Partial Order Reduction

Julian Brunner

September 13, 2023

Abstract

This entry provides a formalization of the abstract theory of ample set partial order reduction as presented in [2, 1]. The formalization includes transition systems with actions, trace theory, as well as basics on finite, infinite, and lazy sequences. We also provide a basic framework for static analysis on concurrent systems with respect to the ample set condition.

Contents

1 List Prefixes

theory *List-Prefixes*
imports *HOL−Library.Prefix-Order*
begin

lemmas [*intro*] = *prefixI strict-prefixI*[*folded less-eq-list-def*]
lemmas [*elim*] = *prefixE strict-prefixE*[*folded less-eq-list-def*]

lemmas [*intro?*] = *take-is-prefix*[*folded less-eq-list-def*]

hide-const (**open**) *Sublist.prefix Sublist.suffix*

lemma *prefix-finI-item*[*intro*!]:
  assumes $a = b$ $u \le v$
  shows $a \# u \le b \# v$
  using *assms* **by** *force*
lemma *prefix-finE-item*[*elim*!]:
  assumes $a \# u \le b \# v$
  obtains $a = b$ $u \le v$
  using *assms* **by** *force*

lemma *prefix-fin-append*[*intro*]: $u \le u$ @ v **by** *auto*
lemma *pprefix-fin-length*[*dest*]:
  assumes $u < v$
  shows *length* $u <$ *length* v

```
  proof −
    obtain a w where 1: v = u @ a # w using assms by rule
    show ?thesis unfolding 1 by simp
  qed

end
```

2 Lists

```
theory List-Extensions
imports HOL−Library.Sublist
begin

  declare remove1-idem[simp]

  lemma nth-append-simps[simp]:
    i < length xs ⟹ (xs @ ys) ! i = xs ! i
    i ≥ length xs ⟹ (xs @ ys) ! i = ys ! (i − length xs)
    unfolding nth-append by simp+

  notation zip (infixr || 51)

  abbreviation project A ≡ filter (λ a. a ∈ A)
  abbreviation select s w ≡ nths w s

  lemma map-plus[simp]: map (plus n) [i ..< j] = [i + n ..< j + n]
  proof (induct n)
    case 0
    show ?case by simp
  next
    case (Suc n)
    have map (plus (Suc n)) [i ..< j] = map (Suc ∘ plus n) [i ..< j] by simp
    also have ... = (map Suc ∘ map (plus n)) [i ..< j] by simp
    also have ... = map Suc (map (plus n) [i ..< j]) by simp
    also have ... = map Suc [i + n ..< j + n] unfolding Suc by simp
   also have ... = [Suc (i + n) ..< Suc (j + n)] unfolding map-Suc-upt by simp
    also have ... = [i + Suc n ..< j + Suc n] by simp
    finally show ?case by this
  qed

  lemma singleton-list-lengthE[elim]:
    assumes length xs = 1
    obtains x
    where xs = [x]
  proof −
    have 0: length xs = Suc 0 using assms by simp
    obtain y ys where 1: xs = y # ys length ys = 0 using 0 Suc-length-conv by
metis
    show ?thesis using that 1 by blast
```

qed

lemma *singleton-hd-last*: *length xs* = *1* $\Longrightarrow$ *hd xs* = *last xs* **by** *fastforce*

lemma *set-subsetI*[*intro*]:
assumes $\bigwedge$ *i*. *i* < *length xs* $\Longrightarrow$ *xs* ! *i* $\in$ *S*
shows *set xs* $\subseteq$ *S*
proof
fix *x*
assume *0*: *x* $\in$ *set xs*
obtain *i* **where** *1*: *i* < *length xs x* = *xs* ! *i* **using** *0* **unfolding** *in-set-conv-nth*
by *blast*
show *x* $\in$ *S* **using** *assms*(*1*) *1* **by** *auto*
qed

lemma *hd-take*[*simp*]:
assumes *n* $\neq$ *0 xs* $\neq$ []
shows *hd* (*take n xs*) = *hd xs*
proof −
have *1*: *take n xs* $\neq$ [] **using** *assms* **by** *simp*
have *2*: *0* < *n* **using** *assms* **by** *simp*
have *hd* (*take n xs*) = *take n xs* ! *0* **using** *hd-conv-nth*[*OF 1*] **by** *this*
also have . . . = *xs* ! *0* **using** *nth-take*[*OF 2*] **by** *this*
also have . . . = *hd xs* **using** *hd-conv-nth*[*OF assms*(*2*)] **by** *simp*
finally show *?thesis* **by** *this*
qed
lemma *hd-drop*[*simp*]:
assumes *n* < *length xs*
shows *hd* (*drop n xs*) = *xs* ! *n*
using *hd-drop-conv-nth assms* **by** *this*
lemma *last-take*[*simp*]:
assumes *n* < *length xs*
shows *last* (*take* (*Suc n*) *xs*) = *xs* ! *n*
using *assms*
proof (*induct xs arbitrary*: *n*)
case (*Nil*)
show *?case* **using** *Nil* **by** *simp*
next
case (*Cons x xs*)
show *?case* **using** *Cons* **by** (*auto*) (*metis Suc-less-eq Suc-pred*)
qed

lemma *split-list-first-unique*:
assumes u_1 @ [*a*] @ u_2 = v_1 @ [*a*] @ v_2 *a* $\notin$ *set* u_1 *a* $\notin$ *set* v_1
shows u_1 = v_1
proof −
obtain *w* **where** u_1 = v_1 @ *w* $\wedge$ *w* @ [*a*] @ u_2 = [*a*] @ v_2 $\vee$
u_1 @ *w* = v_1 $\wedge$ [*a*] @ u_2 = *w* @ [*a*] @ v_2 **using** *assms*(*1*) *append-eq-append-conv2*
by *blast*

thus *?thesis* **using** *assms*(*2*, *3*) **by** (*auto*) (*metis hd-append2 list.sel*(*1*) *list.set-sel*(*1*))+
qed

end

3 Finite Prefixes of Infinite Sequences

theory *Word-Prefixes*
imports
List-Prefixes
../Extensions/List-Extensions
Transition-Systems-and-Automata.Sequence
begin

definition *prefix-fininf* :: *'a list* $\Rightarrow$ *'a stream* $\Rightarrow$ *bool* (**infix** $\leq_{FI}$ *50*)
where $u \leq_{FI} v \equiv \exists\ w.\ u$ @− $w = v$

lemma *prefix-fininfI*[*intro*]:
assumes u @− $w = v$
shows $u \leq_{FI} v$
using *assms* **unfolding** *prefix-fininf-def* **by** *auto*
lemma *prefix-fininfE*[*elim*]:
assumes $u \leq_{FI} v$
obtains w
where $v = u$ @− w
using *assms* **unfolding** *prefix-fininf-def* **by** *auto*

lemma *prefix-fininfI-empty*[*intro*!]: [] $\leq_{FI}$ w **by** *force*
lemma *prefix-fininfI-item*[*intro*!]:
assumes $a = b$ $u \leq_{FI} v$
shows a # $u \leq_{FI} b$ ## v
using *assms* **by** *force*
lemma *prefix-fininfE-item*[*elim*!]:
assumes a # $u \leq_{FI} b$ ## v
obtains $a = b$ $u \leq_{FI} v$
using *assms* **by** *force*

lemma *prefix-fininf-item*[*simp*]: a # $u \leq_{FI} a$ ## $v \longleftrightarrow u \leq_{FI} v$ **by** *force*
lemma *prefix-fininf-list*[*simp*]: w @ $u \leq_{FI} w$ @− $v \longleftrightarrow u \leq_{FI} v$ **by** (*induct w*, *auto*)
lemma *prefix-fininf-conc*[*intro*]: $u \leq_{FI} u$ @− v **by** *auto*
lemma *prefix-fininf-prefix*[*intro*]: *stake k w* $\leq_{FI}$ *w* **using** *stake-sdrop* **by** *blast*
lemma *prefix-fininf-set-range*[*dest*]: $u \leq_{FI} v \Longrightarrow$ *set u* $\subseteq$ *sset v* **by** *auto*

lemma *prefix-fininf-absorb*:
assumes $u \leq_{FI} v$ @− w *length u* $\leq$ *length v*
shows $u \leq v$
proof −
obtain x **where** *1*: u @− $x = v$ @− w **using** *assms*(*1*) **by** *auto*

have $u \leq u$ @ *stake* (*length* v − *length* u) x **by** *rule*
also have ... = *stake* (*length* v) (u @− x) **using** *assms*(*2*) **by** (*simp add*: *stake-shift*)
also have ... = *stake* (*length* v) (v @− w) **unfolding** *1* **by** *rule*
also have ... = v **using** *eq-shift* **by** *blast*
finally show *?thesis* **by** *this*
qed
lemma *prefix-fininf-extend*:
assumes $u \leq_{FI} v$ @− w *length* $v \leq$ *length* u
shows $v \leq u$
proof −
obtain x **where** *1*: u @− x = v @− w **using** *assms*(*1*) **by** *auto*
have $v \leq v$ @ *stake* (*length* u − *length* v) w **by** *rule*
also have ... = *stake* (*length* u) (v @− w) **using** *assms*(*2*) **by** (*simp add*: *stake-shift*)
also have ... = *stake* (*length* u) (u @− x) **unfolding** *1* **by** *rule*
also have ... = u **using** *eq-shift* **by** *blast*
finally show *?thesis* **by** *this*
qed
lemma *prefix-fininf-length*:
assumes $u \leq_{FI} w$ $v \leq_{FI} w$ *length* $u \leq$ *length* v
shows $u \leq v$
proof −
obtain u' v' **where** *1*: $w = u$ @− u' $w = v$ @− v' **using** *assms*(*1*, *2*) **by** *blast+*
have u = *stake* (*length* u) (u @− u') **using** *shift-eq* **by** *blast*
also have ... = *stake* (*length* u) w **unfolding** *1*(*1*) **by** *rule*
also have ... = *stake* (*length* u) (v @− v') **unfolding** *1*(*2*) **by** *rule*
also have ... = *take* (*length* u) v **using** *assms* **by** (*simp add*: *min.absorb2* *stake-append*)
also have ... $\leq v$ **by** *rule*
finally show *?thesis* **by** *this*
qed

lemma *prefix-fininf-append*:
assumes $u \leq_{FI} v$ @− w
obtains (*absorb*) $u \leq v$ | (*extend*) z **where** $u = v$ @ z $z \leq_{FI} w$
proof (*cases length u length v rule*: *le-cases*)
case *le*
obtain x **where** *1*: u @− x = v @− w **using** *assms*(*1*) **by** *auto*
show *?thesis*
proof (*rule absorb*)
have $u \leq u$ @ *stake* (*length* v − *length* u) x **by** *rule*
also have ... = *stake* (*length* v) (u @− x) **using** *le* **by** (*simp add*: *stake-shift*)
also have ... = *stake* (*length* v) (v @− w) **unfolding** *1* **by** *rule*
also have ... = v **using** *eq-shift* **by** *blast*
finally show $u \leq v$ **by** *this*
qed
next

case *ge*
obtain *x* **where** *1*: u @− $x = v$ @− w **using** *assms*(*1*) **by** *auto*
show *?thesis*
proof (*rule extend*)
have u = *stake* (*length u*) (*u* @− *x*) **using** *shift-eq* **by** *auto*
also have . . . = *stake* (*length u*) (*v* @− *w*) **unfolding** *1* **by** *rule*
also have . . . = *v* @ *stake* (*length u* − *length v*) *w* **using** *ge* **by** (*simp add*: *stake-shift*)
finally show *u* = *v* @ *stake* (*length u* − *length v*) *w* **by** *this*
show *stake* (*length u* − *length v*) $w \leq_{FI} w$ **by** *rule*
qed
qed

lemma *prefix-fin-prefix-fininf-trans*[*trans*, *intro*]: $u \leq v \Longrightarrow v \leq_{FI} w \Longrightarrow u \leq_{FI} w$
by (*metis Prefix-Order.prefixE prefix-fininf-def shift-append*)

lemma *prefix-finE-nth*:
assumes $u \leq v$ $i <$ *length u*
shows *u* ! *i* = *v* ! *i*
proof −
obtain *w* **where** *1*: *v* = *u* @ *w* **using** *assms*(*1*) **by** *auto*
show *?thesis* **unfolding** *1* **using** *assms*(*2*) **by** (*simp add*: *nth-append*)
qed
lemma *prefix-fininfI-nth*:
assumes $\bigwedge$ *i*. *i* < *length u* $\Longrightarrow$ *u* ! *i* = *w* !! *i*
shows $u \leq_{FI} w$
proof (*rule prefix-fininfI*)
show *u* @− *sdrop* (*length u*) *w* = *w* **by** (*simp add*: *assms list-eq-iff-nth-eq shift-eq*)
qed

definition *chain* :: (*nat* ⇒ *'a list*) ⇒ *bool*
where *chain w* ≡ *mono w* ∧ (∀ *k*. ∃ *l*. *k* < *length* (*w l*))
definition *limit* :: (*nat* ⇒ *'a list*) ⇒ *'a stream*
where *limit w* ≡ *smap* (λ *k*. *w* (*SOME l*. *k* < *length* (*w l*)) ! *k*) *nats*

lemma *chainI*[*intro?*]:
assumes *mono w*
assumes $\bigwedge$ *k*. ∃ *l*. *k* < *length* (*w l*)
shows *chain w*
using *assms* **unfolding** *chain-def* **by** *auto*
lemma *chainD-mono*[*dest?*]:
assumes *chain w*
shows *mono w*
using *assms* **unfolding** *chain-def* **by** *auto*
lemma *chainE-length*[*elim?*]:
assumes *chain w*
obtains *l*

where $k < \mathit{length}\ (w\ l)$
using *assms* **unfolding** *chain-def* **by** *auto*

