower\ (((w \wedge empty)\frown f) \wedge more)\ k\ )\ \frown(w \wedge empty)$
proof (*induction k*)

case *0*
then show *?case*
proof –
have *1*: $\vdash (w \wedge empty) \frown empty = empty \frown (w \wedge empty)$
by (*metis EmptySChop InitAndEmptyEqvAndEmpty StateAndEmptySChop int-eq*)
show *?thesis*
by (*metis 1 fpower-d-def wpow-0*)
qed
next
case (*Suc k*)
then show *?case*
proof –
have *1*: $\vdash (w \wedge empty) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ (Suc\ k) =$
$(w \wedge empty) \frown ((f \frown (w \wedge empty) \wedge more) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k)$
unfolding *fpower-d-def*
by (*simp add*: *schop-d-def*)
have *11*: $\vdash (f \wedge more) \frown (w \wedge empty) = (sfin\ w \wedge (f \wedge more))$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty int-eq*)
have *12*: $\vdash (f \frown (w \wedge empty)) = (sfin\ w \wedge f)$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection*)
have *13*: $\vdash (f \frown (w \wedge empty) \wedge more) = ((sfin\ w \wedge f) \wedge more)$
using *12* **by** *auto*
have *14*: $\vdash ((sfin\ w \wedge f) \wedge more) = (sfin\ w \wedge (f \wedge more))$
by *fastforce*
have *2*: $\vdash (f \frown (w \wedge empty) \wedge more) = (f \wedge more) \frown (w \wedge empty)$
using *11 13* **by** *fastforce*
have *20*: $\vdash ((f \frown (w \wedge empty) \wedge more) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k) =$
$((f \wedge more) \frown (w \wedge empty)) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k$
by (*simp add*: *2 LeftSChopEqvSChop*)
have *21*: $\vdash ((f \wedge more) \frown (w \wedge empty)) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k =$
$(f \wedge more) \frown ((w \wedge empty) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k)$
by (*meson Prop11 SChopAssoc*)
have *22*: $\vdash (f \wedge more) \frown ((w \wedge empty) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k) =$
$((f \wedge more) \frown (fpower\ ((w \wedge empty) \frown f \wedge more)\ k \frown (w \wedge empty)))$
by (*simp add*: *RightSChopEqvSChop Suc.IH*)
have *23*: $\vdash (w \wedge empty) \frown ((f \frown (w \wedge empty) \wedge more) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k) =$
$(w \wedge empty) \frown (((f \wedge more) \frown (w \wedge empty)) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k)$
using *20 RightSChopEqvSChop* **by** *blast*
have *24*: $\vdash (w \wedge empty) \frown (((f \wedge more) \frown (w \wedge empty)) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k) =$
$(w \wedge empty) \frown ((f \wedge more) \frown ((w \wedge empty) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k))$
using *21 RightSChopEqvSChop* **by** *blast*
have *25*: $\vdash (w \wedge empty) \frown ((f \wedge more) \frown ((w \wedge empty) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k)) =$
$(w \wedge empty) \frown ((f \wedge more) \frown (fpower\ ((w \wedge empty) \frown f \wedge more)\ k \frown (w \wedge empty)))$
using *23 22 RightSChopEqvSChop* **by** *blast*
have *3*: $\vdash (w \wedge empty) \frown ((f \frown (w \wedge empty) \wedge more) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k) =$
$(w \wedge empty) \frown ((f \wedge more) \frown (fpower\ ((w \wedge empty) \frown f \wedge more)\ k \frown (w \wedge empty)))$
using *23 24 25* **by** *fastforce*
have *4*: $\vdash (w \wedge empty) \frown ((f \wedge more) \frown (fpower\ ((w \wedge empty) \frown f \wedge more)\ k \frown (w \wedge empty))) =$
$((w \wedge empty) \frown (f \wedge more)) \frown ((fpower\ ((w \wedge empty) \frown f \wedge more)\ k \frown (w \wedge empty)))$
using *SChopAssoc* **by** *blast*

have *6*: $\vdash$ $((w \wedge empty) \frown (f \wedge more)) \frown ((fpower\ ((w \wedge empty) \frown f \wedge more)\ k \frown (w \wedge empty))) =$
$((w \wedge empty) \frown f \wedge more) \frown ((fpower\ ((w \wedge empty) \frown f \wedge more)\ k \frown (w \wedge empty)))$
by (*meson EmptyAndSChopAndMoreEqvAndSChop LeftSChopEqvSChop*)
have *7*: $\vdash$ $((w \wedge empty) \frown f \wedge more) \frown ((fpower\ ((w \wedge empty) \frown f \wedge more)\ k \frown (w \wedge empty))) =$
$((fpower\ ((w \wedge empty) \frown f \wedge more)\ (Suc\ k) \frown (w \wedge empty)))$
by (*metis SChopAssoc fpower-d-def schop-d-def wpow-Suc*)
show *?thesis* **using** *1 3 4 6 7*
by (*metis integ-reflection*)
qed
qed

lemma *PiChopstarhelp2*:
$\vdash$ $(w \wedge empty) \frown (schopstar(f \frown (w \wedge empty))) = (schopstar((w \wedge empty) \frown f)) \frown (w \wedge empty)$
proof –
have *1*: $\vdash$ $(w \wedge empty) \frown (schopstar(f \frown (w \wedge empty))) =$
$(w \wedge empty) \frown (\exists k.\ fpower\ (\ (f \frown (w \wedge empty)) \wedge more)\ k)$
by (*simp add*: *schopstar-d-def fpowerstar-d-def*)
have *2*: $\vdash$ $(w \wedge empty) \frown (\exists k.\ fpower\ (\ (f \frown (w \wedge empty)) \wedge more)\ k) =$
$(\exists k.\ (w \wedge empty) \frown (fpower\ (\ (f \frown (w \wedge empty)) \wedge more)\ k))$
using *SChopExist* **by** *fastforce*
have *3*: $\vdash$ $(\exists k.\ (w \wedge empty) \frown (fpower\ (\ (f \frown (w \wedge empty)) \wedge more)\ k)) =$
$(\exists k.\ (fpower\ ((w \wedge empty) \frown f \wedge more)\ k) \frown (w \wedge empty))$
by (*simp add*: *ExEqvRule PiChopstarhelp2a*)
have *4*: $\vdash$ $(\exists k.\ (fpower\ ((w \wedge empty) \frown f \wedge more)\ k) \frown (w \wedge empty)) =$
$(\exists k.\ (fpower\ ((w \wedge empty) \frown f \wedge more)\ k)) \frown (w \wedge empty)$
using *ExistSChop* **by** *fastforce*
have *5*: $\vdash$ $(\exists k.\ (fpower\ ((w \wedge empty) \frown f \wedge more)\ k)) \frown (w \wedge empty) =$
$(schopstar((w \wedge empty) \frown f)) \frown (w \wedge empty)$
by (*simp add*: *schopstar-d-def fpowerstar-d-def*)
show *?thesis*
by (*metis 1 2 3 4 5 int-eq*)
qed

lemma *PiFPowerSuca*:
$\vdash$ $(init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) = ((init\ w)\ \Pi\ f) \frown ((init\ w) \wedge (init\ w)\ \Pi\ (fpower\ f\ k))$
unfolding *fpower-d-def* **by** (*metis PiSChopDist schop-d-def wpow-Suc*)

lemma *PiFPowerSucb*:
$\vdash$ $(init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) = ((init\ w)\ \Pi\ f \wedge sfin\ w) \frown ((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k))$
proof –
have *0*: $\vdash$ $(init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) = ((init\ w)\ \Pi\ f) \frown ((init\ w) \wedge (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*meson PiFPowerSuca*)
have *1*: $\vdash ((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k)) = (w \wedge empty) \frown ((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*metis InitAndEmptyEqvAndEmpty Prop10 Prop12 StateAndEmptySChop int-iffD2 integ-reflection lift-and-com*)
have *2*: $\vdash ((init\ w)\ \Pi\ f) \frown ((init\ w) \wedge (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$((init\ w)\ \Pi\ f) \frown ((w \wedge empty) \frown ((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k)))$
using *1 RightSChopEqvSChop* **by** *blast*
have *3*: $\vdash$ $((init\ w)\ \Pi\ f) \frown ((w \wedge empty) \frown ((init\ w) \wedge\ (init\ w)\ \Pi\ (fpower\ f\ k))) =$