lemma *chain-prefix-limit*:
assumes *chain w*
shows $w\ k \leq_{FI} \mathit{limit}\ w$
proof (*rule prefix-fininfI-nth*)
fix i
assume *1*: $i < \mathit{length}\ (w\ k)$
have *2*: $\mathit{mono}\ w\ \bigwedge k.\ \exists\ l.\ k < \mathit{length}\ (w\ l)$ **using** *chainD-mono chainE-length assms* **by** *blast+*
have *3*: $i < \mathit{length}\ (w\ (\mathit{SOME}\ l.\ i < \mathit{length}\ (w\ l)))$ **using** *someI-ex 2*(*2*) **by** *this*
show $w\ k\ !\ i = \mathit{limit}\ w\ !!\ i$
proof (*cases k SOME l. i < length* (*w l*) *rule*: *le-cases*)
case (*le*)
have *4*: $w\ k \leq w\ (\mathit{SOME}\ l.\ i < \mathit{length}\ (w\ l))$ **using** *monoD 2*(*1*) *le* **by** *this*
show *?thesis* **unfolding** *limit-def* **using** *prefix-finE-nth 4 1* **by** *auto*
next
case (*ge*)
have *4*: $w\ (\mathit{SOME}\ l.\ i < \mathit{length}\ (w\ l)) \leq w\ k$ **using** *monoD 2*(*1*) *ge* **by** *this*
show *?thesis* **unfolding** *limit-def* **using** *prefix-finE-nth 4 3* **by** *auto*
qed
qed

lemma *chain-construct-1*:
assumes $P\ 0\ x_0\ \bigwedge k\ x.\ P\ k\ x \Longrightarrow \exists\ x'.\ P\ (\mathit{Suc}\ k)\ x' \wedge f\ x \leq f\ x'$
assumes $\bigwedge k\ x.\ P\ k\ x \Longrightarrow k \leq \mathit{length}\ (f\ x)$
obtains Q
where $\bigwedge k.\ P\ k\ (Q\ k)\ \mathit{chain}\ (f \circ Q)$
proof −
obtain x' **where** *1*:
$P\ 0\ x_0\ \bigwedge k\ x.\ P\ k\ x \Longrightarrow P\ (\mathit{Suc}\ k)\ (x'\ k\ x) \wedge f\ x \leq f\ (x'\ k\ x)$
using *assms*(*1*, *2*) **by** *metis*
define Q **where** $Q \equiv \mathit{rec\text{-}nat}\ x_0\ x'$
have [*simp*]: $Q\ 0 = x_0\ \bigwedge k.\ Q\ (\mathit{Suc}\ k) = x'\ k\ (Q\ k)$ **unfolding** *Q-def* **by** *simp+*
have *2*: $\bigwedge k.\ P\ k\ (Q\ k)$
proof −
fix k
show $P\ k\ (Q\ k)$ **using** *1* **by** (*induct k*, *auto*)
qed
show *?thesis*
proof (*intro that chainI monoI*, *unfold comp-apply*)
fix k
show $P\ k\ (Q\ k)$ **using** *2* **by** *this*
next
fix $x\ y :: \mathit{nat}$
assume $x \leq y$

thus $f\ (Q\ x) \leq f\ (Q\ y)$
proof (*induct* $y - x$ *arbitrary*: $x\ y$)
case *0*
show *?case* **using** *0* **by** *simp*
next
case (*Suc* k)
have $f\ (Q\ x) \leq f\ (Q\ (Suc\ x))$ **using** *1*(*2*) *2* **by** *auto*
also have $\ldots \leq f\ (Q\ y)$ **using** *Suc*(*2*) **by** (*intro Suc*(*1*), *auto*)
finally show *?case* **by** *this*
qed
next
fix k
have *3*: $P\ (Suc\ k)\ (Q\ (Suc\ k))$ **using** *2* **by** *this*
have *4*: $Suc\ k \leq length\ (f\ (Q\ (Suc\ k)))$ **using** *assms*(*3*) *3* **by** *this*
have *5*: $k < length\ (f\ (Q\ (Suc\ k)))$ **using** *4* **by** *auto*
show $\exists\ l.\ k < length\ (f\ (Q\ l))$ **using** *5* **by** *blast*
qed
qed
lemma *chain-construct-2*:
assumes $P\ 0\ x_0\ \bigwedge k\ x.\ P\ k\ x \Longrightarrow \exists\ x'.\ P\ (Suc\ k)\ x' \wedge f\ x \leq f\ x' \wedge g\ x \leq g\ x'$
assumes $\bigwedge k\ x.\ P\ k\ x \Longrightarrow k \leq length\ (f\ x)\ \bigwedge k\ x.\ P\ k\ x \Longrightarrow k \leq length\ (g\ x)$
obtains Q
where $\bigwedge k.\ P\ k\ (Q\ k)$ *chain* $(f \circ Q)$ *chain* $(g \circ Q)$
proof −
obtain x' **where** *1*:
$P\ 0\ x_0\ \bigwedge k\ x.\ P\ k\ x \Longrightarrow P\ (Suc\ k)\ (x'\ k\ x) \wedge f\ x \leq f\ (x'\ k\ x) \wedge g\ x \leq g\ (x'\ k\ x)$
using *assms*(*1*, *2*) **by** *metis*
define Q **where** $Q \equiv rec\text{-}nat\ x_0\ x'$
have [*simp*]: $Q\ 0 = x_0\ \bigwedge k.\ Q\ (Suc\ k) = x'\ k\ (Q\ k)$ **unfolding** *Q-def* **by** *simp+*
have *2*: $\bigwedge k.\ P\ k\ (Q\ k)$
proof −
fix k
show $P\ k\ (Q\ k)$ **using** *1* **by** (*induct* k, *auto*)
qed
show *?thesis*
proof (*intro that chainI monoI*, *unfold comp-apply*)
fix k
show $P\ k\ (Q\ k)$ **using** *2* **by** *this*
next
fix $x\ y :: nat$
assume $x \leq y$
thus $f\ (Q\ x) \leq f\ (Q\ y)$
proof (*induct* $y - x$ *arbitrary*: $x\ y$)
case *0*
show *?case* **using** *0* **by** *simp*
next
case (*Suc* k)

have $f\ (Q\ x) \leq f\ (Q\ (Suc\ x))$ **using** *1*(*2*) *2* **by** *auto*
also have $\ldots \leq f\ (Q\ y)$ **using** *Suc*(*2*) **by** (*intro Suc*(*1*), *auto*)
finally show *?case* **by** *this*
qed
next
fix k
have *3*: $P\ (Suc\ k)\ (Q\ (Suc\ k))$ **using** *2* **by** *this*
have *4*: $Suc\ k \leq length\ (f\ (Q\ (Suc\ k)))$ **using** *assms*(*3*) *3* **by** *this*
have *5*: $k < length\ (f\ (Q\ (Suc\ k)))$ **using** *4* **by** *auto*
show $\exists\ l.\ k < length\ (f\ (Q\ l))$ **using** *5* **by** *blast*
next
fix $x\ y :: nat$
assume $x \leq y$
thus $g\ (Q\ x) \leq g\ (Q\ y)$
proof (*induct* $y - x$ *arbitrary*: $x\ y$)
case *0*
show *?case* **using** *0* **by** *simp*
next
case (*Suc k*)
have $g\ (Q\ x) \leq g\ (Q\ (Suc\ x))$ **using** *1*(*2*) *2* **by** *auto*
also have $\ldots \leq g\ (Q\ y)$ **using** *Suc*(*2*) **by** (*intro Suc*(*1*), *auto*)
finally show *?case* **by** *this*
qed
next
fix k
have *3*: $P\ (Suc\ k)\ (Q\ (Suc\ k))$ **using** *2* **by** *this*
have *4*: $Suc\ k \leq length\ (g\ (Q\ (Suc\ k)))$ **using** *assms*(*4*) *3* **by** *this*
have *5*: $k < length\ (g\ (Q\ (Suc\ k)))$ **using** *4* **by** *auto*
show $\exists\ l.\ k < length\ (g\ (Q\ l))$ **using** *5* **by** *blast*
qed
qed
lemma *chain-construct-2'*:
assumes $P\ 0\ u_0\ v_0\ \bigwedge\ k\ u\ v.\ P\ k\ u\ v \Longrightarrow \exists\ u'\ v'.\ P\ (Suc\ k)\ u'\ v' \wedge u \leq u' \wedge v \leq v'$
assumes $\bigwedge\ k\ u\ v.\ P\ k\ u\ v \Longrightarrow k \leq length\ u\ \bigwedge\ k\ u\ v.\ P\ k\ u\ v \Longrightarrow k \leq length\ v$
obtains $u\ v$
where $\bigwedge\ k.\ P\ k\ (u\ k)\ (v\ k)$ *chain u chain v*
proof −
obtain Q **where** *1*: $\bigwedge\ k.\ (case\text{-}prod \circ P)\ k\ (Q\ k)$ *chain* $(fst \circ Q)$ *chain* $(snd \circ Q)$
proof (*rule chain-construct-2*)
show $\exists\ x'.\ (case\text{-}prod \circ P)\ (Suc\ k)\ x' \wedge fst\ x \leq fst\ x' \wedge snd\ x \leq snd\ x'$
if $(case\text{-}prod \circ P)\ k\ x$ **for** $k\ x$ **using** *assms that* **by** *auto*
show $(case\text{-}prod \circ P)\ 0\ (u_0, v_0)$ **using** *assms* **by** *auto*
show $k \leq length\ (fst\ x)$ **if** $(case\text{-}prod \circ P)\ k\ x$ **for** $k\ x$ **using** *assms that* **by** *auto*
show $k \leq length\ (snd\ x)$ **if** $(case\text{-}prod \circ P)\ k\ x$ **for** $k\ x$ **using** *assms that* **by** *auto*
qed *rule*

show *?thesis*
proof
show $P\ k\ ((fst \circ Q)\ k)\ ((snd \circ Q)\ k)$ **for** k **using** *1*(*1*) **by** (*auto simp*: *prod.case-eq-if*)
show *chain* $(fst \circ Q)$ *chain* $(snd \circ Q)$ **using** *1*(*2*, *3*) **by** *this*
qed
qed

end

4 Sets

theory *Set-Extensions*
imports
HOL−Library.Infinite-Set
begin

declare *finite-subset*[*intro*]

lemma *set-not-emptyI*[*intro 0*]: $x \in S \Longrightarrow S \neq \{\}$ **by** *auto*
lemma *sets-empty-iffI*[*intro 0*]:
assumes $\bigwedge a.\ a \in A \Longrightarrow \exists\ b.\ b \in B$
assumes $\bigwedge b.\ b \in B \Longrightarrow \exists\ a.\ a \in A$
shows $A = \{\} \longleftrightarrow B = \{\}$
using *assms* **by** *auto*
lemma *disjointI*[*intro 0*]:
assumes $\bigwedge x.\ x \in A \Longrightarrow x \in B \Longrightarrow False$
shows $A \cap B = \{\}$
using *assms* **by** *auto*
lemma *range-subsetI*[*intro 0*]:
assumes $\bigwedge x.\ f\,x \in S$
shows *range* $f \subseteq S$
using *assms* **by** *blast*

lemma *finite-imageI-range*:
assumes *finite* (*range* f)
shows *finite* (f ‘ A)
using *finite-subset image-mono subset-UNIV assms* **by** *metis*

lemma *inf-img-fin-domE'*:
assumes *infinite* A
assumes *finite* (f ‘ A)
obtains y
where $y \in f$ ‘ A *infinite* $(A \cap f$ −‘ $\{y\})$
proof (*rule ccontr*)
assume *1*: ¬ *thesis*
have *2*: *finite* $(\bigcup\ y \in f$ ‘ $A.\ A \cap f$ −‘ $\{y\})$
proof (*rule finite-UN-I*)
show *finite* (f ‘ A) **using** *assms*(*2*) **by** *this*

show $\bigwedge y.\ y \in f\ `\ A \Longrightarrow$ *finite* $(A \cap f\ -`\ \{y\})$ **using** *that 1* **by** *blast*
qed
have *3*: $A \subseteq (\bigcup\ y \in f\ `\ A.\ A \cap f\ -`\ \{y\})$ **by** *blast*
show *False* **using** *assms*(*1*) *2 3* **by** *blast*
qed

lemma *vimage-singleton*[*simp*]: $f\ -`\ \{y\} = \{x.\ f\ x = y\}$ **unfolding** *vimage-def* **by** *simp*

lemma *these-alt-def*: *Option.these S* $=$ *Some* $-`$ *S* **unfolding** *Option.these-def* **by** *force*
lemma *the-vimage-subset*: *the* $-`\ \{a\} \subseteq \{None,\ Some\ a\}$ **by** *auto*

lemma *finite-induct-reverse*[*consumes 1*, *case-names remove*]:
assumes *finite S*
assumes $\bigwedge S.$ *finite S* $\Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow P\ (S - \{x\})) \Longrightarrow P\ S$
shows *P S*
using *assms*(*1*)
proof (*induct rule*: *finite-psubset-induct*)
case (*psubset S*)
show *?case*
proof (*rule assms*(*2*))
show *finite S* **using** *psubset*(*1*) **by** *this*
next
fix *x*
assume *0*: $x \in S$
show $P\ (S - \{x\})$
proof (*rule psubset*(*2*))
show $S - \{x\} \subset S$ **using** *0* **by** *auto*
qed
qed
qed

lemma *zero-not-in-Suc-image*[*simp*]: $0 \notin Suc\ `\ A$ **by** *auto*

lemma *Collect-split-Suc*:
$\neg\ P\ 0 \Longrightarrow \{i.\ P\ i\} = Suc\ `\ \{i.\ P\ (Suc\ i)\}$
$P\ 0 \Longrightarrow \{i.\ P\ i\} = \{0\} \cup Suc\ `\ \{i.\ P\ (Suc\ i)\}$
proof $-$
assume $\neg\ P\ 0$
thus $\{i.\ P\ i\} = Suc\ `\ \{i.\ P\ (Suc\ i)\}$
by (*auto*, *metis image-eqI mem-Collect-eq nat.exhaust*)
next
assume *P 0*
thus $\{i.\ P\ i\} = \{0\} \cup Suc\ `\ \{i.\ P\ (Suc\ i)\}$
by (*auto*, *metis imageI mem-Collect-eq not0-implies-Suc*)
qed

lemma *Collect-subsume*[*simp*]:

assumes $\bigwedge x.\ x \in A \Longrightarrow P\ x$
shows $\{x \in A.\ P\ x\} = A$
using *assms* **unfolding** *simp-implies-def* **by** *auto*

lemma *Max-ge'*:
assumes *finite* A $A \neq \{\}$
assumes $b \in A$ $a \leq b$
shows $a \leq Max\ A$
using *assms Max-ge-iff* **by** *auto*

abbreviation *least* $A \equiv LEAST\ k.\ k \in A$

lemma *least-contains*[*intro?*, *simp*]:
fixes A :: $'a$ :: *wellorder set*
assumes $k \in A$
shows *least* $A \in A$
using *assms* **by** (*metis LeastI*)
lemma *least-contains'*[*intro?*, *simp*]:
fixes A :: $'a$ :: *wellorder set*
assumes $A \neq \{\}$
shows *least* $A \in A$
using *assms* **by** (*metis LeastI equals0I*)
lemma *least-least*[*intro?*, *simp*]:
fixes A :: $'a$ :: *wellorder set*
assumes $k \in A$
shows *least* $A \leq k$
using *assms* **by** (*metis Least-le*)
lemma *least-unique*:
fixes A :: $'a$ :: *wellorder set*
assumes $k \in A$ $k \leq$ *least* A
shows $k =$ *least* A
using *assms* **by** (*metis Least-le antisym*)
lemma *least-not-less*:
fixes A :: $'a$ :: *wellorder set*
assumes $k <$ *least* A
shows $k \notin A$
using *assms* **by** (*metis not-less-Least*)
lemma *leastI2-order*[*simp*]:
fixes A :: $'a$ :: *wellorder set*
assumes $A \neq \{\}$ $\bigwedge k.\ k \in A \Longrightarrow (\bigwedge l.\ l \in A \Longrightarrow k \leq l) \Longrightarrow P\ k$
shows P (*least* A)
proof (*rule LeastI2-order*)
show *least* $A \in A$ **using** *assms*(*1*) **by** *rule*
next
fix k
assume *1*: $k \in A$
show *least* $A \leq k$ **using** *1* **by** *rule*
next
fix k

assume *1*: $k \in A$ $\forall$ $l.$ $l \in A \longrightarrow k \leq l$
show $P\ k$ **using** *assms*(*2*) *1* **by** *auto*
qed

lemma *least-singleton*[*simp*]:
fixes $a :: 'a :: wellorder$
shows $least\ \{a\} = a$
by (*metis insert-not-empty least-contains′ singletonD*)

lemma *least-image*[*simp*]:
fixes $f :: 'a :: wellorder \Rightarrow 'b :: wellorder$
assumes $A \neq \{\}$ $\bigwedge k\ l.\ k \in A \Longrightarrow l \in A \Longrightarrow k \leq l \Longrightarrow f\,k \leq f\,l$
shows $least\ (f\ `\ A) = f\ (least\ A)$
proof (*rule leastI2-order*)
show $A \neq \{\}$ **using** *assms*(*1*) **by** *this*
next
fix k
assume *1*: $k \in A$ $\bigwedge i.\ i \in A \Longrightarrow k \leq i$
show $least\ (f\ `\ A) = f\,k$
proof (*rule leastI2-order*)
show $f\ `\ A \neq \{\}$ **using** *assms*(*1*) **by** *simp*
next
fix l
assume *2*: $l \in f\ `\ A$ $\bigwedge i.\ i \in f\ `\ A \Longrightarrow l \leq i$
show $l = f\,k$ **using** *assms*(*2*) *1* *2* **by** *force*
qed
qed

lemma *least-le*:
fixes $A\ B :: 'a :: wellorder\ set$
assumes $B \neq \{\}$
assumes $\bigwedge i.\ i \leq least\ A \Longrightarrow i \leq least\ B \Longrightarrow i \in B \Longrightarrow i \in A$
shows $least\ A \leq least\ B$
proof (*rule ccontr*)
assume *1*: $\neg\ least\ A \leq least\ B$
have *2*: $least\ B \in A$ **using** *assms*(*1*, *2*) *1* **by** *simp*
have *3*: $least\ A \leq least\ B$ **using** *2* **by** *rule*
show *False* **using** *1* *3* **by** *rule*
qed
lemma *least-eq*:
fixes $A\ B :: 'a :: wellorder\ set$
assumes $A \neq \{\}$ $B \neq \{\}$
assumes $\bigwedge i.\ i \leq least\ A \Longrightarrow i \leq least\ B \Longrightarrow i \in A \longleftrightarrow i \in B$
shows $least\ A = least\ B$
using *assms* **by** (*auto intro*: *antisym least-le*)

lemma *least-Suc*[*simp*]:
assumes $A \neq \{\}$
shows $least\ (Suc\ `\ A) = Suc\ (least\ A)$

```
proof (rule antisym)
  obtain k where 10: k ∈ A using assms by blast
  have 11: Suc k ∈ Suc ` A using 10 by auto
  have 20: least A ∈ A using 10 LeastI by metis
  have 21: least (Suc ` A) ∈ Suc ` A using 11 LeastI by metis
  have 30: ⋀ l. l ∈ A ⟹ least A ≤ l using 10 Least-le by metis
  have 31: ⋀ l. l ∈ Suc ` A ⟹ least (Suc ` A) ≤ l using 11 Least-le by metis
  show least (Suc ` A) ≤ Suc (least A) using 20 31 by auto
  show Suc (least A) ≤ least (Suc ` A) using 21 30 by auto
qed

lemma least-Suc-diff[simp]: Suc ` A − {least (Suc ` A)} = Suc ` (A − {least A})
proof (cases A = {})
  case True
  show ?thesis unfolding True by simp
next
  case False
  have Suc ` A − {least (Suc ` A)} = Suc ` A − {Suc (least A)} using False by
simp
  also have ... = Suc ` A − Suc ` {least A} by simp
  also have ... = Suc ` (A − {least A}) by blast
  finally show ?thesis by this
qed

lemma Max-diff-least[simp]:
  fixes A :: 'a :: wellorder set
  assumes finite A A − {least A} ≠ {}
  shows Max (A − {least A}) = Max A
proof −
  have 1: least A ∈ A using assms(2) by auto
  obtain a where 2: a ∈ A − {least A} using assms(2) by blast
  have Max A = Max (insert (least A) (A − {least A})) using insert-absorb 1
by force
  also have ... = max (least A) (Max (A − {least A}))
  proof (rule Max-insert)
    show finite (A − {least A}) using assms(1) by auto
    show A − {least A} ≠ {} using assms(2) by this
  qed
  also have ... = Max (A − {least A})
  proof (rule max-absorb2, rule Max-ge')
    show finite (A − {least A}) using assms(1) by auto
    show A − {least A} ≠ {} using assms(2) by this
    show a ∈ A − {least A} using 2 by this
    show least A ≤ a using 2 by simp
  qed
  finally show ?thesis by rule
qed

lemma nat-set-card-equality-less:
```

fixes A :: *nat set*
assumes $x \in A$ $y \in A$ *card* $\{z \in A.\ z < x\}$ = *card* $\{z \in A.\ z < y\}$
shows $x = y$
proof (*cases x y rule*: *linorder-cases*)
case *less*
have *0*: *finite* $\{z \in A.\ z < y\}$ **by** *simp*
have *1*: $\{z \in A.\ z < x\} \subset \{z \in A.\ z < y\}$ **using** *assms*(*1*, *2*) *less* **by** *force*
have *2*: *card* $\{z \in A.\ z < x\}$ < *card* $\{z \in A.\ z < y\}$ **using** *psubset-card-mono*
0 1 **by** *this*
show *?thesis* **using** *assms*(*3*) *2* **by** *simp*
next
case *equal*
show *?thesis* **using** *equal* **by** *this*
next
case *greater*
have *0*: *finite* $\{z \in A.\ z < x\}$ **by** *simp*
have *1*: $\{z \in A.\ z < y\} \subset \{z \in A.\ z < x\}$ **using** *assms*(*1*, *2*) *greater* **by** *force*
have *2*: *card* $\{z \in A.\ z < y\}$ < *card* $\{z \in A.\ z < x\}$ **using** *psubset-card-mono*
0 1 **by** *this*
show *?thesis* **using** *assms*(*3*) *2* **by** *simp*
qed

lemma *nat-set-card-equality-le*:
fixes A :: *nat set*
assumes $x \in A$ $y \in A$ *card* $\{z \in A.\ z \le x\}$ = *card* $\{z \in A.\ z \le y\}$
shows $x = y$
proof (*cases x y rule*: *linorder-cases*)
case *less*
have *0*: *finite* $\{z \in A.\ z \le y\}$ **by** *simp*
have *1*: $\{z \in A.\ z \le x\} \subset \{z \in A.\ z \le y\}$ **using** *assms*(*1*, *2*) *less* **by** *force*
have *2*: *card* $\{z \in A.\ z \le x\}$ < *card* $\{z \in A.\ z \le y\}$ **using** *psubset-card-mono*
0 1 **by** *this*
show *?thesis* **using** *assms*(*3*) *2* **by** *simp*
next
case *equal*
show *?thesis* **using** *equal* **by** *this*
next
case *greater*
have *0*: *finite* $\{z \in A.\ z \le x\}$ **by** *simp*
have *1*: $\{z \in A.\ z \le y\} \subset \{z \in A.\ z \le x\}$ **using** *assms*(*1*, *2*) *greater* **by** *force*
have *2*: *card* $\{z \in A.\ z \le y\}$ < *card* $\{z \in A.\ z \le x\}$ **using** *psubset-card-mono*
0 1 **by** *this*
show *?thesis* **using** *assms*(*3*) *2* **by** *simp*
qed

lemma *nat-set-card-mono*[*simp*]:
fixes A :: *nat set*
assumes $x \in A$
shows *card* $\{z \in A.\ z < x\}$ < *card* $\{z \in A.\ z < y\}$ $\longleftrightarrow$ $x < y$

proof
  assume *1*: *card* $\{z \in A.\ z < x\} <$ *card* $\{z \in A.\ z < y\}$
  show $x < y$
  proof (*rule ccontr*)
    assume *2*: $\neg\ x < y$
    have *3*: *card* $\{z \in A.\ z < y\} \leq$ *card* $\{z \in A.\ z < x\}$ **using** *2* **by** (*auto intro*:
card-mono)
    show *False* **using** *1 3* **by** *simp*
  qed
next
  assume *1*: $x < y$
  show *card* $\{z \in A.\ z < x\} <$ *card* $\{z \in A.\ z < y\}$
  proof (*intro psubset-card-mono psubsetI*)
    show *finite* $\{z \in A.\ z < y\}$ **by** *simp*
    show $\{z \in A.\ z < x\} \subseteq \{z \in A.\ z < y\}$ **using** *1* **by** *auto*
    show $\{z \in A.\ z < x\} \neq \{z \in A.\ z < y\}$ **using** *assms 1* **by** *blast*
  qed
qed

lemma *card-one*[*elim*]:
  assumes *card* $A = 1$
  obtains a
  where $A = \{a\}$
  using *assms* **by** (*metis One-nat-def card-Suc-eq*)

lemma *image-alt-def*: f ‘ $A = \{f\ x\ |x.\ x \in A\}$ **by** *auto*

lemma *supset-mono-inductive*[*mono*]:
  assumes $\bigwedge x.\ x \in B \longrightarrow x \in C$
  shows $A \subseteq B \longrightarrow A \subseteq C$
  using *assms* **by** *auto*
lemma *Collect-mono-inductive*[*mono*]:
  assumes $\bigwedge x.\ P\ x \longrightarrow Q\ x$
  shows $x \in \{x.\ P\ x\} \longrightarrow x \in \{x.\ Q\ x\}$
  using *assms* **by** *auto*

lemma *image-union-split*:
  assumes f ‘ $(A \cup B) = g$ ‘ C
  obtains $D\ E$
  where f ‘ $A = g$ ‘ D f ‘ $B = g$ ‘ E $D \subseteq C$ $E \subseteq C$
  using *assms* **unfolding** *image-Un*
  by (*metis* (*erased*, *lifting*) *inf-sup-ord*(*3*) *inf-sup-ord*(*4*) *subset-imageE*)
lemma *image-insert-split*:
  assumes *inj* g f ‘ *insert* a $B = g$ ‘ C
  obtains $d\ E$
  where $f\ a = g\ d$ f ‘ $B = g$ ‘ E $d \in C$ $E \subseteq C$
proof −
  have *1*: f ‘ $(\{a\} \cup B) = g$ ‘ C **using** *assms*(*2*) **by** *simp*
  obtain $D\ E$ **where** *2*: f ‘ $\{a\} = g$ ‘ D f ‘ $B = g$ ‘ E $D \subseteq C$ $E \subseteq C$

```
    using image-union-split 1 by this
   obtain d where 3: D = {d} using assms(1) 2(1) by (auto, metis (erased,
opaque-lifting) imageE
     image-empty image-insert inj-image-eq-iff singletonI)
  show ?thesis using that 2 unfolding 3 by simp
 qed

end
```

5 Basics

```
theory Basic-Extensions
imports HOL−Library.Infinite-Set
begin
```

5.1 Types

```
  type-synonym 'a step = 'a ⇒ 'a
```

5.2 Rules

```
  declare less-imp-le[dest, simp]

  declare le-funI[intro]
  declare le-funE[elim]
  declare le-funD[dest]

  lemma IdI'[intro]:
    assumes x = y
    shows (x, y) ∈ Id
    using assms by auto

  lemma (in order) order-le-cases:
    assumes x ≤ y
    obtains (eq) x = y | (lt) x < y
    using assms le-less by auto

  lemma (in linorder) linorder-cases':
    obtains (le) x ≤ y | (gt) x > y
    by force

  lemma monoI-comp[intro]:
    assumes mono f mono g
    shows mono (f ∘ g)
    using assms by (intro monoI, auto dest: monoD)
  lemma strict-monoI-comp[intro]:
    assumes strict-mono f strict-mono g
    shows strict-mono (f ∘ g)
    using assms by (intro strict-monoI, auto dest: strict-monoD)
```

lemma *eq-le-absorb*[*simp*]:
  fixes $x\ y :: 'a :: order$
  shows $x = y \wedge x \leq y \longleftrightarrow x = y$ $x \leq y \wedge x = y \longleftrightarrow x = y$
  by *auto*

lemma *INFM-Suc*[*simp*]: $(\exists_\infty\ i.\ P\ (Suc\ i)) \longleftrightarrow (\exists_\infty\ i.\ P\ i)$
  unfolding *INFM-nat* **using** *Suc-lessE less-Suc-eq* **by** *metis*
lemma *INFM-plus*[*simp*]: $(\exists_\infty\ i.\ P\ (i + n :: nat)) \longleftrightarrow (\exists_\infty\ i.\ P\ i)$
proof (*induct n*)
  case *0*
  show *?case* **by** *simp*
next
  case (*Suc n*)
  have $(\exists_\infty\ i.\ P\ (i + Suc\ n)) \longleftrightarrow (\exists_\infty\ i.\ P\ (Suc\ i + n))$ **by** *simp*
  also have $\ldots \longleftrightarrow (\exists_\infty\ i.\ P\ (i + n))$ **using** *INFM-Suc* **by** *this*
  also have $\ldots \longleftrightarrow (\exists_\infty\ i.\ P\ i)$ **using** *Suc* **by** *this*
  finally show *?case* **by** *this*
qed
lemma *INFM-minus*[*simp*]: $(\exists_\infty\ i.\ P\ (i - n :: nat)) \longleftrightarrow (\exists_\infty\ i.\ P\ i)$
proof (*induct n*)
  case *0*
  show *?case* **by** *simp*
next
  case (*Suc n*)
  have $(\exists_\infty\ i.\ P\ (i - Suc\ n)) \longleftrightarrow (\exists_\infty\ i.\ P\ (Suc\ i - Suc\ n))$ **using** *INFM-Suc*
by *meson*
  also have $\ldots \longleftrightarrow (\exists_\infty\ i.\ P\ (i - n))$ **by** *simp*
  also have $\ldots \longleftrightarrow (\exists_\infty\ i.\ P\ i)$ **using** *Suc* **by** *this*
  finally show *?case* **by** *this*
qed