$(((init\ w)\ \Pi\ f)\frown\ (w \wedge\ empty))\frown((init\ w) \wedge\ \ (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*simp add*: *SChopAssoc*)
have *4*: $\vdash (((init\ w)\ \Pi\ f)\frown\ (w \wedge\ empty)) = ((init\ w)\ \Pi\ f \wedge\ sfin\ w)$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)
have *5*: $\vdash (((init\ w)\ \Pi\ f)\frown\ (w \wedge\ empty))\frown((init\ w) \wedge\ \ (init\ w)\ \Pi\ (fpower\ f\ k)) =$
$((init\ w)\ \Pi\ f \wedge\ sfin\ w)\frown((init\ w) \wedge\ \ (init\ w)\ \Pi\ (fpower\ f\ k))$
by (*simp add*: *4 LeftSChopEqvSChop*)
show *?thesis*
using *0 2 3 5* **by** *fastforce*
qed

lemma *PiFPowerSucc*:
$\vdash (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) =$
$(fpower\ ((init\ w)\ \Pi\ f \wedge\ sfin\ w)\ (Suc\ k))\frown((init\ w) \wedge (init\ w)\ \Pi\ empty)$
proof (*induction k*)
case *0*
then show *?case*
using *PiFPowerSucb*[*of w f 0*]
unfolding *schop-d-def fpower-d-def wpow-0 wpow-Suc*
proof –
assume *a1*: $\vdash init\ w\ \Pi\ ((f \wedge finite);empty) =$
$((init\ w\ \Pi\ f \wedge\ sfin\ w) \wedge\ finite);(init\ w \wedge\ init\ w\ \Pi\ empty)$
have $\vdash(((init\ w\ \Pi\ f \wedge\ sfin\ w) \wedge\ finite);empty \wedge\ finite) = ((init\ w\ \Pi\ f \wedge\ sfin\ w) \wedge\ finite)$
by (*metis AndChopB ChopEmpty Prop10 inteq-reflection*)
then show $\vdash init\ w\ \Pi\ ((f \wedge\ finite);empty) =$
$(((init\ w\ \Pi\ f \wedge\ sfin\ w) \wedge\ finite);empty \wedge\ finite);(init\ w \wedge\ init\ w\ \Pi\ empty)$
by (*metis a1 int-eq*)
qed
next
case (*Suc k*)
then show *?case*
proof –
have *3*: $\vdash (init\ w\ \Pi\ f)\ \frown$
$(init\ w \wedge\ fpower\ (init\ w\ \Pi\ f \wedge\ sfin\ w)\ (Suc\ k)\ \frown\ (init\ w \wedge\ init\ w\ \Pi\ empty)) =$
$((init\ w\ \Pi\ f \wedge\ sfin\ w)\ \frown\ fpower\ (init\ w\ \Pi\ f \wedge\ sfin\ w)\ (Suc\ k))\ \frown$
$(init\ w \wedge\ init\ w\ \Pi\ empty)$
by (*metis* (*no-types*, *lifting*) *AndSFinSChopEqvStateAndSChop InitAndEmptyEqvAndEmpty SChopAssoc SFinEqvTrueSChopAndEmpty inteq-reflection*)
have *4*: $\vdash(init\ w\ \Pi\ f)\ \frown\ (init\ w \wedge\ init\ w\ \Pi\ (f\ \frown\ fpower\ f\ k)) =$
$((init\ w\ \Pi\ f \wedge\ sfin\ w)\ \frown$
$((init\ w\ \Pi\ f \wedge\ sfin\ w)\ \frown\ fpower\ (init\ w\ \Pi\ f \wedge\ sfin\ w)\ k))\ \frown$
$(init\ w \wedge\ init\ w\ \Pi\ empty)$
using *3 Suc.IH* **unfolding** *fpower-d-def wpow-Suc schop-d-def* **by** (*metis int-eq*)
show *?thesis*
by (*metis 4 PiFPowerSuca fpower-d-def inteq-reflection schop-d-def wpow-Suc*)
qed
qed

lemma *PiFPowerSucd*:
$\vdash (init\ w)\ \Pi\ (fpower\ f\ (Suc\ k)) = (fpower\ ((init\ w)\ \Pi\ f \wedge\ sfin\ w)\ (Suc\ k))\frown((init\ w)\ \Pi\ empty)$

proof (*induction k*)
case *0*
then show *?case* **unfolding** *fpower-d-def wpow-0 wpow-Suc*
by (*metis* (*no-types*, *opaque-lifting*) *InitAndEmptyEqvAndEmpty PiSChopDist SChopAndEmptyEqvSChopAndEmpty*
SChopAssoc SFinEqvTrueSChopAndEmpty StateAndEmptySChop integ-reflection lift-and-com schop-d-def)
next
case (*Suc k*)
then show *?case*
proof −
have *1*: $\vdash$ *init w* Π *fpower f* (*Suc* (*Suc k*)) = (*init w*) Π (*f*$\frown$(*fpower f* (*Suc k*)))
by (*metis PiFPowerSuca PiSChopDist inteq-reflection*)
have *2*: $\vdash$ (*init w*) Π (*f*$\frown$(*fpower f* (*Suc k*))) = ((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*init w*) Π (*fpower f* (*Suc k*)))
by (*simp add*: *PiSChopDist*)
have *3*: $\vdash$ ((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*init w*) Π (*fpower f* (*Suc k*))) =
((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) $\wedge$ (*init w*) Π *empty*))
by (*metis 2 PiFPowerSucc inteq-reflection*)
have *4*: $\vdash$ ((*init w*) Π *f*)$\frown$((*init w*) $\wedge$ (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) $\wedge$ (*init w*) Π *empty*)) =
((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) $\wedge$ (*init w*) Π *empty*))
by (*metis AndSFinSChopEqvStateAndSChop InitAndEmptyEqvAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection*)
have *5*: $\vdash$ ((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) $\wedge$ (*init w*) Π *empty*)) =
((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc* (*Suc k*)))$\frown$((*init w*) $\wedge$ (*init w*) Π *empty*))
by (*metis 1 2 4 PiFPowerSucc inteq-reflection*)
have *6*: $\vdash$ ((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc* (*Suc k*)))$\frown$((*init w*) $\wedge$ (*init w*) Π *empty*)) =
((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc* (*Suc k*)))$\frown$((*init w*) Π *empty*))
by (*metis* (*no-types*, *lifting*) *PiFPowerSucc SChopAssoc Suc fpower-d-def inteq-reflection schop-d-def wpow-Suc*)
show *?thesis*
by (*metis 1 2 3 4 5 6 inteq-reflection*)
qed
qed

lemma *SFin-SChop*:
$\vdash$ (*f* $\wedge$ *sfin w*)$\frown$ *g* = (*f* $\wedge$ *fin w*)$\frown$*g*
proof −
have *1*: $\vdash$ (*f* $\wedge$ *fin w*)$\frown$ *g* = (*f* $\wedge$ (*fin w* $\wedge$ *finite*)) $\frown$ *g*
by (*metis* (*no-types*, *lifting*) *LeftChopEqvChop Prop11 Prop12 lift-and-com schop-d-def*)
have *2*: $\vdash$ (*fin w* $\wedge$ *finite*) = (*sfin w* $\wedge$ *finite*)
by (*metis* (*no-types*, *lifting*) *EmptyImpFinite FinAndEmpty FinEqvTrueChopAndEmpty Finprop*(*5*) *Prop10 SChopAndB SChopImpFinite SFinAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-imp-trans sfin-d-def*)
have *3*: $\vdash$ (*f* $\wedge$ *sfin w*)$\frown$ *g* = (*f* $\wedge$ (*sfin w* $\wedge$ *finite*)) $\frown$ *g*
by (*metis AndSChopCommute DiamondEmptyEqvFinite Prop10 SChopAndB SFinEqvTrueSChopAndEmpty*

TrueSChopEqvDiamond int-eq)
show *?thesis*
using *1 2 3* **by** (*metis inteq-reflection*)
qed

lemma *PiChopstar*:
⊢ (*init w*) Π (*schopstar f*) =
(*init* (¬ *w*)) 𝒰 ((*init w*) ∧ (*schopstar*(((*init w*) Π *f*) ∧ *sfin w*))⁀*wnext*(□ (*init* (¬ *w*))))
proof −
have *1*: ⊢ (*init w*) Π (*schopstar f*) = (*init w*) Π (∃ *k*. *fpower f k*)
by (*metis FPowerstardef PiEqvRule SChopstar-FPowerstar inteq-reflection*)
have *2*: ⊢ (*init w*) Π (∃ *k*. *fpower f k*) = (∃ *k*. (*init w*) Π (*fpower f k*))
by (*simp add*: *Valid-def pi-d-def*)
have *3*: ⊢ (∃ *k*. (*init w*) Π (*fpower f k*)) =
((*init w*) Π *empty* ∨ (∃ *k*. (*init w*) Π (*fpower f* (*Suc k*))))
using *PiFPowerExpand* **by** *auto*
have *4*: ⊢ (∃ *k*. (*init w*) Π (*fpower f* (*Suc k*))) =
(∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*))⁀((*init w*) Π *empty*))
by (*meson ExEqvRule PiFPowerSucd*)
have *5*: ⊢ (*init w*) Π *empty* = *empty* ⁀ ((*init w*) Π *empty*)
by (*simp add*: *EmptySChop int-iffD1 int-iffD2 int-iffI*)
have *6*: ⊢ (∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*))⁀((*init w*) Π *empty*)) =
(∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*)))⁀((*init w*) Π *empty*)
by (*simp add*: *ExistSChop*)
have *7*: ⊢ ((*init w*) Π *empty* ∨ (∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*))⁀((*init w*) Π *empty*)))=
(*empty*⁀ ((*init w*) Π *empty*) ∨
(∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*)))⁀((*init w*) Π *empty*))
using *5 6* **by** *fastforce*
have *8*: ⊢ (*empty*⁀ ((*init w*) Π *empty*) ∨
(∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*)))⁀((*init w*) Π *empty*)) =
(*empty* ∨ (∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*))))⁀((*init w*) Π *empty*)
by (*meson OrSChopEqv Prop11*)
have *9*: ⊢ *empty* = (*fpower* ((*init w*) Π *f* ∧ *sfin w*) *0*)
unfolding *fpower-d-def* **by** *simp*
have *10*: ⊢ (*empty* ∨ (∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*)))) =
((*fpower* ((*init w*) Π *f* ∧ *sfin w*) *0*) ∨ (∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*))))
unfolding *fpower-d-def* **by** *simp*
have *11*: ⊢ ((*fpower* ((*init w*) Π *f* ∧ *sfin w*) *0*) ∨ (∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*)))) =
(∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) *k*))
using *exists-expand*[*of w f*] **unfolding** *fpower-d-def*
by (*metis* (*mono-tags*) *Valid-def inteq-reflection wpow-0 wpowersem1*)
have *12*: ⊢ ((*init w*) Π *empty* ∨
(∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*))⁀((*init w*) Π *empty*))) =
(∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*k*)))⁀ ((*init w*) Π *empty*)
by (*metis 11 7 8 9 inteq-reflection*)
have *13*: ⊢ (∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*k*)))⁀ ((*init w*) Π *empty*) =
(*schopstar* ((*init w*) Π *f* ∧ *sfin w*)) ⁀ ((*init w*) Π *empty*)
by (*metis FPowerstardef LeftSChopEqvSChop SChopstar-FPowerstar inteq-reflection*)
have *14*: ⊢ (*schopstar* ((*init w*) Π *f* ∧ *sfin w*))⁀ ((*init w*) Π *empty*) =
(((*init w*) Π *empty*) ∨

$(((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w){\frown}(\mathit{schopstar}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w))){\frown}((\mathit{init}\ w)\ \Pi\ \mathit{empty}))$
by (*metis 5 OrSChopEqvRule SChopstar-unfoldl-eq inteq-reflection*)
have *15*: $\vdash ((\mathit{init}\ w)\ \Pi\ \mathit{empty}) = (\mathit{init}\ (\neg\ w))\ \mathcal{U}\ (\ (\mathit{init}\ w) \wedge \mathit{wnext}\ (\Box\ (\mathit{init}\ (\neg\ w))))$
by (*simp add*: *PiEmpty*)
have *16*:$\vdash (\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w) \frown (\mathit{schopstar}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w))\ ){\frown}((\mathit{init}\ w)\ \Pi\ \mathit{empty}) =$
$(\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w) \frown (\mathit{schopstar}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w))\ ){\frown}\ \mathit{wnext}\ (\Box\ (\mathit{init}\ (\neg\ w)))$
proof −
have *161*: $\vdash (\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w) \frown (\mathit{schopstar}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w))\ ){\frown}((\mathit{init}\ w)\ \Pi\ \mathit{empty}) =$
$(\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w) \frown ((\mathit{schopstar}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w)) \frown((\mathit{init}\ w)\ \Pi\ \mathit{empty})))$
by (*meson Prop11 SChopAssoc*)
have *162*: $\vdash ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w) = ((\mathit{init}\ w)\ \Pi\ f){\frown}(w \wedge \mathit{empty})$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)

have *163*: $\vdash (\mathit{schopstar}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w)) = (\mathit{schopstar}\ (((\mathit{init}\ w)\ \Pi\ f){\frown}\ (w \wedge \mathit{empty})))$
by (*metis 162 SChopstardef inteq-reflection*)
have *164*: $\vdash ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w) \frown (\mathit{schopstar}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w)) =$
$(\ (\mathit{init}\ w)\ \Pi\ f){\frown}((w \wedge \mathit{empty}){\frown}(\mathit{schopstar}\ (\ ((\mathit{init}\ w)\ \Pi\ f) \frown (w \wedge \mathit{empty}))))$
using *162 SChopAssoc int-eq* **by** *metis*
have *165*: $\vdash (\ (\mathit{init}\ w)\ \Pi\ f){\frown}((w \wedge \mathit{empty}){\frown}(\mathit{schopstar}\ (\ ((\mathit{init}\ w)\ \Pi\ f) \frown (w \wedge \mathit{empty})))) =$
$(\ (\mathit{init}\ w)\ \Pi\ f){\frown}\ ((\mathit{schopstar}\ (\ (w \wedge \mathit{empty}){\frown}\ (\ (\mathit{init}\ w)\ \Pi\ f))){\frown}(w \wedge \mathit{empty}))$
by (*simp add*: *PiChopstarhelp2 RightSChopEqvSChop*)
have *166*: $\vdash ((\ (\mathit{init}\ w)\ \Pi\ f){\frown}\ ((\mathit{schopstar}\ (\ (w \wedge \mathit{empty}){\frown}\ (\ (\mathit{init}\ w)\ \Pi\ f))){\frown}(w \wedge \mathit{empty}))){\frown}((\mathit{init}\ w)\ \Pi\ \mathit{empty}) =$
$(\ ((\ (\mathit{init}\ w)\ \Pi\ f){\frown}\ (\mathit{schopstar}\ (\ (w \wedge \mathit{empty}){\frown}\ (\ (\mathit{init}\ w)\ \Pi\ f)))) \frown(\ (\mathit{init}\ w) \wedge ((\mathit{init}\ w)\ \Pi\ \mathit{empty})\ )\ )$
by (*metis* (*no-types*, *lifting*) *InitAndEmptyEqvAndEmpty SChopAssoc StateAndEmptySChop int-eq*)
have *167*: $\vdash (\ ((\ (\mathit{init}\ w)\ \Pi\ f){\frown}\ (\mathit{schopstar}\ (\ (w \wedge \mathit{empty}){\frown}\ (\ (\mathit{init}\ w)\ \Pi\ f)))) \frown(\ (\mathit{init}\ w) \wedge ((\mathit{init}\ w)\ \Pi\ \mathit{empty})\ )\ ) =$
$(\ ((\ (\mathit{init}\ w)\ \Pi\ f){\frown}\ (\mathit{schopstar}\ (\ (w \wedge \mathit{empty}){\frown}\ (\ (\mathit{init}\ w)\ \Pi\ f)))) \frown((w \wedge \mathit{empty}) \frown \mathit{wnext}\ (\Box\ (\mathit{init}\ (\neg\ w)))\ )\ )$
by (*simp add*: *RightSChopEqvSChop StateAndPiEmpty*)
have *168*: $\vdash (\ ((\ (\mathit{init}\ w)\ \Pi\ f){\frown}\ (\mathit{schopstar}\ (\ (w \wedge \mathit{empty}){\frown}\ (\ (\mathit{init}\ w)\ \Pi\ f)))) \frown((w \wedge \mathit{empty}) \frown \mathit{wnext}\ (\Box\ (\mathit{init}\ (\neg\ w)))\ )\ ) =$
$(\ ((\ (\mathit{init}\ w)\ \Pi\ f){\frown}\ (\mathit{schopstar}\ (\ (w \wedge \mathit{empty}){\frown}\ (\ (\mathit{init}\ w)\ \Pi\ f)))) \frown(w \wedge \mathit{empty})) \frown \mathit{wnext}\ (\Box\ (\mathit{init}\ (\neg\ w)))$
by (*simp add*: *SChopAssoc*)
have *169*: $\vdash (\ ((\ (\mathit{init}\ w)\ \Pi\ f){\frown}\ (\mathit{schopstar}\ (\ (w \wedge \mathit{empty}){\frown}\ (\ (\mathit{init}\ w)\ \Pi\ f)))) \frown(w \wedge \mathit{empty})) =$
$(\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w) \frown (\mathit{schopstar}\ ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w))\ )$
using *164 165 SChopAssoc* **by** *fastforce*
show *?thesis* **by** (*metis 164 165 166 167 168 169 int-eq*)
qed
have *17*: $\vdash ((\mathit{init}\ w)\ \Pi\ f \wedge \mathit{sfin}\ w) = ((\mathit{init}\ w)\ \Pi\ f){\frown}((\mathit{init}\ w) \wedge \mathit{empty})$
by (*metis InitAndEmptyEqvAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty*
inteq-reflection lift-and-com)
have *18*: $\vdash ((\mathit{init}\ w)\ \Pi\ f) = \mathit{wprev}(\Box\ (\mathit{init}\ (\neg\ w))){\frown}((\mathit{init}\ w) \wedge (\mathit{init}\ w)\ \Pi\ f)$
by (*simp add*: *WPrevPi*)
have *19*: $\vdash \mathit{wprev}(\Box\ (\mathit{init}\ (\neg\ w))){\frown}((\mathit{init}\ w) \wedge (\mathit{init}\ w)\ \Pi\ f) =$
$\mathit{wprev}(\Box\ (\mathit{init}\ (\neg\ w))){\frown}(((\mathit{init}\ w) \wedge \mathit{empty}) \frown ((\mathit{init}\ w)\ \Pi\ f))$
by (*meson RightSChopImpSChop StateAndEmptySChop int-iffD1 int-iffD2 int-iffI*)