5.3 Constants

definition $const :: 'a \Rightarrow 'b \Rightarrow 'a$
  where $const\ x \equiv \lambda\ \text{-}.\ x$
definition $const2 :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'a$
  where $const2\ x \equiv \lambda\ \text{-}\ \text{-}.\ x$
definition $const3 :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow 'a$
  where $const3\ x \equiv \lambda\ \text{-}\ \text{-}\ \text{-}.\ x$
definition $const4 :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'a$
  where $const4\ x \equiv \lambda\ \text{-}\ \text{-}\ \text{-}\ \text{-}.\ x$
definition $const5 :: 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow 'e \Rightarrow 'f \Rightarrow 'a$
  where $const5\ x \equiv \lambda\ \text{-}\ \text{-}\ \text{-}\ \text{-}\ \text{-}.\ x$

lemma *const-apply*[*simp*]: $const\ x\ y = x$ **unfolding** *const-def* **by** *rule*
lemma *const2-apply*[*simp*]: $const2\ x\ y\ z = x$ **unfolding** *const2-def* **by** *rule*
lemma *const3-apply*[*simp*]: $const3\ x\ y\ z\ u = x$ **unfolding** *const3-def* **by** *rule*
lemma *const4-apply*[*simp*]: $const4\ x\ y\ z\ u\ v = x$ **unfolding** *const4-def* **by** *rule*

lemma *const5-apply*[*simp*]: *const5 x y z u v w* = *x* **unfolding** *const5-def* **by** *rule*

definition *zip-fun* :: $('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c) \Rightarrow 'a \Rightarrow 'b \times 'c$ (**infixr** $\parallel$ *51*)
where $f \parallel g \equiv \lambda\ x.\ (f\ x,\ g\ x)$

lemma *zip-fun-simps*[*simp*]:
$(f \parallel g)\ x = (f\ x,\ g\ x)$
$fst \circ (f \parallel g) = f$
$snd \circ (f \parallel g) = g$
$fst \circ h \parallel snd \circ h = h$
$fst\ `\ range\ (f \parallel g) = range\ f$
$snd\ `\ range\ (f \parallel g) = range\ g$
unfolding *zip-fun-def* **by** *force+*

lemma *zip-fun-eq*[*dest*]:
assumes $f \parallel g = h \parallel i$
shows $f = h$ $g = i$
using *assms* **unfolding** *zip-fun-def* **by** (*auto dest*: *fun-cong*)

lemma *zip-fun-range-subset*[*intro*, *simp*]: $range\ (f \parallel g) \subseteq range\ f \times range\ g$
unfolding *zip-fun-def* **by** *blast*
lemma *zip-fun-range-finite*[*elim*]:
assumes *finite* (*range* $(f \parallel g)$)
obtains *finite* (*range f*) *finite* (*range g*)
proof
show *finite* (*range f*) **using** *finite-imageI* [*OF assms*(*1*), *of fst*]
by (*simp add*: *image-image*)
show *finite* (*range g*) **using** *finite-imageI* [*OF assms*(*1*), *of snd*]
by (*simp add*: *image-image*)
qed

lemma *zip-fun-split*:
obtains *f g*
where $h = f \parallel g$
proof
show $h = fst \circ h \parallel snd \circ h$ **by** *simp*
qed

abbreviation *None-None* ≡ (*None*, *None*)
abbreviation *None-Some* ≡ λ (*y*). (*None*, *Some y*)
abbreviation *Some-None* ≡ λ (*x*). (*Some x*, *None*)
abbreviation *Some-Some* ≡ λ (*x*, *y*). (*Some x*, *Some y*)

abbreviation *None-None-None* ≡ (*None*, *None*, *None*)
abbreviation *None-None-Some* ≡ λ (*z*). (*None*, *None*, *Some z*)
abbreviation *None-Some-None* ≡ λ (*y*). (*None*, *Some y*, *None*)
abbreviation *None-Some-Some* ≡ λ (*y*, *z*). (*None*, *Some y*, *Some z*)
abbreviation *Some-None-None* ≡ λ (*x*). (*Some x*, *None*, *None*)

abbreviation *Some-None-Some* $\equiv \lambda\ (x,\ z).\ (Some\ x,\ None,\ Some\ z)$
abbreviation *Some-Some-None* $\equiv \lambda\ (x,\ y).\ (Some\ x,\ Some\ y,\ None)$
abbreviation *Some-Some-Some* $\equiv \lambda\ (x,\ y,\ z).\ (Some\ x,\ Some\ y,\ Some\ z)$

lemma *inj-Some2*[*simp*, *intro*]:
inj None-Some
inj Some-None
inj Some-Some
by (*rule injI*, *force*)+

lemma *inj-Some3*[*simp*, *intro*]:
inj None-None-Some
inj None-Some-None
inj None-Some-Some
inj Some-None-None
inj Some-None-Some
inj Some-Some-None
inj Some-Some-Some
by (*rule injI*, *force*)+

definition *swap* :: $'a \times 'b \Rightarrow 'b \times 'a$
where *swap x* $\equiv$ (*snd x*, *fst x*)

lemma *swap-simps*[*simp*]: *swap* $(a,\ b) = (b,\ a)$ **unfolding** *swap-def* **by** *simp*
lemma *swap-inj*[*intro*, *simp*]: *inj swap* **by** (*rule injI*, *auto*)
lemma *swap-surj*[*intro*, *simp*]: *surj swap* **by** (*rule surjI*[**where** *?f* = *swap*], *auto*)
lemma *swap-bij*[*intro*, *simp*]: *bij swap* **by** (*rule bijI*, *auto*)

definition *push* :: $('a \times 'b) \times 'c \Rightarrow 'a \times 'b \times 'c$
where *push x* $\equiv$ (*fst* (*fst x*), *snd* (*fst x*), *snd x*)
definition *pull* :: $'a \times 'b \times 'c \Rightarrow ('a \times 'b) \times 'c$
where *pull x* $\equiv$ ((*fst x*, *fst* (*snd x*)), *snd* (*snd x*))

lemma *push-simps*[*simp*]: *push* $((x,\ y),\ z) = (x,\ y,\ z)$ **unfolding** *push-def* **by** *simp*
lemma *pull-simps*[*simp*]: *pull* $(x,\ y,\ z) = ((x,\ y),\ z)$ **unfolding** *pull-def* **by** *simp*

definition *label* :: $'vertex \times 'label \times 'vertex \Rightarrow 'label$
where *label* $\equiv$ *fst* $\circ$ *snd*

lemma *label-select*[*simp*]: *label* $(p,\ a,\ q) = a$ **unfolding** *label-def* **by** *simp*

5.4 Theorems for @termcurry and @termsplit

lemma *curry-split*[*simp*]: *curry* $\circ$ *case-prod* = *id* **by** *auto*
lemma *split-curry*[*simp*]: *case-prod* $\circ$ *curry* = *id* **by** *auto*

lemma *curry-le*[*simp*]: *curry f* $\leq$ *curry g* $\longleftrightarrow$ *f* $\leq$ *g* **unfolding** *le-fun-def* **by** *force*

lemma *split-le*[*simp*]: *case-prod f* $\leq$ *case-prod g* $\longleftrightarrow$ *f* $\leq$ *g* **unfolding** *le-fun-def* **by** *force*

lemma *mono-curry-left*[*simp*]: *mono* (*curry* $\circ$ *h*) $\longleftrightarrow$ *mono h*
  unfolding *mono-def* **by** *fastforce*
lemma *mono-split-left*[*simp*]: *mono* (*case-prod* $\circ$ *h*) $\longleftrightarrow$ *mono h*
  unfolding *mono-def* **by** *fastforce*
lemma *mono-curry-right*[*simp*]: *mono* (*h* $\circ$ *curry*) $\longleftrightarrow$ *mono h*
  unfolding *mono-def split-le*[*symmetric*] **by** *bestsimp*
lemma *mono-split-right*[*simp*]: *mono* (*h* $\circ$ *case-prod*) $\longleftrightarrow$ *mono h*
  unfolding *mono-def curry-le*[*symmetric*] **by** *bestsimp*

lemma *Collect-curry*[*simp*]: $\{x.\ P\ (curry\ x)\} = case\text{-}prod\ `\ \{x.\ P\ x\}$ **using** *image-Collect* **by** *fastforce*

lemma *Collect-split*[*simp*]: $\{x.\ P\ (case\text{-}prod\ x)\} = curry\ `\ \{x.\ P\ x\}$ **using** *image-Collect* **by** *force*

lemma *gfp-split-curry*[*simp*]: *gfp* (*case-prod* $\circ$ *f* $\circ$ *curry*) = *case-prod* (*gfp f*)
proof –
  have *gfp* (*case-prod* $\circ$ *f* $\circ$ *curry*) = *Sup* $\{u.\ u \leq case\text{-}prod\ (f\ (curry\ u))\}$ **unfolding** *gfp-def* **by** *simp*
  also have ... = *Sup* $\{u.\ curry\ u \leq curry\ (case\text{-}prod\ (f\ (curry\ u)))\}$ **unfolding** *curry-le* **by** *simp*
  also have ... = *Sup* $\{u.\ curry\ u \leq f\ (curry\ u)\}$ **by** *simp*
  also have ... = *Sup* (*case-prod* ‘ $\{u.\ u \leq f\ u\}$) **unfolding** *Collect-curry*[*of* λ *u.* $u \leq f\ u$] **by** *simp*
  also have ... = *case-prod* (*Sup* $\{u.\ u \leq f\ u\}$) **by** (*force simp add*: *image-comp*)
  also have ... = *case-prod* (*gfp f*) **unfolding** *gfp-def* **by** *simp*
  finally show *?thesis* **by** *this*
qed
lemma *gfp-curry-split*[*simp*]: *gfp* (*curry* $\circ$ *f* $\circ$ *case-prod*) = *curry* (*gfp f*)
proof –
  have *gfp* (*curry* $\circ$ *f* $\circ$ *case-prod*) = *Sup* $\{u.\ u \leq curry\ (f\ (case\text{-}prod\ u))\}$ **unfolding** *gfp-def* **by** *simp*
  also have ... = *Sup* $\{u.\ case\text{-}prod\ u \leq case\text{-}prod\ (curry\ (f\ (case\text{-}prod\ u)))\}$ **unfolding** *split-le* **by** *simp*
  also have ... = *Sup* $\{u.\ case\text{-}prod\ u \leq f\ (case\text{-}prod\ u)\}$ **by** *simp*
  also have ... = *Sup* (*curry* ‘ $\{u.\ u \leq f\ u\}$) **unfolding** *Collect-split*[*of* λ *u.* $u \leq f\ u$] **by** *simp*
  also have ... = *curry* (*Sup* $\{u.\ u \leq f\ u\}$) **by** (*force simp add*: *image-comp*)
  also have ... = *curry* (*gfp f*) **unfolding** *gfp-def* **by** *simp*
  finally show *?thesis* **by** *this*
qed

lemma *not-someI*:
  assumes $\bigwedge x.\ P\ x \Longrightarrow False$
  shows $\neg\ P\ (SOME\ x.\ P\ x)$

using *assms* **by** *blast*
lemma *some-ccontr*:
assumes $(\bigwedge x.\ \neg\ P\ x) \Longrightarrow False$
shows $P\ (SOME\ x.\ P\ x)$
using *assms someI-ex ccontr* **by** *metis*

end

6 Relations

theory *Relation-Extensions*
imports
Basic-Extensions
begin

abbreviation *rev-lex-prod* (**infixr** <*rlex*> *80*)
where r_1 <*rlex*> $r_2 \equiv$ *inv-image* (r_2 <*lex*> r_1) *swap*

lemmas *sym-rtranclp*[*intro*] = *sym-rtrancl*[*to-pred*]

definition *liftablep* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
where *liftablep* $r\ f \equiv \forall\ x\ y.\ r\ x\ y \longrightarrow r\ (f\ x)\ (f\ y)$

lemma *liftablepI*[*intro*]:
assumes $\bigwedge x\ y.\ r\ x\ y \Longrightarrow r\ (f\ x)\ (f\ y)$
shows *liftablep* $r\ f$
using *assms*
unfolding *liftablep-def*
by *simp*
lemma *liftablepE*[*elim*]:
assumes *liftablep* $r\ f$
assumes $r\ x\ y$
obtains $r\ (f\ x)\ (f\ y)$
using *assms*
unfolding *liftablep-def*
by *simp*

lemma *liftablep-rtranclp*:
assumes *liftablep* $r\ f$
shows *liftablep* $r^{**}\ f$
proof
fix $x\ y$
assume $r^{**}\ x\ y$
thus $r^{**}\ (f\ x)\ (f\ y)$
using *assms*
by (*induct rule*: *rtranclp-induct*, *force*+)
qed

definition *confluentp* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow bool$

where *confluentp r* ≡ $\forall$ *x y1 y2*. r^{**} *x y1* $\longrightarrow$ r^{**} *x y2* $\longrightarrow$ ($\exists$ *z*. r^{**} *y1 z* $\wedge$ r^{**} *y2 z*)

lemma *confluentpI*[*intro*]:
assumes $\bigwedge$ *x y1 y2*. r^{**} *x y1* $\Longrightarrow$ r^{**} *x y2* $\Longrightarrow$ $\exists$ *z*. r^{**} *y1 z* $\wedge$ r^{**} *y2 z*
shows *confluentp r*
using *assms*
unfolding *confluentp-def*
by *simp*

lemma *confluentpE*[*elim*]:
assumes *confluentp r*
assumes r^{**} *x y1* r^{**} *x y2*
obtains *z*
where r^{**} *y1 z* r^{**} *y2 z*
using *assms*
unfolding *confluentp-def*
by *blast*

lemma *confluentpI'*[*intro*]:
assumes $\bigwedge$ *x y1 y2*. r^{**} *x y1* $\Longrightarrow$ *r x y2* $\Longrightarrow$ $\exists$ *z*. r^{**} *y1 z* $\wedge$ r^{**} *y2 z*
shows *confluentp r*
proof
fix *x y1 y2*
assume r^{**} *x y1* r^{**} *x y2*
thus $\exists$ *z*. r^{**} *y1 z* $\wedge$ r^{**} *y2 z* **using** *assms* **by** (*induct rule*: *rtranclp-induct*, *force+*)
qed

lemma *transclp-eq-implies-confluent-imp*:
assumes $r1^{**}$ = $r2^{**}$
assumes *confluentp r1*
shows *confluentp r2*
using *assms*
by *force*

lemma *transclp-eq-implies-confluent-eq*:
assumes $r1^{**}$ = $r2^{**}$
shows *confluentp r1* $\longleftrightarrow$ *confluentp r2*
using *assms transclp-eq-implies-confluent-imp*
by *metis*

definition *diamondp* :: (*'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*
where *diamondp r* ≡ $\forall$ *x y1 y2*. *r x y1* $\longrightarrow$ *r x y2* $\longrightarrow$ ($\exists$ *z*. *r y1 z* $\wedge$ *r y2 z*)

lemma *diamondpI*[*intro*]:
assumes $\bigwedge$ *x y1 y2*. *r x y1* $\Longrightarrow$ *r x y2* $\Longrightarrow$ $\exists$ *z*. *r y1 z* $\wedge$ *r y2 z*
shows *diamondp r*
using *assms*

unfolding *diamondp-def*
by *simp*

lemma *diamondpE*[*elim*]:
assumes *diamondp r*
assumes *r x y1 r x y2*
obtains *z*
where *r y1 z r y2 z*
using *assms*
unfolding *diamondp-def*
by *blast*

lemma *diamondp-implies-confluentp*:
assumes *diamondp r*
shows *confluentp r*
proof (*rule confluentpI'*)
fix *x y1 y2*
assume r^{**} *x y1 r x y2*
hence ∃ *z*. *r y1 z* ∧ r^{**} *y2 z* **using** *assms* **by** (*induct rule*: *rtranclp-induct*, *force+*)
thus ∃ *z*. r^{**} *y1 z* ∧ r^{**} *y2 z* **by** *blast*
qed

locale *wellfounded-relation* =
fixes *R* :: *'a* ⇒ *'a* ⇒ *bool*
assumes *wellfounded*: *wfP R*

end

7 Transition Systems

theory *Transition-System-Extensions*
imports
Basics/Word-Prefixes
Extensions/Set-Extensions
Extensions/Relation-Extensions
Transition-Systems-and-Automata.Transition-System
Transition-Systems-and-Automata.Transition-System-Extra
Transition-Systems-and-Automata.Transition-System-Construction
begin

context *transition-system-initial*
begin

definition *cycles* :: *'state* ⇒ *'transition list set*
where *cycles p* ≡ {*w*. *path w p* ∧ *target w p* = *p*}

lemma *cyclesI*[*intro!*]:
assumes *path w p target w p* = *p*

shows $w \in \mathit{cycles}\ p$
using *assms* **unfolding** *cycles-def* **by** *auto*
lemma *cyclesE*[*elim*!]:
assumes $w \in \mathit{cycles}\ p$
obtains $\mathit{path}\ w\ p\ \mathit{target}\ w\ p = p$
using *assms* **unfolding** *cycles-def* **by** *auto*

inductive-set *executable* :: *'transition set*
where *executable*: $p \in \mathit{nodes} \Longrightarrow \mathit{enabled}\ a\ p \Longrightarrow a \in \mathit{executable}$

lemma *executableI-step*[*intro*!]:
assumes $p \in \mathit{nodes}\ \mathit{enabled}\ a\ p$
shows $a \in \mathit{executable}$
using *executable assms* **by** *this*
lemma *executableI-words-fin*[*intro*!]:
assumes $p \in \mathit{nodes}\ \mathit{path}\ w\ p$
shows $\mathit{set}\ w \subseteq \mathit{executable}$
using *assms* **by** (*induct w arbitrary*: *p*, *auto del*: *subsetI*)
lemma *executableE*[*elim?*]:
assumes $a \in \mathit{executable}$
obtains *p*
where $p \in \mathit{nodes}\ \mathit{enabled}\ a\ p$
using *assms* **by** *induct auto*

end

locale *transition-system-interpreted* =
transition-system ex en
for *ex* :: $'action \Rightarrow 'state \Rightarrow 'state$
and *en* :: $'action \Rightarrow 'state \Rightarrow bool$
and *int* :: $'state \Rightarrow 'interpretation$
begin

definition *visible* :: *'action set*
where $\mathit{visible} \equiv \{a.\ \exists\ q.\ \mathit{en}\ a\ q \wedge \mathit{int}\ q \neq \mathit{int}\ (\mathit{ex}\ a\ q)\}$

lemma *visibleI*[*intro*]:
assumes $\mathit{en}\ a\ q\ \mathit{int}\ q \neq \mathit{int}\ (\mathit{ex}\ a\ q)$
shows $a \in \mathit{visible}$
using *assms* **unfolding** *visible-def* **by** *auto*
lemma *visibleE*[*elim*]:
assumes $a \in \mathit{visible}$
obtains *q*
where $\mathit{en}\ a\ q\ \mathit{int}\ q \neq \mathit{int}\ (\mathit{ex}\ a\ q)$
using *assms* **unfolding** *visible-def* **by** *auto*

abbreviation $\mathit{invisible} \equiv -\ \mathit{visible}$

lemma *execute-fin-word-invisible*:

assumes *path w p set w* $\subseteq$ *invisible*
shows *int* (*target w p*) = *int p*
using *assms* **by** (*induct w arbitrary*: *p rule*: *list.induct*, *auto*)
lemma *execute-inf-word-invisible*:
assumes *run w p* $k \leq l \bigwedge i.\ k \leq i \Longrightarrow i < l \Longrightarrow w\ !!\ i \notin$ *visible*
shows *int* ((*p* ## *trace w p*) !! *k*) = *int* ((*p* ## *trace w p*) !! *l*)
proof −
have (*p* ## *trace w p*) !! *l* = *target* (*stake l w*) *p* **by** *simp*
also have *stake l w* = *stake k w* @ *stake* ($l - k$) (*sdrop k w*) **using** *assms*(*2*) **by** *simp*
also have *target* ... *p* = *target* (*stake* ($l - k$) (*sdrop k w*)) (*target* (*stake k w*) *p*)
unfolding *fold-append comp-apply* **by** *rule*
also have *int* ... = *int* (*target* (*stake k w*) *p*)
proof (*rule execute-fin-word-invisible*)
have *w* = *stake l w* @− *sdrop l w* **by** *simp*
also have *stake l w* = *stake k w* @ *stake* ($l - k$) (*sdrop k w*) **using** *assms*(*2*) **by** *simp*
finally have *1*: *run* (*stake k w* @− *stake* ($l - k$) (*sdrop k w*) @− *sdrop l w*) *p*
unfolding *shift-append* **using** *assms*(*1*) **by** *simp*
show *path* (*stake* ($l - k$) (*sdrop k w*)) (*target* (*stake k w*) *p*) **using** *1* **by** *auto*
show *set* (*stake* ($l - k$) (*sdrop k w*)) $\subseteq$ *invisible* **using** *assms*(*3*) **by** (*auto simp*: *set-stake-snth*)
qed
also have ... = *int* ((*p* ## *trace w p*) !! *k*) **by** *simp*
finally show *?thesis* **by** *rule*
qed

end

locale *transition-system-complete* =
transition-system-initial ex en init +
transition-system-interpreted ex en int
for *ex* :: *'action* ⇒ *'state* ⇒ *'state*
and *en* :: *'action* ⇒ *'state* ⇒ *bool*
and *init* :: *'state* ⇒ *bool*
and *int* :: *'state* ⇒ *'interpretation*
begin

definition *language* :: *'interpretation stream set*
where *language* ≡ {*smap int* (*p* ## *trace w p*) |*p w*. *init p* ∧ *run w p*}

lemma *languageI*[*intro*!]:
assumes *w* = *smap int* (*p* ## *trace v p*) *init p run v p*
shows *w* ∈ *language*
using *assms* **unfolding** *language-def* **by** *auto*
lemma *languageE*[*elim*!]:

assumes $w \in language$
obtains $p\ v$
where $w = smap\ int\ (p\ \#\#\ trace\ v\ p)\ init\ p\ run\ v\ p$
using *assms* **unfolding** *language-def* **by** *auto*

end

locale *transition-system-finite-nodes* =
transition-system-initial ex en init
for $ex :: 'action \Rightarrow 'state \Rightarrow 'state$
and $en :: 'action \Rightarrow 'state \Rightarrow bool$
and $init :: 'state \Rightarrow bool$
+
assumes *reachable-finite*: *finite nodes*

locale *transition-system-cut* =
transition-system-finite-nodes ex en init
for $ex :: 'action \Rightarrow 'state \Rightarrow 'state$
and $en :: 'action \Rightarrow 'state \Rightarrow bool$
and $init :: 'state \Rightarrow bool$
+
fixes $cuts :: 'action\ set$
assumes *cycles-cut*: $p \in nodes \Longrightarrow w \in cycles\ p \Longrightarrow w \neq [] \Longrightarrow set\ w \cap cuts \neq \{\}$
begin

inductive $scut :: 'state \Rightarrow 'state \Rightarrow bool$
where *scut*: $p \in nodes \Longrightarrow en\ a\ p \Longrightarrow a \notin cuts \Longrightarrow scut\ p\ (ex\ a\ p)$

declare *scut.intros*[*intro*!]
declare *scut.cases*[*elim*!]

lemma *scut-reachable*:
assumes *scut p q*
shows $p \in nodes\ q \in nodes$
using *assms* **by** *auto*
lemma *scut-trancl*:
assumes $scut^{++}\ p\ q$
obtains w
where $path\ w\ p\ target\ w\ p = q\ set\ w \cap cuts = \{\}\ w \neq []$
using *assms*
proof (*induct arbitrary*: *thesis*)
case (*base q*)
show *?case* **using** *base* **by** *force*
next
case (*step q r*)
obtain w **where** *1*: $path\ w\ p\ target\ w\ p = q\ set\ w \cap cuts = \{\}\ w \neq []$
using *step*(*3*) **by** *this*
obtain a **where** *2*: $en\ a\ q\ a \notin cuts\ ex\ a\ q = r$ **using** *step*(*2*) **by** *auto*

show *?case*
proof (*rule step*(*4*))
show *path* (w @ [a]) p **using** *1 2* **by** *auto*
show *target* (w @ [a]) $p = r$ **using** *1 2* **by** *auto*
show *set* (w @ [a]) $\cap$ *cuts* = {} **using** *1 2* **by** *auto*
show w @ [a] $\neq$ [] **by** *auto*
qed
qed

sublocale *wellfounded-relation* $scut^{-1-1}$
proof (*unfold-locales*, *intro finite-acyclic-wf-converse*[*to-pred*] *acyclicI*[*to-pred*], *safe*)
have *1*: {(p, q). *scut* p q} $\subseteq$ *nodes* $\times$ *nodes* **using** *scut-reachable* **by** *blast*
have *2*: *finite* (*nodes* $\times$ *nodes*)
using *finite-cartesian-product reachable-finite* **by** *blast*
show *finite* {(p, q). *scut* p q} **using** *1 2* **by** *blast*
next
fix p
assume *1*: $scut^{++}$ p p
have *2*: $p \in$ *nodes* **using** *1 tranclE*[*to-pred*] *scut-reachable* **by** *metis*
obtain w **where** *3*: *path* w p *target* w $p = p$ *set* $w \cap$ *cuts* = {} $w \neq$ []
using *scut-trancl 1* **by** *this*
have *4*: $w \in$ *cycles* p **using** *3*(*1*, *2*) **by** *auto*
have *5*: *set* $w \cap$ *cuts* $\neq$ {} **using** *cycles-cut 2 4 3*(*4*) **by** *this*
show *False* **using** *3*(*3*) *5* **by** *simp*
qed

lemma *no-cut-scut*:
assumes $p \in$ *nodes en a p a* $\notin$ *cuts*
shows $scut^{-1-1}$ (*ex a p*) p
using *assms* **by** *auto*

end

locale *transition-system-sticky* =
transition-system-complete ex en init int +
transition-system-cut ex en init sticky
for *ex* :: *'action* ⇒ *'state* ⇒ *'state*
and *en* :: *'action* ⇒ *'state* ⇒ *bool*
and *init* :: *'state* ⇒ *bool*
and *int* :: *'state* ⇒ *'interpretation*
and *sticky* :: *'action set*
+
assumes *executable-visible-sticky*: *executable* $\cap$ *visible* $\subseteq$ *sticky*

end

8 Trace Theory

theory *Traces*
imports *Basics/Word-Prefixes*
begin

locale *traces* =
fixes *ind* :: *'item* ⇒ *'item* ⇒ *bool*
assumes *independence-symmetric*[*sym*]: *ind a b* ⟹ *ind b a*
begin

abbreviation *Ind* :: *'item set* ⇒ *'item set* ⇒ *bool*
where *Ind A B* ≡ ∀ *a* ∈ *A*. ∀ *b* ∈ *B*. *ind a b*

inductive *eq-swap* :: *'item list* ⇒ *'item list* ⇒ *bool* (**infix** $=_S$ *50*)
where *swap*: *ind a b* ⟹ *u* @ [*a*] @ [*b*] @ *v* $=_S$ *u* @ [*b*] @ [*a*] @ *v*

declare *eq-swap.intros*[*intro*]
declare *eq-swap.cases*[*elim*]

lemma *eq-swap-sym*[*sym*]: *v* $=_S$ *w* ⟹ *w* $=_S$ *v* **using** *independence-symmetric* **by** *auto*

lemma *eq-swap-length*[*dest*]: $w_1 =_S w_2$ ⟹ *length* w_1 = *length* w_2 **by** *force*
lemma *eq-swap-range*[*dest*]: $w_1 =_S w_2$ ⟹ *set* w_1 = *set* w_2 **by** *force*

lemma *eq-swap-extend*:
assumes $w_1 =_S w_2$
shows *u* @ w_1 @ *v* $=_S$ *u* @ w_2 @ *v*
using *assms*
proof *induct*
case (*swap a b u' v'*)
have *u* @ (*u'* @ [*a*] @ [*b*] @ *v'*) @ *v* = (*u* @ *u'*) @ [*a*] @ [*b*] @ (*v'* @ *v*) **by** *simp*
also have ... $=_S$ (*u* @ *u'*) @ [*b*] @ [*a*] @ (*v'* @ *v*) **using** *swap* **by** *blast*
also have ... = *u* @ (*u'* @ [*b*] @ [*a*] @ *v'*) @ *v* **by** *simp*
finally show *?case* **by** *this*
qed

lemma *eq-swap-remove1*:
assumes $w_1 =_S w_2$
obtains (*equal*) *remove1 c* w_1 = *remove1 c* w_2 | (*swap*) *remove1 c* w_1 $=_S$ *remove1 c* w_2
using *assms*
proof *induct*
case (*swap a b u v*)
have *c* ∉ *set* (*u* @ [*a*] @ [*b*] @ *v*) ∨
c ∈ *set u* ∨
c ∉ *set u* ∧ *c* = *a* ∨