have *20*: $\vdash$ *wprev*(□ (*init* (¬ *w*)))⌢(((*init w*) ∧ *empty*) ⌢ ((*init w*) Π *f*)) =
(*init* (¬ *w*)) $\mathcal{U}$ ((*init w*) ∧ ((*init w*) Π *f*))
using *18 19 StatePiUntil* **by** *fastforce*
have *21*: $\vdash$ (((*init w*) Π *f* ∧ *sfin w*)⌢(*schopstar* ((*init w*) Π *f* ∧ *sfin w*)))⌢ *wnext* (□ (*init* (¬ *w*))) =
(*init* (¬ *w*)) $\mathcal{U}$ (
(*init w*) ∧
((((*init w*) Π *f*) ∧ *sfin w*)⌢(*schopstar* ((*init w*) Π *f* ∧ *sfin w*)))⌢*wnext* (□ (*init* (¬ *w*)))
)
proof −
have $\vdash$(*init w* Π *f*) ⌢ (*init w* ∧ *empty*) = (*init w* Π *f* ∧ *sfin w*)
by (*metis 17 inteq-reflection*)
then show *?thesis* **using** *StatePiUntil*[*of w f*]
UntilChopDist[*of w LIFT*((*init w* ∧ *empty*)) *LIFT*((*schopstar* ((*init w*) Π *f* ∧ *sfin w*)))⌢*wnext* (□ (*init* (¬ *w*)))]
by (*metis* (*no-types*, *lifting*) *AndSFinSChopEqvStateAndSChop SChopAssoc StateUntilEqvWPrevChop int-eq*)
qed
have *22*: $\vdash$ ((*init* (¬ *w*)) $\mathcal{U}$ ((*init w*) ∧ *wnext* (□ (*init* (¬ *w*)))) ∨
(*init* (¬ *w*)) $\mathcal{U}$ (
(*init w*) ∧
((((*init w*) Π *f*) ∧ *sfin w*)⌢(*schopstar* ((*init w*) Π *f* ∧ *sfin w*)))⌢*wnext* (□ (*init* (¬ *w*)))
)) =
(*init* (¬ *w*)) $\mathcal{U}$ (
((*init w*) ∧ *wnext* (□ (*init* (¬ *w*)))) ∨
((*init w*) ∧ ((((*init w*) Π *f*) ∧ *sfin w*)⌢(*schopstar*((*init w*) Π *f* ∧ *sfin w*)))⌢*wnext* (□ (*init* (¬ *w*))))
)
using *UntilOrDist* **by** *fastforce*
have *23*: $\vdash$ (
((*init w*) ∧ *wnext* (□ (*init* (¬ *w*)))) ∨
((*init w*) ∧ ((((*init w*) Π *f*) ∧ *sfin w*)⌢(*schopstar* ((*init w*) Π *f* ∧ *sfin w*)))⌢*wnext* (□ (*init* (¬ *w*))))
) =
((*init w*) ∧ (*schopstar* (((*init w*) Π *f*) ∧ *sfin w*))⌢*wnext*(□ (*init* (¬ *w*))))
by (*metis* (*mono-tags*, *lifting*) *EmptyOrSChopEqv Prop11 SChopOrEqvRule SChopstar-unfoldl-eq State-AndEmptySChop int-eq*)
have *24*: $\vdash$ (*init w*) Π (*schopstar f*) = ((*init w*) Π *empty* ∨ (∃ *k*. (*init w*) Π (*fpower f* (*Suc k*))))
using *1 2 3* **by** *fastforce*
have *25*: $\vdash$ ((*init w*) Π *empty* ∨ (∃ *k*. (*init w*) Π (*fpower f* (*Suc k*)))) =
((*init w*) Π *empty* ∨ (∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*))⌢((*init w*) Π *empty*)))
using *4* **by** *fastforce*
have *26*: $\vdash$ ((*init w*) Π *empty* ∨
(∃ *k*. (*fpower* ((*init w*) Π *f* ∧ *sfin w*) (*Suc k*))⌢((*init w*) Π *empty*))) =
(*schopstar* ((*init w*) Π *f* ∧ *sfin w*)) ⌢ ((*init w*) Π *empty*)
using *12 13* **by** *fastforce*
have *27*: $\vdash$ (*schopstar* ((*init w*) Π *f* ∧ *sfin w*))⌢ ((*init w*) Π *empty*) =
((*init* (¬ *w*)) $\mathcal{U}$ ((*init w*) ∧ *wnext* (□ (*init* (¬ *w*)))) ∨
(((*init w*) Π *f* ∧ *sfin w*) ⌢ (*schopstar* ((*init w*) Π *f* ∧ *sfin w*)))⌢ *wnext* (□ (*init* (¬ *w*))))
using *14 15 16* **by** *fastforce*
have *28*:$\vdash$ (*schopstar* ((*init w*) Π *f* ∧ *sfin w*)) ⌢ ((*init w*) Π *empty*) =
(*init* (¬ *w*)) $\mathcal{U}$ ((*init w*) ∧ (*schopstar* (((*init w*) Π *f*) ∧ *sfin w*))⌢*wnext*(□ (*init* (¬ *w*))))
using *27 21 22 23*

by (*metis inteq-reflection*)
show *?thesis*
by (*metis 24 25 26 28 inteq-reflection*)
qed

lemma *PiChopstar-var1*:
$\vdash$ (*init w*) Π (*schopstar f*) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)
proof –
have *1*: $\vdash$ (*init w*) Π (*schopstar f*) = (*init w*) Π ($\exists$ *k*. *fpower f k*)
by (*metis FPowerstardef PiEqvRule SChopstar-FPowerstar inteq-reflection*)
have *2*: $\vdash$ (*init w*) Π ($\exists$ *k*. *fpower f k*) = ($\exists$ *k*. (*init w*) Π (*fpower f k*))
by (*simp add*: *Valid-def pi-d-def*)
have *3*: $\vdash$ ($\exists$ *k*. (*init w*) Π (*fpower f k*)) =
((*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*init w*) Π (*fpower f* (*Suc k*))))
using *PiFPowerExpand* **by** *auto*
have *4*: $\vdash$ ($\exists$ *k*. (*init w*) Π (*fpower f* (*Suc k*))) =
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) Π *empty*))
by (*meson ExEqvRule PiFPowerSucd*)
have *5*: $\vdash$ (*init w*) Π *empty* = *empty* $\frown$ ((*init w*) Π *empty*)
by (*simp add*: *EmptySChop int-iffD1 int-iffD2 int-iffI*)
have *6*: $\vdash$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) Π *empty*)) =
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))$\frown$((*init w*) Π *empty*)
by (*simp add*: *ExistSChop*)
have *7*: $\vdash$ ((*init w*) Π *empty* $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) Π *empty*)))=
(*empty*$\frown$ ((*init w*) Π *empty*) $\vee$
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))$\frown$((*init w*) Π *empty*))
using *5 6* **by** *fastforce*
have *8*: $\vdash$ (*empty*$\frown$ ((*init w*) Π *empty*) $\vee$
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))$\frown$((*init w*) Π *empty*)) =
(*empty* $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))))$\frown$((*init w*) Π *empty*)
by (*meson OrSChopEqv Prop11*)
have *9*: $\vdash$ *empty* = (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) *0*)
unfolding *fpower-d-def* **by** *simp*
have *10*: $\vdash$ (*empty* $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))) =
((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) *0*) $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))))
unfolding *fpower-d-def* **by** *simp*
have *11*: $\vdash$ ((*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) *0*) $\vee$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*)))) =
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) *k*))
using *exists-expand*[*of w f*] **unfolding** *fpower-d-def*
by (*metis* (*mono-tags*) *Valid-def inteq-reflection wpow-0 wpowersem1*)
have *12*: $\vdash$ ((*init w*) Π *empty* $\vee$
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*Suc k*))$\frown$((*init w*) Π *empty*))) =
($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*k*)))$\frown$ ((*init w*) Π *empty*)
by (*metis 11 7 8 9 inteq-reflection*)
have *13*: $\vdash$ ($\exists$ *k*. (*fpower* ((*init w*) Π *f* $\wedge$ *sfin w*) (*k*)))$\frown$ ((*init w*) Π *empty*) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)
by (*metis FPowerstardef LeftSChopEqvSChop SChopstar-FPowerstar inteq-reflection*)
show *?thesis*

by (*metis 1 12 13 2 3 4 inteq-reflection*)
qed

lemma *PiChopstar-var2*:
 $\vdash$ (*init w*) Π (*schopstar f*) =
  ((*init w*) Π *empty* $\vee$
  (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$(((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*))))))

proof –
 have *1*: $\vdash$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) =
  (*empty* $\vee$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ ((*init w*) Π *f* $\wedge$ *sfin w*))
  by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite Prop10 Prop12 SCSEqvOrChopSCSB SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond int-eq int-iffD2 lift-and-com*)
 have *2*: $\vdash$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*) =
  (((*init w*) Π *empty*) $\vee$ ((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$((*init w*) Π *empty*))
  using *1 EmptyOrSChopEqvRule* **by** *blast*
 have *3*: $\vdash$ ((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$((*init w*) Π *empty*) =
  (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ (((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π *empty*))
  by (*meson Prop11 SChopAssoc*)
 have *4*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π *empty*)) =
  (((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*)))))
  by (*metis* (*no-types*, *lifting*) *InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop StateAnd-PiEmpty inteq-reflection*)
 have *5*: $\vdash$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$ (((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$((*init w*) Π *empty*)) =
  (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))$\frown$(((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*)))))
  by (*metis 4 RightChopEqvChop schop-d-def*)
 show *?thesis*
 by (*metis 2 3 5 PiChopstar-var1 int-eq*)
qed