$c \notin set\ u \wedge c \neq a \wedge c = b \vee$
$c \notin set\ u \wedge c \neq a \wedge c \neq b \wedge c \in set\ v$
by *auto*
thus *?case*
proof (*elim disjE*)
assume *0*: $c \notin set\ (u\ @\ [a]\ @\ [b]\ @\ v)$
have *1*: $c \notin set\ (u\ @\ [b]\ @\ [a]\ @\ v)$ **using** *0* **by** *auto*
have *2*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) = u\ @\ [a]\ @\ [b]\ @\ v$ **using** *remove1-idem 0* **by** *this*
have *3*: $remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v) = u\ @\ [b]\ @\ [a]\ @\ v$ **using** *remove1-idem 1* **by** *this*
have *4*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) =_S remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v)$
unfolding *2 3* **using** *eq-swap.intros swap*(*1*) **by** *this*
show *thesis* **using** *swap*(*3*) *4* **by** *this*
next
assume *0*: $c \in set\ u$
have *2*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) = remove1\ c\ u\ @\ [a]\ @\ [b]\ @\ v$
unfolding *remove1-append* **using** *0* **by** *simp*
have *3*: $remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v) = remove1\ c\ u\ @\ [b]\ @\ [a]\ @\ v$
unfolding *remove1-append* **using** *0* **by** *simp*
have *4*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) =_S remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v)$
unfolding *2 3* **using** *eq-swap.intros swap*(*1*) **by** *this*
show *thesis* **using** *swap*(*3*) *4* **by** *this*
next
assume *0*: $c \notin set\ u \wedge c = a$
have *2*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) = u\ @\ [b]\ @\ v$
unfolding *remove1-append* **using** *remove1-idem 0* **by** *auto*
have *3*: $remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v) = u\ @\ [b]\ @\ v$
unfolding *remove1-append* **using** *remove1-idem 0* **by** *auto*
have *4*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) = remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v)$
unfolding *2 3* **by** *rule*
show *thesis* **using** *swap*(*2*) *4* **by** *this*
next
assume *0*: $c \notin set\ u \wedge c \neq a \wedge c = b$
have *2*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) = u\ @\ [a]\ @\ v$
unfolding *remove1-append* **using** *remove1-idem 0* **by** *auto*
have *3*: $remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v) = u\ @\ [a]\ @\ v$
unfolding *remove1-append* **using** *remove1-idem 0* **by** *auto*
have *4*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) = remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v)$
unfolding *2 3* **by** *rule*
show *thesis* **using** *swap*(*2*) *4* **by** *this*
next
assume *0*: $c \notin set\ u \wedge c \neq a \wedge c \neq b \wedge c \in set\ v$
have *2*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) = u\ @\ [a]\ @\ [b]\ @\ remove1\ c\ v$
unfolding *remove1-append* **using** *0* **by** *simp*
have *3*: $remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v) = u\ @\ [b]\ @\ [a]\ @\ remove1\ c\ v$
unfolding *remove1-append* **using** *0* **by** *simp*
have *4*: $remove1\ c\ (u\ @\ [a]\ @\ [b]\ @\ v) =_S remove1\ c\ (u\ @\ [b]\ @\ [a]\ @\ v)$
unfolding *2 3* **using** *eq-swap.intros swap*(*1*) **by** *this*

show *?thesis* **using** *swap*(*3*) *4* **by** *this*
qed
qed

lemma *eq-swap-rev*:
assumes $w_1 =_S w_2$
shows *rev* $w_1 =_S$ *rev* w_2
using *assms*
proof *induct*
case (*swap a b u v*)
have *1*: *rev v* @ [*a*] @ [*b*] @ *rev u* $=_S$ *rev v* @ [*b*] @ [*a*] @ *rev u* **using** *swap* **by** *blast*
have *2*: *rev v* @ [*b*] @ [*a*] @ *rev u* $=_S$ *rev v* @ [*a*] @ [*b*] @ *rev u* **using** *1 eq-swap-sym* **by** *blast*
show *?case* **using** *2* **by** *simp*
qed

abbreviation *eq-fin* :: *'item list* ⇒ *'item list* ⇒ *bool* (**infix** $=_F$ *50*)
where *eq-fin* ≡ *eq-swap***

lemma *eq-fin-symp*[*intro*, *sym*]: $u =_F v \Longrightarrow v =_F u$
using *eq-swap-sym sym-rtrancl*[*to-pred*] **unfolding** *symp-def* **by** *metis*

lemma *eq-fin-length*[*dest*]: $w_1 =_F w_2 \Longrightarrow$ *length* w_1 = *length* w_2
by (*induct rule*: *rtranclp.induct*, *auto*)
lemma *eq-fin-range*[*dest*]: $w_1 =_F w_2 \Longrightarrow$ *set* w_1 = *set* w_2
by (*induct rule*: *rtranclp.induct*, *auto*)

lemma *eq-fin-remove1*:
assumes $w_1 =_F w_2$
shows *remove1 c* $w_1 =_F$ *remove1 c* w_2
using *assms*
proof *induct*
case (*base*)
show *?case* **by** *simp*
next
case (*step* w_2 w_3)
show *?case*
using *step*(*2*)
proof (*cases rule*: *eq-swap-remove1*[**where** *?c* = *c*])
case *equal*
show *?thesis* **using** *step equal* **by** *simp*
next
case *swap*
show *?thesis* **using** *step swap* **by** *auto*
qed
qed

lemma *eq-fin-rev*:

assumes $w_1 =_F w_2$
shows *rev* $w_1 =_F$ *rev* w_2
using *assms* **by** (*induct*, *auto dest*: *eq-swap-rev*)

lemma *eq-fin-concat-eq-fin-start*:
assumes u @ $v_1 =_F u$ @ v_2
shows $v_1 =_F v_2$
using *assms*
proof (*induct u arbitrary*: v_1 v_2 *rule*: *rev-induct*)
case (*Nil*)
show *?case* **using** *Nil* **by** *simp*
next
case (*snoc a u*)
have *1*: u @ $[a]$ @ $v_1 =_F u$ @ $[a]$ @ v_2 **using** *snoc*(*2*) **by** *simp*
have *2*: $[a]$ @ $v_1 =_F [a]$ @ v_2 **using** *snoc*(*1*) *1* **by** *this*
show *?case* **using** *eq-fin-remove1*[*OF 2*, *of a*] **by** *simp*
qed

lemma *eq-fin-concat*: u @ w_1 @ $v =_F u$ @ w_2 @ $v \longleftrightarrow w_1 =_F w_2$
proof
assume *0*: u @ w_1 @ $v =_F u$ @ w_2 @ v
have *1*: w_1 @ $v =_F w_2$ @ v **using** *eq-fin-concat-eq-fin-start 0* **by** *this*
have *2*: *rev* (w_1 @ v) $=_F$ *rev* (w_2 @ v) **using** *1* **by** (*blast dest*: *eq-fin-rev*)
have *3*: *rev v* @ *rev* $w_1 =_F$ *rev v* @ *rev* w_2 **using** *2* **by** *simp*
have *4*: *rev* $w_1 =_F$ *rev* w_2 **using** *eq-fin-concat-eq-fin-start 3* **by** *this*
have *5*: *rev* (*rev* w_1) $=_F$ *rev* (*rev* w_2) **using** *4* **by** (*blast dest*: *eq-fin-rev*)
show $w_1 =_F w_2$ **using** *5* **by** *simp*
next
show u @ w_1 @ $v =_F u$ @ w_2 @ v **if** $w_1 =_F w_2$
using *that* **by** (*induct*, *auto dest*: *eq-swap-extend*[*of - - u v*])
qed
lemma *eq-fin-concat-start*[*iff*]: w @ $w_1 =_F w$ @ $w_2 \longleftrightarrow w_1 =_F w_2$
using *eq-fin-concat*[*of w -* []] **by** *simp*
lemma *eq-fin-concat-end*[*iff*]: w_1 @ $w =_F w_2$ @ $w \longleftrightarrow w_1 =_F w_2$
using *eq-fin-concat*[*of* [] *- w*] **by** *simp*

lemma *ind-eq-fin′*:
assumes *Ind* $\{a\}$ (*set v*)
shows $[a]$ @ $v =_F v$ @ $[a]$
using *assms*
proof (*induct v*)
case (*Nil*)
show *?case* **by** *simp*
next
case (*Cons b v*)
have *1*: *Ind* $\{a\}$ (*set v*) **using** *Cons*(*2*) **by** *auto*
have *2*: *ind a b* **using** *Cons*(*2*) **by** *auto*
have $[a]$ @ b # $v = [a]$ @ $[b]$ @ v **by** *simp*
also have $\ldots =_S [b]$ @ $[a]$ @ v **using** *eq-swap.intros*[*OF 2*, *of* []] **by** *auto*

also have $\ldots =_F [b]$ @ v @ $[a]$ **using** *Cons*(*1*) *1* **by** *blast*
also have $\ldots = (b \# v)$ @ $[a]$ **by** *simp*
finally show *?case* **by** *this*
qed

lemma *ind-eq-fin*[*intro*]:
  assumes *Ind* (*set u*) (*set v*)
  shows u @ $v =_F v$ @ u
using *assms*
proof (*induct u*)
  case (*Nil*)
  show *?case* **by** *simp*
next
  case (*Cons a u*)
  have *1*: *Ind* (*set u*) (*set v*) **using** *Cons*(*2*) **by** *auto*
  have *2*: *Ind* $\{a\}$ (*set v*) **using** *Cons*(*2*) **by** *auto*
  have $(a \# u)$ @ $v = [a]$ @ u @ v **by** *simp*
  also have $\ldots =_F [a]$ @ v @ u **using** *Cons*(*1*) *1* **by** *blast*
  also have $\ldots = ([a]$ @ $v)$ @ u **by** *simp*
  also have $\ldots =_F (v$ @ $[a])$ @ u **using** *ind-eq-fin′ 2* **by** *blast*
  also have $\ldots = v$ @ $(a \# u)$ **by** *simp*
  finally show *?case* **by** *this*
qed

definition *le-fin* :: *′item list* $\Rightarrow$ *′item list* $\Rightarrow$ *bool* (**infix** $\preceq_F$ *50*)
  where $w_1 \preceq_F w_2 \equiv \exists\ v_1.\ w_1$ @ $v_1 =_F w_2$

lemma *le-finI*[*intro 0*]:
  assumes w_1 @ $v_1 =_F w_2$
  shows $w_1 \preceq_F w_2$
  using *assms* **unfolding** *le-fin-def* **by** *auto*
lemma *le-finE*[*elim 0*]:
  assumes $w_1 \preceq_F w_2$
  obtains v_1
  where w_1 @ $v_1 =_F w_2$
  using *assms* **unfolding** *le-fin-def* **by** *auto*

lemma *le-fin-empty*[*simp*]: $[] \preceq_F w$ **by** *force*
lemma *le-fin-trivial*[*intro*]: $w_1 =_F w_2 \Longrightarrow w_1 \preceq_F w_2$
proof
  assume *1*: $w_1 =_F w_2$
  show w_1 @ $[] =_F w_2$ **using** *1* **by** *simp*
qed

lemma *le-fin-length*[*dest*]: $w_1 \preceq_F w_2 \Longrightarrow$ *length* $w_1 \leq$ *length* w_2 **by** *force*
lemma *le-fin-range*[*dest*]: $w_1 \preceq_F w_2 \Longrightarrow$ *set* $w_1 \subseteq$ *set* w_2 **by** *force*

lemma *eq-fin-alt-def*: $w_1 =_F w_2 \longleftrightarrow w_1 \preceq_F w_2 \wedge w_2 \preceq_F w_1$
proof

show $w_1 \preceq_F w_2 \wedge w_2 \preceq_F w_1$ **if** $w_1 =_F w_2$ **using** *that* **by** *blast*
next
assume *0*: $w_1 \preceq_F w_2 \wedge w_2 \preceq_F w_1$
have *1*: $w_1 \preceq_F w_2$ $w_2 \preceq_F w_1$ **using** *0* **by** *auto*
have *10*: *length* w_1 = *length* w_2 **using** *1* **by** *force*
obtain v_1 v_2 **where** *2*: w_1 @ $v_1 =_F w_2$ w_2 @ $v_2 =_F w_1$ **using** *1* **by** (*elim le-finE*)
have *3*: *length* w_1 = *length* (w_1 @ v_1) **using** *2 10* **by** *force*
have *4*: $w_1 = w_1$ @ v_1 **using** *3* **by** *auto*
have *5*: *length* w_2 = *length* (w_2 @ v_2) **using** *2 10* **by** *force*
have *6*: $w_2 = w_2$ @ v_2 **using** *5* **by** *auto*
show $w_1 =_F w_2$ **using** *4 6 2* **by** *simp*
qed

lemma *le-fin-reflp*[*simp*, *intro*]: $w \preceq_F w$ **by** *auto*
lemma *le-fin-transp*[*intro*, *trans*]:
assumes $w_1 \preceq_F w_2$ $w_2 \preceq_F w_3$
shows $w_1 \preceq_F w_3$
proof −
obtain v_1 **where** *1*: w_1 @ $v_1 =_F w_2$ **using** *assms*(*1*) **by** *rule*
obtain v_2 **where** *2*: w_2 @ $v_2 =_F w_3$ **using** *assms*(*2*) **by** *rule*
show *?thesis*
proof
have w_1 @ v_1 @ v_2 = (w_1 @ v_1) @ v_2 **by** *simp*
also have ... $=_F w_2$ @ v_2 **using** *1* **by** *blast*
also have ... $=_F w_3$ **using** *2* **by** *blast*
finally show w_1 @ v_1 @ $v_2 =_F w_3$ **by** *this*
qed
qed
lemma *eq-fin-le-fin-transp*[*intro*, *trans*]:
assumes $w_1 =_F w_2$ $w_2 \preceq_F w_3$
shows $w_1 \preceq_F w_3$
using *assms* **by** *auto*
lemma *le-fin-eq-fin-transp*[*intro*, *trans*]:
assumes $w_1 \preceq_F w_2$ $w_2 =_F w_3$
shows $w_1 \preceq_F w_3$
using *assms* **by** *auto*
lemma *prefix-le-fin-transp*[*intro*, *trans*]:
assumes $w_1 \leq w_2$ $w_2 \preceq_F w_3$
shows $w_1 \preceq_F w_3$
proof −
obtain v_1 **where** *1*: $w_2 = w_1$ @ v_1 **using** *assms*(*1*) **by** *rule*
obtain v_2 **where** *2*: w_2 @ $v_2 =_F w_3$ **using** *assms*(*2*) **by** *rule*
show *?thesis*
proof
show w_1 @ v_1 @ $v_2 =_F w_3$ **using** *1 2* **by** *simp*
qed
qed
lemma *le-fin-prefix-transp*[*intro*, *trans*]:

assumes $w_1 \preceq_F w_2$ $w_2 \le w_3$
shows $w_1 \preceq_F w_3$
proof −
obtain v_1 **where** *1*: w_1 @ $v_1 =_F w_2$ **using** *assms*(*1*) **by** *rule*
obtain v_2 **where** *2*: $w_3 = w_2$ @ v_2 **using** *assms*(*2*) **by** *rule*
show *?thesis*
proof
have w_1 @ v_1 @ $v_2 = (w_1$ @ $v_1)$ @ v_2 **by** *simp*
also have $\ldots =_F w_2$ @ v_2 **using** *1* **by** *blast*
also have $\ldots = w_3$ **using** *2* **by** *simp*
finally show w_1 @ v_1 @ $v_2 =_F w_3$ **by** *this*
qed
qed
lemma *prefix-eq-fin-transp*[*intro*, *trans*]:
assumes $w_1 \le w_2$ $w_2 =_F w_3$
shows $w_1 \preceq_F w_3$
using *assms* **by** *auto*

lemma *le-fin-concat-start*[*iff*]: w @ $w_1 \preceq_F w$ @ $w_2 \longleftrightarrow w_1 \preceq_F w_2$
proof
assume *0*: w @ $w_1 \preceq_F w$ @ w_2
obtain v_1 **where** *1*: w @ w_1 @ $v_1 =_F w$ @ w_2 **using** *0* **by** *auto*
show $w_1 \preceq_F w_2$ **using** *1* **by** *auto*
next
assume *0*: $w_1 \preceq_F w_2$
obtain v_1 **where** *1*: w_1 @ $v_1 =_F w_2$ **using** *0* **by** *auto*
have *2*: $(w$ @ $w_1)$ @ $v_1 =_F w$ @ w_2 **using** *1* **by** *auto*
show w @ $w_1 \preceq_F w$ @ w_2 **using** *2* **by** *blast*
qed
lemma *le-fin-concat-end*[*dest*]:
assumes $w_1 \preceq_F w_2$
shows $w_1 \preceq_F w_2$ @ w
proof −
obtain v_1 **where** *1*: w_1 @ $v_1 =_F w_2$ **using** *assms* **by** *rule*
show *?thesis*
proof
have w_1 @ v_1 @ $w = (w_1$ @ $v_1)$ @ w **by** *simp*
also have $\ldots =_F w_2$ @ w **using** *1* **by** *blast*
finally show w_1 @ v_1 @ $w =_F w_2$ @ w **by** *this*
qed
qed

definition *le-fininf* :: *'item list* $\Rightarrow$ *'item stream* $\Rightarrow$ *bool* (**infix** $\preceq_{FI}$ *50*)
where $w_1 \preceq_{FI} w_2 \equiv \exists\ v_2.\ v_2 \le_{FI} w_2 \wedge w_1 \preceq_F v_2$

lemma *le-fininfI*[*intro 0*]:
assumes $v_2 \le_{FI} w_2$ $w_1 \preceq_F v_2$
shows $w_1 \preceq_{FI} w_2$
using *assms* **unfolding** *le-fininf-def* **by** *auto*

lemma *le-fininfE*[*elim 0*]:
  assumes $w_1 \preceq_{FI} w_2$
  obtains v_2
  where $v_2 \leq_{FI} w_2$ $w_1 \preceq_F v_2$
  using *assms* **unfolding** *le-fininf-def* **by** *auto*

lemma *le-fininf-empty*[*simp*]: $[] \preceq_{FI} w$ **by** *force*

lemma *le-fininf-range*[*dest*]: $w_1 \preceq_{FI} w_2 \Longrightarrow$ *set* $w_1 \subseteq$ *sset* w_2 **by** *force*

lemma *eq-fin-le-fininf-transp*[*intro*, *trans*]:
  assumes $w_1 =_F w_2$ $w_2 \preceq_{FI} w_3$
  shows $w_1 \preceq_{FI} w_3$
  using *assms* **by** *blast*
lemma *le-fin-le-fininf-transp*[*intro*, *trans*]:
  assumes $w_1 \preceq_F w_2$ $w_2 \preceq_{FI} w_3$
  shows $w_1 \preceq_{FI} w_3$
  using *assms* **by** *blast*
lemma *prefix-le-fininf-transp*[*intro*, *trans*]:
  assumes $w_1 \leq w_2$ $w_2 \preceq_{FI} w_3$
  shows $w_1 \preceq_{FI} w_3$
  using *assms* **by** *auto*
lemma *le-fin-prefix-fininf-transp*[*intro*, *trans*]:
  assumes $w_1 \preceq_F w_2$ $w_2 \leq_{FI} w_3$
  shows $w_1 \preceq_{FI} w_3$
  using *assms* **by** *auto*
lemma *eq-fin-prefix-fininf-transp*[*intro*, *trans*]:
  assumes $w_1 =_F w_2$ $w_2 \leq_{FI} w_3$
  shows $w_1 \preceq_{FI} w_3$
  using *assms* **by** *auto*

lemma *le-fininf-concat-start*[*iff*]: w @ $w_1 \preceq_{FI} w$ @− $w_2 \longleftrightarrow w_1 \preceq_{FI} w_2$
proof
  assume *0*: w @ $w_1 \preceq_{FI} w$ @− w_2
  obtain v_2 **where** *1*: $v_2 \leq_{FI} w$ @− w_2 w @ $w_1 \preceq_F v_2$ **using** *0* **by** *rule*
  have *2*: *length* $w \leq$ *length* v_2 **using** *1*(*2*) **by** *force*
  have *4*: $w \leq v_2$ **using** *prefix-fininf-extend*[*OF 1*(*1*) *2*] **by** *this*
  obtain v_1 **where** *5*: $v_2 = w$ @ v_1 **using** *4* **by** *rule*
  show $w_1 \preceq_{FI} w_2$
  proof
    show $v_1 \leq_{FI} w_2$ **using** *1*(*1*) **unfolding** *5* **by** *auto*
    show $w_1 \preceq_F v_1$ **using** *1*(*2*) **unfolding** *5* **by** *simp*
  qed
next
  assume *0*: $w_1 \preceq_{FI} w_2$
  obtain v_2 **where** *1*: $v_2 \leq_{FI} w_2$ $w_1 \preceq_F v_2$ **using** *0* **by** *rule*
  show w @ $w_1 \preceq_{FI} w$ @− w_2
  proof
    show w @ $v_2 \leq_{FI} (w$ @− $w_2)$ **using** *1*(*1*) **by** *auto*

    show w @ $w_1 \preceq_F w$ @ v_2 **using** *1*(*2*) **by** *auto*
  qed
qed

lemma *le-fininf-singleton*[*intro*, *simp*]: $[shd\ v] \preceq_{FI} v$
proof −
  have $[shd\ v] \preceq_{FI} shd\ v\ \#\#\ sdrop\ 1\ v$ **by** *blast*
  also have $\ldots = v$ **by** *simp*
  finally show *?thesis* **by** *this*
qed

definition *le-inf* :: *'item stream* ⇒ *'item stream* ⇒ *bool* (**infix** $\preceq_I$ *50*)
  where $w_1 \preceq_I w_2 \equiv \forall\ v_1.\ v_1 \leq_{FI} w_1 \longrightarrow v_1 \preceq_{FI} w_2$

lemma *le-infI*[*intro 0*]:
  assumes $\bigwedge v_1.\ v_1 \leq_{FI} w_1 \Longrightarrow v_1 \preceq_{FI} w_2$
  shows $w_1 \preceq_I w_2$
  using *assms* **unfolding** *le-inf-def* **by** *auto*
lemma *le-infE*[*elim 0*]:
  assumes $w_1 \preceq_I w_2$ $v_1 \leq_{FI} w_1$
  obtains $v_1 \preceq_{FI} w_2$
  using *assms* **unfolding** *le-inf-def* **by** *auto*

lemma *le-inf-range*[*dest*]:
  assumes $w_1 \preceq_I w_2$
  shows $sset\ w_1 \subseteq sset\ w_2$
proof
  fix a
  assume *1*: $a \in sset\ w_1$
  obtain i **where** *2*: $a = w_1\ !!\ i$ **using** *1* **by** (*metis imageE sset-range*)
  have *3*: $stake\ (Suc\ i)\ w_1 \leq_{FI} w_1$ **by** *rule*
  have *4*: $stake\ (Suc\ i)\ w_1 \preceq_{FI} w_2$ **using** *assms 3* **by** *rule*
  have *5*: $w_1\ !!\ i \in set\ (stake\ (Suc\ i)\ w_1)$ **by** (*meson lessI set-stake-snth*)
  show $a \in sset\ w_2$ **unfolding** *2* **using** *5 4* **by** *fastforce*
qed

lemma *le-inf-reflp*[*simp*, *intro*]: $w \preceq_I w$ **by** *auto*
lemma *prefix-fininf-le-inf-transp*[*intro*, *trans*]:
  assumes $w_1 \leq_{FI} w_2$ $w_2 \preceq_I w_3$
  shows $w_1 \preceq_{FI} w_3$
  using *assms* **by** *blast*
lemma *le-fininf-le-inf-transp*[*intro*, *trans*]:
  assumes $w_1 \preceq_{FI} w_2$ $w_2 \preceq_I w_3$
  shows $w_1 \preceq_{FI} w_3$
  using *assms* **by** *blast*
lemma *le-inf-transp*[*intro*, *trans*]:
  assumes $w_1 \preceq_I w_2$ $w_2 \preceq_I w_3$
  shows $w_1 \preceq_I w_3$
  using *assms* **by** *blast*

lemma *le-infI′*:
  assumes $\bigwedge k.\ \exists\ v.\ v \leq_{FI} w_1 \wedge k < \mathit{length}\ v \wedge v \preceq_{FI} w_2$
  shows $w_1 \preceq_I w_2$
proof
  fix u
  assume *1*: $u \leq_{FI} w_1$
  obtain v **where** *2*: $v \leq_{FI} w_1$ $\mathit{length}\ u < \mathit{length}\ v$ $v \preceq_{FI} w_2$ **using** *assms* **by** *auto*
  have *3*: $\mathit{length}\ u \leq \mathit{length}\ v$ **using** *2*(*2*) **by** *auto*
  have *4*: $u \leq v$ **using** *prefix-fininf-length 1 2*(*1*) *3* **by** *this*
  show $u \preceq_{FI} w_2$ **using** *4 2*(*3*) **by** *rule*
qed

lemma *le-infI-chain-left*:
  assumes *chain* w $\bigwedge k.\ w\ k \preceq_{FI} v$
  shows *limit* $w \preceq_I v$
proof (*rule le-infI′*)
  fix k
  obtain l **where** *1*: $k < \mathit{length}\ (w\ l)$ **using** *assms*(*1*) **by** *rule*
  show $\exists\ va.\ va \leq_{FI} \mathit{limit}\ w \wedge k < \mathit{length}\ va \wedge va \preceq_{FI} v$
  proof (*intro exI conjI*)
    show $w\ l \leq_{FI} \mathit{limit}\ w$ **using** *chain-prefix-limit assms*(*1*) **by** *this*
    show $k < \mathit{length}\ (w\ l)$ **using** *1* **by** *this*
    show $w\ l \preceq_{FI} v$ **using** *assms*(*2*) **by** *this*
  qed
qed
lemma *le-infI-chain-right*:
  assumes *chain* w $\bigwedge u.\ u \leq_{FI} v \Longrightarrow u \preceq_F w\ (l\ u)$
  shows $v \preceq_I \mathit{limit}\ w$
proof
  fix u
  assume *1*: $u \leq_{FI} v$
  show $u \preceq_{FI} \mathit{limit}\ w$
  proof
    show $w\ (l\ u) \leq_{FI} \mathit{limit}\ w$ **using** *chain-prefix-limit assms*(*1*) **by** *this*
    show $u \preceq_F w\ (l\ u)$ **using** *assms*(*2*) *1* **by** *this*
  qed
qed
lemma *le-infI-chain-right′*:
  assumes *chain* w $\bigwedge k.\ \mathit{stake}\ k\ v \preceq_F w\ (l\ k)$
  shows $v \preceq_I \mathit{limit}\ w$
proof (*rule le-infI-chain-right*)
  show *chain* w **using** *assms*(*1*) **by** *this*
next
  fix u
  assume *1*: $u \leq_{FI} v$
  have *2*: $\mathit{stake}\ (\mathit{length}\ u)\ v = u$ **using** *1* **by** (*simp add*: *prefix-fininf-def shift-eq*)
  have *3*: $\mathit{stake}\ (\mathit{length}\ u)\ v \preceq_F w\ (l\ (\mathit{length}\ u))$ **using** *assms*(*2*) **by** *this*

show $u \preceq_F w$ $(l\ (\mathit{length}\ u))$ **using** *3* **unfolding** *2* **by** *this*
qed

definition *eq-inf* :: *'item stream* ⇒ *'item stream* ⇒ *bool* (**infix** $=_I$ *50*)
where $w_1 =_I w_2 \equiv w_1 \preceq_I w_2 \wedge w_2 \preceq_I w_1$