lemma *PiChopstar-var3*:
 $\vdash$ (*init w*) Π (*f*$\frown$(*schopstar f*)) =
  (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$(((*init w*) Π *f* $\wedge$ *sfin w*)$\frown$(*wnext* ($\Box$ (*init* ($\neg$*w*)))))
proof –
 have *1*: $\vdash$ (*init w*) Π (*f*$\frown$(*schopstar f*)) =
  ((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ ((*init w*) Π (*schopstar f*))
  by (*metis PiSChopDist SlideInitSFin inteq-reflection*)
 have *2*: $\vdash$ ((*init w*) Π (*schopstar f*)) = (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)
  using *PiChopstar-var1* **by** *auto*
 have *3*: $\vdash$ ((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ ((*init w*) Π (*schopstar f*)) =
  ((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*))
  using *2 RightSChopEqvSChop* **by** *blast*
 have *4*: $\vdash$ ((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)) =
  (((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))) $\frown$ ((*init w*) Π *empty*)
  by (*simp add*: *SChopAssoc*)

have *5*: ⊢ $(((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown (((init\ w)\ \Pi\ f \wedge sfin\ w) \wedge finite)$
by (*simp add*: *SChopplusCommute*)
have *6*: ⊢ $(((init\ w)\ \Pi\ f \wedge sfin\ w) \wedge finite) = (((init\ w)\ \Pi\ f \wedge sfin\ w))$
using *SFinEQvFinAndFinite* **by** *fastforce*
have *7*: ⊢ $(((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown ((init\ w)\ \Pi\ empty) =$
$((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (wnext\ (\Box\ (init\ (\neg w))))$
by (*metis* (*no-types*, *lifting*) *InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop StateAndPiEmpty inteq-reflection*)
show *?thesis*
by (*metis* (*no-types*, *lifting*) *1 2 5 6 7 SChopAssoc inteq-reflection*)
qed

lemma *PiChopstar-var4*:
⊢ $(init\ w)\ \Pi\ (f \frown (schopstar\ f)\ ) =$
$(((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))) \frown (wnext\ (\Box\ (init\ (\neg w))))$
proof –
have *1*: ⊢ $(init\ w)\ \Pi\ (f \frown (schopstar\ f)\ ) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown (((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (wnext\ (\Box\ (init\ (\neg w)))))$
using *PiChopstar-var3* **by** *blast*
have *2*: ⊢ $(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown (((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (wnext\ (\Box\ (init\ (\neg w))))) =$
$((schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown (wnext\ (\Box\ (init\ (\neg w))))$
by (*simp add*: *SChopAssoc*)
have *3*: ⊢ $((init\ w)\ \Pi\ f \wedge sfin\ w) = (((init\ w)\ \Pi\ f \wedge sfin\ w) \wedge finite)$
using *SFinEQvFinAndFinite* **by** *fastforce*
have *4*: ⊢ $(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown (((init\ w)\ \Pi\ f \wedge sfin\ w) \wedge finite) =$
$(((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)))$
by (*simp add*: *SChopstar-slide-var*)
show *?thesis*
by (*metis 1 2 3 4 int-eq*)
qed

lemma *PiChopstar-var5*:
⊢ $(init\ w)\ \Pi\ (f \frown (schopstar\ f)\ ) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown ((init\ w)\ \Pi\ (f \wedge finite))$
proof –
have *1*: ⊢ $(init\ w)\ \Pi\ (f \frown (schopstar\ f)) =$
$((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (\ (init\ w)\ \Pi\ (schopstar\ f))$
by (*metis PiSChopDist SlideInitSFin inteq-reflection*)
have *2*: ⊢ $((init\ w)\ \Pi\ (schopstar\ f)) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown ((init\ w)\ \Pi\ empty)$
using *PiChopstar-var1* **by** *auto*
have *3*: ⊢ $((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (\ (init\ w)\ \Pi\ (schopstar\ f)) =$
$((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (\ (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown ((init\ w)\ \Pi\ empty))$
using *2 RightSChopEqvSChop* **by** *blast*
have *4*: ⊢ $((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (\ (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown ((init\ w)\ \Pi\ empty)) =$
$(((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))) \frown ((init\ w)\ \Pi\ empty)$
using *SChopAssoc* **by** *blast*
have *5*: ⊢ $(((init\ w)\ \Pi\ f \wedge sfin\ w) \frown (schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w))) =$
$(schopstar\ ((init\ w)\ \Pi\ f \wedge sfin\ w)) \frown (((init\ w)\ \Pi\ f \wedge sfin\ w) \wedge finite)$

by (*simp add*: *SChopplusCommute*)
have *6*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*) $\wedge$ *finite*) = (((*init w*) Π *f* $\wedge$ *sfin w*))
using *SFinEQvFinAndFinite* **by** *fastforce*
have *7*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*) =
(*init w*) Π (*f*$\frown$*empty*)
by (*metis PiSChopDist SlideInitSFin inteq-reflection*)
have *8*: $\vdash$ (*init w*) Π (*f*$\frown$*empty*) = (*init w*) Π (*f* $\wedge$ *finite*)
by (*simp add*: *ChopEmpty PiEqvRule schop-d-def*)
have *9*: $\vdash$ (((*init w*) Π *f* $\wedge$ *sfin w*) $\frown$ (*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*))) $\frown$ ((*init w*) Π *empty*) =
((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$(((*init w*) Π *f* $\wedge$ *sfin w*))) $\frown$ ((*init w*) Π *empty*)
by (*metis 4 5 6 inteq-reflection*)
have *10*: $\vdash$ ((*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$(((*init w*) Π *f* $\wedge$ *sfin w*))) $\frown$ ((*init w*) Π *empty*) =
(*schopstar* ((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$((((*init w*) Π *f* $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*))
by (*meson Prop11 SChopAssoc*)
show *?thesis*
by (*metis 1 10 3 4 7 8 9 int-eq*)
qed

lemma *PiChopstar-var6*:
$\vdash$ (*init w*) Π (*schopstar f*) =
((((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))
$\frown$(*wnext*($\Box$(*init* ($\neg$*w*)))))
proof –
have *1*: $\vdash$ (*schopstar f*) = (*schopstar* (*f* $\vee$ *empty*))
by (*metis SChopstar-star2 SChopstar-swap inteq-reflection*)
have *1*: $\vdash$ (*init w*) Π (*schopstar* (*f* $\vee$ *empty*)) =
(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*)
by (*simp add*: *PiChopstar-var1*)
have *2*: $\vdash$ (*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) =
(*empty* $\vee$ ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))
by (*meson Prop06 SCSEqvOrChopSCSB SChopstar-slide-var*)
have *3*: $\vdash$ (*empty* $\vee$ ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ ((*init w*) Π *empty*)
= (((*init w*) Π *empty*) $\vee$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ ((*init w*) Π *empty*))

using *EmptyOrSChopEqv* **by** *blast*
have *4*: $\vdash$ ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) =
(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*))$\frown$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*) $\wedge$ *finite*)
by (*simp add*: *SChopplusCommute*)
have *5*: $\vdash$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*) $\wedge$ *finite*) =
(((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*))
using *SFinEQvFinAndFinite* **by** *fastforce*
have *6*: $\vdash$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) $\frown$ ((*init w*) Π *empty*) =
(((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)) $\frown$ (*wnext*($\Box$(*init* ($\neg$*w*))))
by (*metis* (*no-types*, *lifting*) *InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop State-AndPiEmpty int-eq*)
have *7*: $\vdash$ (((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)$\frown$(*schopstar* ((*init w*) Π (*f* $\vee$ *empty*) $\wedge$ *sfin w*)))$\frown$ ((*init w*)

Π *empty*) =
(((*init w*) Π (*f* ∨ *empty*) ∧ *sfin w*)⌢(*schopstar* ((*init w*) Π (*f* ∨ *empty*) ∧ *sfin w*)))⌢ (*wnext*(□(*init* (¬*w*))))
by (*metis* (*no-types*, *lifting*) *4 5 6 SChopAssoc inteq-reflection*)
have *8*: ⊢ (((*init w*) Π *empty*) ∨ (((*init w*) Π (*f* ∨ *empty*) ∧ *sfin w*)⌢(*schopstar* ((*init w*) Π (*f* ∨ *empty*) ∧ *sfin w*)))⌢ ((*init w*) Π *empty*)) =
((((*init w*) Π (*f* ∨ *empty*) ∧ *sfin w*)⌢(*schopstar* ((*init w*) Π (*f* ∨ *empty*) ∧ *sfin w*)))⌢ ((*init w*) Π *empty*))
by (*metis 1 2 3 7 FiniteAndEmptyEqvEmpty PiChopstar-var4 Prop05 SChopstar-sup-id-star1 int-iffD1 inteq-reflection*)
show *?thesis*
by (*meson PiChopstar-var4 PiEqvRule Prop04 SChopstar-star2 SChopstar-sup-id-star1 SChopstar-swap UntilImpOr UntilIntro lift-imp-trans*)
qed

13.6 Omega and Pi

lemma *OmegaImpDiamond*:
⊢ (*omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*)) ⟶ ◇ (*init w*)
proof –
have *1*: ⊢ (*omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*)) =
((((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*) ∧ *more*) ∧ *finite*);(*omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*))
using *OmegaUnroll*[*of LIFT* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*)] **by** *blast*
have *2*: ⊢ ((((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*) ∧ *more*) ∧ *finite*) ⟶ ◇ (*init w*)
using *PiImpDiamond* **by** *fastforce*
have *3*: ⊢ ((((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*) ∧ *more*) ∧ *finite*) =
((((*init w*) Π (*f* ∧ *more*) ∧ *more*) ∧ *sfin w*) ∧ *finite*)
by *auto*
have *4*: ⊢ ((((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*) ∧ *more*) ∧ *finite*);(*omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*)) =
((((*init w*) Π (*f* ∧ *more*) ∧ *more*) ∧ *sfin w*) ∧ *finite*);(*omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*))
using *3 LeftChopEqvChop* **by** *blast*
have *5*: ⊢ ((((*init w*) Π (*f* ∧ *more*) ∧ *more*) ∧ *sfin w*) ∧ *finite*);(*omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*))
= (((*init w*) Π (*f* ∧ *more*) ∧ *more*) ∧ *finite*);((*init w*) ∧ *omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*))
using *SlideInitSFin*[*of LIFT* ((*init w*) Π (*f* ∧ *more*) ∧ *more*) *w*
LIFT (*omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*))] **unfolding** *schop-d-def*
by (*metis inteq-reflection*)
have *6*: ⊢ (((*init w*) Π (*f* ∧ *more*) ∧ *more*) ∧ *finite*);((*init w*) ∧ *omega* ((*init w*) Π (*f* ∧ *more*) ∧ *sfin w*))
⟶ ◇ (*init w*)
by (*metis ChopAndB FiniteChopImpDiamond inteq-reflection lift-and-com lift-imp-trans*)
show *?thesis* **using** *1 3 4 5 6* **by** (*metis int-eq*)
qed

lemma *PiStateOmegaUnroll*:
⊢ (*init w*) Π (*omega f*) = (((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *more*) ∧ *sfin w*); ((*init w*) Π (*omega f*))
proof –
have *1*: ⊢ (*init w*) Π (*omega f*) = (*init w*) Π (((*f* ∧ *more*) ∧ *finite*);(*omega f*))
by (*simp add*: *OmegaUnroll PiEqvRule*)

have *2*: $\vdash$ (*init w*) Π (((*f* $\wedge$ *more*) $\wedge$ *finite*);(*omega f*)) =
((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π (*omega f*))
using *PiSChopDist*[*of w LIFT* (*f* $\wedge$ *more*) *LIFT* (*omega f*)] **unfolding** *schop-d-def*
by *simp*
have *3*: $\vdash$ ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *finite*) = ((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *finite*)
by (*meson PiFiniteAbsorb Prop11*)
have *4*: $\vdash$ ((*init w*) Π (*f* $\wedge$ *more*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π (*omega f*)) =
((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π (*omega f*))
by (*simp add*: *3 LeftChopEqvChop*)
have *5*: $\vdash$ ((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *finite*);((*init w*) $\wedge$ (*init w*) Π (*omega f*)) =
(((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*) $\wedge$ *finite*);((*init w*) Π (*omega f*))
using *SlideInitSFin*[*of LIFT* (*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) *w LIFT* (*init w*) Π (*omega f*)]
unfolding *schop-d-def* **by** *blast*
have *6*: $\vdash$ (((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*) $\wedge$ *finite*) =
(((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*))
using *SFinEQvFinAndFinite* **by** *fastforce*
have *7*: $\vdash$ (*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) = ((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *more*)
by (*metis PiAnd PiAndPiImpPiAnd PiMoreAbsorb Prop10 Prop12 inteq-reflection*)
have *8*: $\vdash$ (((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*)) =
((((*init w*) Π ((*f* $\wedge$ *more*) $\wedge$ *finite*) $\wedge$ *more*) $\wedge$ *sfin w*))
using *7* **by** *auto*
show *?thesis*
by (*metis 1 2 4 5 6 8 int-eq*)
qed

lemma *PiAOmega-sfin*:
assumes $\neg$ *nfinite l*
nnth l 0 = *0*
nidx l
($\forall$ *i*. *enat* (*nnth l i*) $\leq$ *nlength* σ)
shows ((*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) $\models$ *sfin w*) =
w (*NNil* (*nnth* σ (*nnth l* (*Suc i*))))
using *assms nidx-expand*[*of l*]
by (*simp add*: *sfin-defs*)
(*metis diff-add less-imp-le-nat linorder-le-cases ndropn-nlast nfinite-ntaken nsubn-def1 ntaken-all ntaken-ndropn-nlast*)

lemma *PiAOmega-sfin-exist*:
assumes $\neg$ *nfinite l*
nnth l 0 = *0*
nidx l
($\forall$ *i*. *enat* (*nnth l i*) $\leq$ *nlength* σ)
w (*NNil* (*nnth* σ (*nnth l* (*Suc i*))))
shows ($\exists$ *x*$\in$*nset* (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))) . *w* (*NNil x*))
using *assms nidx-expand*[*of l*] *in-nset-conv-nnth*[*of -* (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))]
unfolding *nsubn-def1*
by (*metis diff-add less-imp-le-nat linorder-le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-ndropn-nlast*)

lemma *PiAOmega-help2*:
($\sigma \models (aomega\ ((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w))) =$
$(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge$
$nidx\ l \wedge (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ \ ($
$((nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \models f) \wedge$
$0 <\ nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \wedge$
$nfinite\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
$\wedge\ w\ (NNil\ (nnth\ \sigma\ (nnth\ l\ (Suc\ i))))$
$)$
$)\ )$

proof –
have *1*: $(\sigma \models (aomega\ ((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w))) =$
$(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge$
$nidx\ l \wedge (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models init\ w\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w))$
unfolding *aomega-d-def* **by** *blast*
have *2*: $(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge$
$nidx\ l \wedge (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models init\ w\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w)) =$
$(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge$
$nidx\ l \wedge (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ \ (\ (\exists\, x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ w\ (NNil\ x)) \wedge$
$(nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models (f \wedge more) \wedge finite)$
$\wedge\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models sfin\ w)$
$)$
$)\ )$
using *Pistate*[*of w LIFT* $((f \wedge more) \wedge finite)$] **by** *auto*
have *3*: $(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge$
$nidx\ l \wedge (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ \ (\ (\exists\, x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ w\ (NNil\ x)) \wedge$
$(nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models (f \wedge more) \wedge finite)$
$\wedge\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models sfin\ w)$
$)$
$)\ ) =$
$(\exists\, l.\ \neg\ nfinite\ l \wedge$
$nnth\ l\ 0 = 0 \wedge$
$nidx\ l \wedge (\forall\, i.\ enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
$(\forall\, i.\ \ (\ (\exists\, x{\in}nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ .\ w\ (NNil\ x)) \wedge$
$((nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \models f) \wedge$
$0 <\ nlength\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))) \wedge$
$nfinite\ (nfilter\ (\lambda y.\ w\ (NNil\ y))\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))$
$\wedge\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models sfin\ w)$
$)$

))
unfolding *more-defs finite-defs* **by** *simp*
have *4*: $(\exists l. \neg$ *nfinite* $l \wedge$
nnth l $0 = 0 \wedge$
nidx $l \wedge (\forall i.$ *enat* (*nnth* l i) $\leq$ *nlength* $\sigma) \wedge$
$(\forall i.$ ($(\exists x \in$*nset* (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i))) . w (*NNil* x)) $\wedge$
((*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i)))) $\models f) \wedge$
$0<$ *nlength* (*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i)))) $\wedge$
nfinite (*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i))))
$\wedge$ ((*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i))) $\models$ *sfin* w)
)
)) =
$(\exists l. \neg$ *nfinite* $l \wedge$
nnth l $0 = 0 \wedge$
nidx $l \wedge (\forall i.$ *enat* (*nnth* l i) $\leq$ *nlength* $\sigma) \wedge$
$(\forall i.$ ($(\exists x \in$*nset* (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i))) . w (*NNil* x)) $\wedge$
((*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i)))) $\models f) \wedge$
$0<$ *nlength* (*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i)))) $\wedge$
nfinite (*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i))))
$\wedge$ w (*NNil* (*nnth* σ (*nnth* l (*Suc* i))))
)
))
using *PiAOmega-sfin*[*of* - σ w] **by** *blast*
have *5*: $(\exists l. \neg$ *nfinite* $l \wedge$
nnth l $0 = 0 \wedge$
nidx $l \wedge (\forall i.$ *enat* (*nnth* l i) $\leq$ *nlength* $\sigma) \wedge$
$(\forall i.$ ($(\exists x \in$*nset* (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i))) . w (*NNil* x)) $\wedge$
((*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i)))) $\models f) \wedge$
$0<$ *nlength* (*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i)))) $\wedge$
nfinite (*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i))))
$\wedge$ w (*NNil* (*nnth* σ (*nnth* l (*Suc* i))))
)
)) =
$(\exists l. \neg$ *nfinite* $l \wedge$
nnth l $0 = 0 \wedge$
nidx $l \wedge (\forall i.$ *enat* (*nnth* l i) $\leq$ *nlength* $\sigma) \wedge$
$(\forall i.$ (
((*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i)))) $\models f) \wedge$
$0<$ *nlength* (*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i)))) $\wedge$
nfinite (*nfilter* ($\lambda y.$ w (*NNil* y)) (*nsubn* σ (*nnth* l i) (*nnth* l (*Suc* i))))
$\wedge$ w (*NNil* (*nnth* σ (*nnth* l (*Suc* i))))
)
))
using *PiAOmega-sfin-exist*[*of* - σ w] **by** *blast*
show *?thesis*
using *1 2 3 4 5* **by** *presburger*
qed

primcorec *oml* :: $('a \Rightarrow bool) \Rightarrow 'a\ nellist \Rightarrow nat\ nellist \Rightarrow nat \Rightarrow nat\ nellist$

where
oml P xs l x =
NCons (*case x of 0* ⇒ *0* |
Suc j ⇒ (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*nnth l x*) *xs*)))))
(*oml P xs l* (*Suc x*))

lemma *oml-nnth-zero*:
nnth (*oml P xs l 0*) *0* = *0*
using *oml.disc-iff oml.simps*(*2*) *nhd-conv-nnth*
by (*metis old.nat.simps*(*4*))

lemma *oml-nnth-one*:
assumes ¬ *nfinite l*
¬ *nfinite xs*
shows *nnth* (*oml P xs l 0*) *1* = (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*nnth l 1*) *xs*))))
using *assms*
oml.code oml.disc-iff oml.simps(*2*) *nhd-conv-nnth nnth-Suc-NCons*
by (*metis One-nat-def old.nat.simps*(*5*))

lemma *oml-nnth*:
assumes ¬ *nfinite l*
¬ *nfinite xs*
shows *nnth* (*oml P xs l x*) *i* =
(*if x* = *0 then*
(*if i* = *0 then 0 else* (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*nnth l i*) *xs*)))))
else (*the-enat* (*nlength*(*nfilter P* (*ntaken* (*nnth l* (*i*+*x*)) *xs*)))))
using *assms*
proof (*induct i arbitrary*: *x*)
case *0*
then show *?case*
by (*metis Nitpick.case-nat-unfold ndropn-0 ndropn-nnth nhd-conv-nnth oml.disc-iff oml.simps*(*2*))
next
case (*Suc i*)
then show *?case*
by (*metis One-nat-def Zero-not-Suc add.commute add-Suc-shift nnth-Suc-NCons oml.code plus-1-eq-Suc*)
qed

lemma *oml-infinite*:
assumes ¬ *nfinite l*
¬ *nfinite xs*
shows ¬ *nfinite* (*oml P xs l x*)
proof
assume *nfinite* (*oml P xs l x*)
thus *False*
using *assms*
proof (*induct zs*≡ (*oml P xs l x*) *arbitrary*: *x rule*: *nfinite-induct*)
case (*NNil y*)
then show *?case* **by** (*metis nellist.disc*(*1*) *oml.disc-iff*)
next

case (*NCons x nell*)
then show *?case* **by** (*metis nellist.sel*(*5*) *oml.simps*(*3*))
qed
qed

lemma *PiAOmegaImpSem*:
assumes ($\sigma \models$ (*aomega* ((*init w*) Π (($f \wedge$ *more*) $\wedge$ *finite*) $\wedge$ *sfin w*)))
shows ($\sigma \models$ (*init w*) Π (*aomega f*))
proof –
have *1*: ($\exists$ *l*. $\neg$ *nfinite l* $\wedge$
nnth l 0 = *0* $\wedge$
nidx l $\wedge$ ($\forall$ *i*. *enat* (*nnth l i*) $\leq$ *nlength* σ) $\wedge$
($\forall$ *i*. (
((*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))) $\models f$) $\wedge$
0< *nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))) $\wedge$
nfinite (*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))))
$\wedge$ *w* (*NNil* (*nnth* σ (*nnth l* (*Suc i*))))
)
))
using *assms PiAOmega-help2*[*of w f* σ] **by** *blast*
obtain *l* **where** *2*: $\neg$ *nfinite l* $\wedge$
nnth l 0 = *0* $\wedge$
nidx l $\wedge$ ($\forall$ *i*. *enat* (*nnth l i*) $\leq$ *nlength* σ) $\wedge$
($\forall$ *i*. (
((*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))) $\models f$) $\wedge$
0< *nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*)))) $\wedge$
nfinite (*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn* σ (*nnth l i*) (*nnth l* (*Suc i*))))
$\wedge$ *w* (*NNil* (*nnth* σ (*nnth l* (*Suc i*))))
)
)
using *1* **by** *auto*
have *3*: $\bigwedge$*j*. *0*<*j* $\longrightarrow$ *w* (*NNil* (*nnth* σ (*nnth l j*)))
using *2* **by** (*metis Suc-diff-1*)
have *31*: $\bigwedge$*j*. *0*<*j* $\longrightarrow$ *nfinite* (*nfilter* (λ*y*. *w* (*NNil y*)) (*ntaken* (*nnth l j*) σ))
by (*metis 2 3 nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast*)
have *32*: $\bigwedge$*j*. (*nnth l j*) < (*nnth l* (*Suc j*))
by (*metis 2 linorder-le-cases nfinite-ntaken nidx-expand ntaken-all*)
have *4*: $\bigwedge$*j*. *0*<*j* $\longrightarrow$
(*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn* σ (*nnth l j*) (*nnth l* (*Suc j*)))) =
(*nsubn* (*nfilter* (λ*y*. *w* (*NNil y*)) σ)
(*the-enat* (*nlength*(*nfilter* (λ*y*. *w* (*NNil y*)) (*ntaken* (*nnth l j*) σ))))
(*the-enat* (*nlength*(*nfilter* (λ*y*. *w* (*NNil y*)) (*ntaken* (*nnth l* (*Suc j*)) σ))))
)

using *2 3 32 nfilter-nsubn*[*of* - σ (λ*y*. *w* (*NNil y*))]
by (*metis 31 less-imp-le-nat nfinite-nlength-enat the-enat.simps zero-less-Suc*)
have *5*: (*nfilter* (λ*y*. *w* (*NNil y*)) (*nsubn* σ *0* (*nnth l* (*Suc 0*)))) =
(*nsubn* (*nfilter* (λ*y*. *w* (*NNil y*)) σ)
0

(*the-enat* (*nlength*(*nfilter* (λ*y*. *w* (*NNil y*)) (*ntaken* (*nnth l* (*Suc 0*)) σ))))
)
by (*simp add*: *2 nfilter-nsubn-zero*)
have *6*: *0*< (*the-enat* (*nlength*(*nfilter* (λ*y*. *w* (*NNil y*)) (*ntaken* (*nnth l* (*Suc 0*)) σ))))
by (*metis 2 5 gr0I i0-less nlength-NNil nsubn-def1 ntaken-0 zero-less-diff*)
have *7*: $\bigwedge$*j*. *0*<*j* $\longrightarrow$ (*the-enat* (*nlength*(*nfilter* (λ*y*. *w* (*NNil y*)) (*ntaken* (*nnth l j*) σ)))) <
(*the-enat* (*nlength*(*nfilter* (λ*y*. *w* (*NNil y*)) (*ntaken* (*nnth l* (*Suc j*)) σ))))
by (*metis 2 4 bot-nat-0.not-eq-extremum i0-less nlength-NNil nsubn-def1 ntaken-0 zero-less-diff*)
have *8*: ¬ *nfinite* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*)
using *2 infinite-nidx-imp-infinite-interval oml-infinite* **by** *blast*
have *9*: *nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*) *0* = *0*
by (*simp add*: *oml-nnth-zero*)
have *10*: $\bigwedge$*j*. *0*< *j* $\longrightarrow$ *nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*) *j* =
(*the-enat* (*nlength*(*nfilter* (λ*y*. *w* (*NNil y*)) (*ntaken* (*nnth l j*) σ))))
by (*metis 2 infinite-nidx-imp-infinite-interval less-not-refl2 oml-nnth*)
have *11*: $\bigwedge$*j* . *nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*) *j* < *nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*) (*Suc j*)
by (*metis 10 6 7 9 bot-nat-0.not-eq-extremum zero-less-Suc*)
have *12*: *nidx* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*)
using *nidx-expand*[*of* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*)]
using *11* **by** *blast*
have *13*: ($\forall$ *i*. *enat* (*nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*) *i*) ≤ *nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) σ))
by (*metis 10 3 bot-nat-0.not-eq-extremum enat-ile le-zero-eq linorder-le-cases*
nfilter-chop1-ntaken ntaken-all ntaken-nlast oml-nnth-zero the-enat.simps zero-enat-def)
have *14*: ($\forall$ *i*. *f* (*nsubn* (*nfilter* (λ*y*. *w* (*NNil y*)) σ)
(*nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*) *i*)
(*nnth* (*oml* (λ*y*. *w* (*NNil y*)) σ *l 0*) (*Suc i*))))
by (*metis 10 2 4 5 9 bot-nat-0.not-eq-extremum zero-less-Suc*)
have *15*: ($\exists$ *x*∈*nset* σ. *w* (*NNil x*))
by (*meson 2 in-nset-conv-nnth*)
have *16*: (($\exists$ *x*∈*nset* σ. *w* (*NNil x*)) ∧
($\exists$ *l*. ¬ *nfinite l* ∧
nnth l 0 = *0* ∧
nidx l ∧
($\forall$ *i*. *enat* (*nnth l i*) ≤ *nlength* (*nfilter* (λ*y*. *w* (*NNil y*)) σ)) ∧
($\forall$ *i*. *f* (*nsubn* (*nfilter* (λ*y*. *w* (*NNil y*)) σ) (*nnth l i*) (*nnth l* (*Suc i*))))))
using *12 13 14 15 8 oml-nnth-zero* **by** *blast*
have *17*: (σ ⊨ (*init w*) Π (*aomega f*))
using *16*
using *Pistate*[*of w LIFT aomega f* σ] **unfolding** *aomega-d-def* **by** *blast*
show *?thesis* **using** *17* **by** *auto*
qed

lemma *PiAOmegaImp*:
⊢ (*aomega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*)) $\longrightarrow$ (*init w*) Π (*aomega f*)
unfolding *Valid-def* **using** *PiAOmegaImpSem*
using *unl-lift2* **by** *blast*

lemma *PiOmegaImpSem*:
assumes (σ ⊨ (*omega* ((*init w*) Π ((*f* ∧ *more*) ∧ *finite*) ∧ *sfin w*)))

shows $(\sigma \models (init\ w)\ \Pi\ (omega\ f))$
by (*metis OmegaEqvAOmega PiAOmegaImpSem assms inteq-reflection*)

lemma *PiOmegaImp*:
$\vdash (omega\ ((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w)) \longrightarrow (init\ w)\ \Pi\ (omega\ f)$
unfolding *Valid-def* **using** *PiOmegaImpSem* **using** *unl-lift2* **by** *blast*

lemma *PiOmega*:
$\vdash (init\ w)\ \Pi\ (omega\ f) = (omega\ ((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w))$
proof –
have *0*: $\vdash (((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge more) \wedge sfin\ w) =$
$((((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w) \wedge more) \wedge finite)$
using *SFinEQvFinAndFinite* **by** *fastforce*
have *1*: $\vdash (init\ w)\ \Pi\ (omega\ f) \longrightarrow inf$
by (*metis AndChopB AndMoreAndFiniteEqvAndFmore OmegaMoreEqvInf OmegaWeakCoinduct 0 PiStateOmegaUnroll fmore-d-def int-eq int-simps*(*20*))
have *2*: $\vdash (init\ w)\ \Pi\ (omega\ f) \longrightarrow$
$inf \wedge (((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w) \wedge fmore);((init\ w)\ \Pi\ (omega\ f))$
by (*metis 1 AndMoreSChopEqvAndFmoreChop 0 PiStateOmegaUnroll Prop12 int-eq int-iffD1 schop-d-def*)
have *3*: $\vdash (init\ w)\ \Pi\ (omega\ f) \longrightarrow (omega\ ((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w))$
using *OmegaIntro* **by** (*metis 2 Prop12*)
have *4*: $\vdash (omega\ ((init\ w)\ \Pi\ ((f \wedge more) \wedge finite) \wedge sfin\ w)) \longrightarrow (init\ w)\ \Pi\ (omega\ f)$
by (*simp add*: *PiOmegaImp*)
show *?thesis*
by (*simp add*: *3 4 int-iffI*)
qed

end

14 First Order Finite and Infinite ITL theorems

theory *FOTheorems*
imports
SChopTheorems Chopstar
begin

We give the proofs of a list of first order (in)finite ITL theorems.

lemma *EExI-unl*:
$w \models f\ x \Longrightarrow w \models (\exists\exists\ x.\ f\ x)$
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*: $\vdash \neg(f\ x) \lor (\exists\exists\ x.\ g\ x)$ **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*: $\bigwedge x. \vdash f\,x \longrightarrow (g \longrightarrow (\exists\exists\ x.\ g \wedge f\,x))$
using *EExI*[*of* $\lambda x.$ *LIFT*$(g \wedge f\,x)$] **by** *fastforce*
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 $\vdash (\forall\forall\ x.\ f\ x) = (\forall\forall\ x.\ g\ x)$
unfolding *forall-state-d-def* **using** *assms EExEqvRule*[*of* $\lambda x.\ (LIFT(\neg\ f\ x))\ \lambda x.\ (LIFT(\neg g\ x))$]
by *fastforce*

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*: $\bigwedge x . \vdash (\neg(f\ x) \vee \neg(g\ x)) = (\neg((f\ x) \wedge (g\ x)))$
by *auto*
have *2*: $\vdash (\exists\exists\ x.\ \neg(f\ x) \vee \neg(g\ x)) = (\exists\exists\ x.\ \neg((f\ x) \wedge (g\ x)))$
using *1* **by** (*simp add*: *EExEqvRule*)
have *3*: $\vdash(\exists\exists\ fa.\ \neg\ (f\ fa \wedge g\ fa)) = (\exists\exists\ fa.\ \neg\ f\ fa \vee \neg\ g\ fa)$
using *2* **by** *auto*
have *4*: $\vdash(\neg\ (\neg\ (\exists\exists\ fa.\ \neg\ f\ fa) \wedge \neg\ (\exists\exists\ f.\ \neg\ g\ f))) = ((\exists\exists\ fa.\ \neg\ f\ fa) \vee (\exists\exists\ f.\ \neg\ g\ f))$
by *auto*
have $\vdash((\exists\exists\ fa.\ \neg\ f\ fa) \vee (\exists\exists\ f.\ \neg\ g\ f)) = (\exists\exists\ fa.\ \neg\ f\ fa \vee \neg\ g\ fa)$
by (*simp add*: *EExOrDist inteq-reflection*)
then have *5*: $\vdash(\neg\ (\exists\exists\ fa.\ \neg\ f\ fa) \wedge \neg\ (\exists\exists\ f.\ \neg\ g\ f)) = (\neg\ (\exists\exists\ fa.\ \neg\ f\ fa \vee \neg\ g\ fa))$
by *auto*
show *?thesis* **using** *3 5* **unfolding** *forall-state-d-def* **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*: $\vdash (\neg(g \wedge (\forall\forall\ x.\ f\ x))) = (\neg(\forall\forall\ x.\ g \wedge f\ x))$
by (*metis 1 3 5 6 inteq-reflection*)
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*: $\vdash (\neg(g \vee (\forall\forall\ x.\ f\ x))) = (\neg(\forall\forall\ x.\ g \vee f\ x))$ **by** (*metis 1 3 5 6 inteq-reflection*)
from *7* **show** *?thesis* **by** *auto*
qed

lemma *EExImpChopRule*:
assumes $\vdash f\ x \longrightarrow g\ x$
shows $\vdash (\exists\exists\ x.\ h;(f\ x) \longrightarrow h;(g\ x))$
using *RightChopImpChop*[*of f x g x h*]
EExImpRule[*of* $\lambda x.\ LIFT(h;(f\ x))\ x\ \lambda x.\ LIFT(h;(g\ x))$] *assms* **by** *auto*

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

lemma *EExChopRightNoDep*:
$\vdash (\exists\exists\ x.\ (f\ x);g) = (\exists\exists\ x.\ (f\ x));g$
by (*auto simp add*: *exist-state-d-def Valid-def itl-defs*)

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

lemma *EExChopLeftNoDep*:
$\vdash (\exists\exists\ x.\ g;(f\ x)\ ) = g;(\exists\exists\ x.\ f\ x)$
by (*auto simp add*: *exist-state-d-def Valid-def itl-defs*)

lemma *EExEExChopEqvEExEExChop*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (f\ v);(g\ y)\ )) = (\exists\exists\ y.\ (\exists\exists\ v.\ (f\ v);(g\ y)\ ))$
by (*simp add*: *exist-state-d-def Valid-def itl-defs*) *blast*

lemma *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 itl-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 itl-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 (*simp add*: *exist-state-d-def Valid-def itl-defs*) *blast*

lemma *ExLenOrInf*:
$\vdash (\exists\ n.\ len(n)) \vee inf$
using *nfinite-nlength-enat* **by** (*auto simp add*: *Valid-def len-defs itl-defs*)

lemma *CSPowerChop*:
$\vdash (f^\star) = (\exists\, n.\ fpower\ (f \wedge more)\ n);(empty \vee (f \wedge more) \wedge inf)$
by (*simp add*: *chopstar-d-def fpowerstar-d-def powerstar-d-def Valid-def*)

lemma *ExChopRightNoDep*:
$\vdash (\exists\ x.\ (f\ x);g) = (\exists\ x.\ (f\ x));g$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *ExChopLeftNoDep*:
$\vdash (\exists\ x.\ g;(f\ x)\ ) = g;(\exists\ x.\ f\ x)$
by (*auto simp add*: *Valid-def itl-defs*)

lemma *ExExEqvExEx*:
$\vdash (\exists\ x.\ (\exists\ y.\ (f\ x);(g\ y))) = (\exists\ y.\ (\exists\ x.\ (f\ x);(g\ y)))$
by (*auto simp add*: *Valid-def itl-defs*)

end

References

[1] A. Armstrong, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2013. http://isa-afp.org/entries/Kleene_Algebra.shtml, Formal proof development.

[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] A. Lochbihler. Coinductive. *Archive of Formal Proofs*, feb 2010. https://www.isa-afp.org/entries/Coinductive.html, Formal proof development.

[4] S. Merz. An Encoding of TLA in Isabelle. http://www.pst.informatik.uni-muenchen.de/~merz/isabelle/. Part of the Isabelle distribution., 1998.

[5] B. Moszkowski. 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.

[6] 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.

[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.