lemma *eq-infI*[*intro 0*]:
assumes $w_1 \preceq_I w_2$ $w_2 \preceq_I w_1$
shows $w_1 =_I w_2$
using *assms* **unfolding** *eq-inf-def* **by** *auto*
lemma *eq-infE*[*elim 0*]:
assumes $w_1 =_I w_2$
obtains $w_1 \preceq_I w_2$ $w_2 \preceq_I w_1$
using *assms* **unfolding** *eq-inf-def* **by** *auto*

lemma *eq-inf-range*[*dest*]: $w_1 =_I w_2 \Longrightarrow \mathit{sset}\ w_1 = \mathit{sset}\ w_2$ **by** *force*

lemma *eq-inf-reflp*[*simp*, *intro*]: $w =_I w$ **by** *auto*
lemma *eq-inf-symp*[*intro*]: $w_1 =_I w_2 \Longrightarrow w_2 =_I w_1$ **by** *auto*
lemma *eq-inf-transp*[*intro*, *trans*]:
assumes $w_1 =_I w_2$ $w_2 =_I w_3$
shows $w_1 =_I w_3$
using *assms* **by** *blast*
lemma *le-fininf-eq-inf-transp*[*intro*, *trans*]:
assumes $w_1 \preceq_{FI} w_2$ $w_2 =_I w_3$
shows $w_1 \preceq_{FI} w_3$
using *assms* **by** *blast*
lemma *le-inf-eq-inf-transp*[*intro*, *trans*]:
assumes $w_1 \preceq_I w_2$ $w_2 =_I w_3$
shows $w_1 \preceq_I w_3$
using *assms* **by** *blast*
lemma *eq-inf-le-inf-transp*[*intro*, *trans*]:
assumes $w_1 =_I w_2$ $w_2 \preceq_I w_3$
shows $w_1 \preceq_I w_3$
using *assms* **by** *blast*
lemma *prefix-fininf-eq-inf-transp*[*intro*, *trans*]:
assumes $w_1 \leq_{FI} w_2$ $w_2 =_I w_3$
shows $w_1 \preceq_{FI} w_3$
using *assms* **by** *blast*

lemma *le-inf-concat-start*[*iff*]: w @− $w_1 \preceq_I w$ @− $w_2 \longleftrightarrow w_1 \preceq_I w_2$
proof
assume *1*: w @− $w_1 \preceq_I w$ @− w_2
show $w_1 \preceq_I w_2$
proof
fix v_1
assume *2*: $v_1 \leq_{FI} w_1$
have w @ $v_1 \leq_{FI} w$ @− w_1 **using** *2* **by** *auto*
also have … $\preceq_I w$ @− w_2 **using** *1* **by** *this*

finally show $v_1 \preceq_{FI} w_2$ **by** *rule*
  qed
next
  assume *1*: $w_1 \preceq_I w_2$
  show w @− $w_1 \preceq_I w$ @− w_2
  proof
    fix v_1
    assume *2*: $v_1 \leq_{FI} w$ @− w_1
    then show $v_1 \preceq_{FI} w$ @− w_2
    proof (*cases rule*: *prefix-fininf-append*)
      case (*absorb*)
      show *?thesis* **using** *absorb* **by** *auto*
    next
      case (*extend z*)
      show *?thesis* **using** *1 extend* **by** *auto*
    qed
  qed
qed
lemma *eq-fin-le-inf-concat-end*[*dest*]: $w_1 =_F w_2 \Longrightarrow w_1$ @− $w \preceq_I w_2$ @− w
proof
  fix v_1
  assume *1*: $w_1 =_F w_2$ $v_1 \leq_{FI} w_1$ @− w
  show $v_1 \preceq_{FI} w_2$ @− w
  using *1*(*2*)
  proof (*cases rule*: *prefix-fininf-append*)
    case (*absorb*)
    show *?thesis*
    proof
      show $w_2 \leq_{FI} (w_2$ @− $w)$ **by** *auto*
      show $v_1 \preceq_F w_2$ **using** *absorb 1*(*1*) **by** *auto*
    qed
  next
    case (*extend* w')
    show *?thesis*
    proof
      show w_2 @ $w' \leq_{FI} (w_2$ @− $w)$ **using** *extend*(*2*) **by** *auto*
      show $v_1 \preceq_F w_2$ @ w' **unfolding** *extend*(*1*) **using** *1*(*1*) **by** *auto*
    qed
  qed
qed

lemma *eq-inf-concat-start*[*iff*]: w @− $w_1 =_I w$ @− $w_2 \longleftrightarrow w_1 =_I w_2$ **by** *blast*
lemma *eq-inf-concat-end*[*dest*]: $w_1 =_F w_2 \Longrightarrow w_1$ @− $w =_I w_2$ @− w
proof −
  assume *0*: $w_1 =_F w_2$
  have *1*: $w_2 =_F w_1$ **using** *0* **by** *auto*
  show w_1 @− $w =_I w_2$ @− w
    using *eq-fin-le-inf-concat-end*[*OF 0*] *eq-fin-le-inf-concat-end*[*OF 1*] **by** *auto*
qed

lemma *le-fininf-suffixI*[*intro*]:
  assumes $w =_I w_1 \mathbin{@-} w_2$
  shows $w_1 \preceq_{FI} w$
  using *assms* **by** *blast*
lemma *le-fininf-suffixE*[*elim*]:
  assumes $w_1 \preceq_{FI} w$
  obtains w_2
  where $w =_I w_1 \mathbin{@-} w_2$
proof −
  obtain v_2 **where** *1*: $v_2 \leq_{FI} w$ $w_1 \preceq_F v_2$ **using** *assms*(*1*) **by** *rule*
  obtain u_1 **where** *2*: $w_1 @ u_1 =_F v_2$ **using** *1*(*2*) **by** *rule*
  obtain v_2' **where** *3*: $w = v_2 \mathbin{@-} v_2'$ **using** *1*(*1*) **by** *rule*
  show *?thesis*
  proof
    show $w =_I w_1 \mathbin{@-} u_1 \mathbin{@-} v_2'$ **unfolding** *3* **using** *2* **by** *fastforce*
  qed
qed

lemma *subsume-fin*:
  assumes $u_1 \preceq_{FI} w$ $v_1 \preceq_{FI} w$
  obtains w_1
  where $u_1 \preceq_F w_1$ $v_1 \preceq_F w_1$
proof −
  obtain u_2 **where** *2*: $u_2 \leq_{FI} w$ $u_1 \preceq_F u_2$ **using** *assms*(*1*) **by** *rule*
  obtain v_2 **where** *3*: $v_2 \leq_{FI} w$ $v_1 \preceq_F v_2$ **using** *assms*(*2*) **by** *rule*
  show *?thesis*
  proof (*cases length* u_2 *length* v_2 *rule*: *le-cases*)
    case *le*
    show *?thesis*
    proof
      show $u_1 \preceq_F v_2$ **using** *2*(*2*) *prefix-fininf-length*[*OF 2*(*1*) *3*(*1*) *le*] **by** *auto*
      show $v_1 \preceq_F v_2$ **using** *3*(*2*) **by** *this*
    qed
  next
    case *ge*
    show *?thesis*
    proof
      show $u_1 \preceq_F u_2$ **using** *2*(*2*) **by** *this*
      show $v_1 \preceq_F u_2$ **using** *3*(*2*) *prefix-fininf-length*[*OF 3*(*1*) *2*(*1*) *ge*] **by** *auto*
    qed
  qed
qed

lemma *eq-fin-end*:
  assumes $u_1 =_F u_2$ $u_1 @ v_1 =_F u_2 @ v_2$
  shows $v_1 =_F v_2$
proof −
  have $u_1 @ v_2 =_F u_2 @ v_2$ **using** *assms*(*1*) **by** *blast*

also have $\ldots =_F u_1$ @ v_1 **using** *assms(2)* **by** *blast*
finally show *?thesis* **by** *blast*
qed

definition *indoc* :: *'item* ⇒ *'item list* ⇒ *bool*
where *indoc a u* ≡ ∃ u_1 u_2. $u = u_1$ @ $[a]$ @ u_2 ∧ $a \notin$ *set* u_1 ∧ *Ind* $\{a\}$ (*set* u_1)

lemma *indoc-set*: *indoc a u* ⟹ $a \in$ *set u* **unfolding** *indoc-def* **by** *auto*

lemma *indoc-appendI1*[*intro*]:
assumes *indoc a u*
shows *indoc a* (*u* @ *v*)
using *assms* **unfolding** *indoc-def* **by** *force*
lemma *indoc-appendI2*[*intro*]:
assumes $a \notin$ *set u Ind* $\{a\}$ (*set u*) *indoc a v*
shows *indoc a* (*u* @ *v*)
proof −
obtain v_1 v_2 **where** *1*: $v = v_1$ @ $[a]$ @ v_2 $a \notin$ *set* v_1 *Ind* $\{a\}$ (*set* v_1)
using *assms(3)* **unfolding** *indoc-def* **by** *blast*
show *?thesis*
proof (*unfold indoc-def*, *intro exI conjI*)
show *u* @ $v =$ (*u* @ v_1) @ $[a]$ @ v_2 **unfolding** *1(1)* **by** *simp*
show $a \notin$ *set* (*u* @ v_1) **using** *assms(1) 1(2)* **by** *auto*
show *Ind* $\{a\}$ (*set* (*u* @ v_1)) **using** *assms(2) 1(3)* **by** *auto*
qed
qed
lemma *indoc-appendE*[*elim!*]:
assumes *indoc a* (*u* @ *v*)
obtains (*first*) $a \in$ *set u indoc a u* | (*second*) $a \notin$ *set u Ind* $\{a\}$ (*set u*) *indoc a v*
proof −
obtain w_1 w_2 **where** *1*: *u* @ $v = w_1$ @ $[a]$ @ w_2 $a \notin$ *set* w_1 *Ind* $\{a\}$ (*set* w_1)
using *assms* **unfolding** *indoc-def* **by** *blast*
show *?thesis*
proof (*cases* $a \in$ *set u*)
case *True*
obtain u_1 u_2 **where** *2*: $u = u_1$ @ $[a]$ @ u_2 $a \notin$ *set* u_1
using *split-list-first*[*OF True*] **by** *auto*
have *3*: $w_1 = u_1$
proof (*rule split-list-first-unique*)
show w_1 @ $[a]$ @ $w_2 = u_1$ @ $[a]$ @ u_2 @ *v* **using** *1(1)* **unfolding** *2(1)* **by** *simp*
show $a \notin$ *set* w_1 **using** *1(2)* **by** *auto*
show $a \notin$ *set* u_1 **using** *2(2)* **by** *this*
qed
show *?thesis*
proof (*rule first*)
show $a \in$ *set u* **using** *True* **by** *this*

```
          show indoc a u
          proof (unfold indoc-def, intro exI conjI)
            show u = u₁ @ [a] @ u₂ using 2(1) by this
            show a ∉ set u₁ using 1(2) unfolding 3 by this
            show Ind {a} (set u₁) using 1(3) unfolding 3 by this
          qed
        qed
      next
        case False
        have 2: a ∈ set v using indoc-set assms False by fastforce
        obtain v₁ v₂ where 3: v = v₁ @ [a] @ v₂ a ∉ set v₁
          using split-list-first[OF 2] by auto
        have 4: w₁ = u @ v₁
        proof (rule split-list-first-unique)
          show w₁ @ [a] @ w₂ = (u @ v₁) @ [a] @ v₂ using 1(1) unfolding 3(1)
by simp
          show a ∉ set w₁ using 1(2) by auto
          show a ∉ set (u @ v₁) using False 3(2) by auto
        qed
        show ?thesis
        proof (rule second)
          show a ∉ set u using False by this
          show Ind {a} (set u) using 1(3) 4 by auto
          show indoc a v
          proof (unfold indoc-def, intro exI conjI)
            show v = v₁ @ [a] @ v₂ using 3(1) by this
            show a ∉ set v₁ using 1(2) unfolding 4 by auto
            show Ind {a} (set v₁) using 1(3) unfolding 4 by auto
          qed
        qed
      qed
    qed

    lemma indoc-single: indoc a [b] ⟷ a = b
    proof
      assume 1: indoc a [b]
      obtain u₁ u₂ where 2: [b] = u₁ @ [a] @ u₂ Ind {a} (set u₁)
        using 1 unfolding indoc-def by auto
      show a = b using 2(1)
       by (metis append-eq-Cons-conv append-is-Nil-conv list.distinct(2) list.inject)
    next
      assume 1: a = b
      show indoc a [b]
      unfolding indoc-def 1
      proof (intro exI conjI)
        show [b] = [] @ [b] @ [] by simp
        show b ∉ set [] by simp
        show Ind {b} (set []) by simp
      qed
```

qed

lemma *indoc-append*[*simp*]: *indoc a* (*u* @ *v*) $\longleftrightarrow$
indoc a u $\vee$ *a* $\notin$ *set u* $\wedge$ *Ind* {*a*} (*set u*) $\wedge$ *indoc a v* **by** *blast*
lemma *indoc-Nil*[*simp*]: *indoc a* [] $\longleftrightarrow$ *False* **unfolding** *indoc-def* **by** *auto*
lemma *indoc-Cons*[*simp*]: *indoc a* (*b* # *v*) $\longleftrightarrow$ *a* = *b* $\vee$ *a* $\neq$ *b* $\wedge$ *ind a b* $\wedge$ *indoc a v*
proof −
have *indoc a* (*b* # *v*) $\longleftrightarrow$ *indoc a* ([*b*] @ *v*) **by** *simp*
also have ... $\longleftrightarrow$ *indoc a* [*b*] $\vee$ *a* $\notin$ *set* [*b*] $\wedge$ *Ind* {*a*} (*set* [*b*]) $\wedge$ *indoc a v*
unfolding *indoc-append* **by** *rule*
also have ... $\longleftrightarrow$ *a* = *b* $\vee$ *a* $\neq$ *b* $\wedge$ *ind a b* $\wedge$ *indoc a v* **unfolding** *indoc-single* **by** *simp*
finally show *?thesis* **by** *this*
qed

lemma *eq-swap-indoc*: *u* $=_S$ *v* $\Longrightarrow$ *indoc c u* $\Longrightarrow$ *indoc c v* **by** *auto*
lemma *eq-fin-indoc*: *u* $=_F$ *v* $\Longrightarrow$ *indoc c u* $\Longrightarrow$ *indoc c v* **by** (*induct rule*: *rtranclp.induct*, *auto*)

lemma *eq-fin-ind′*:
assumes [*a*] @ *u* $=_F$ u_1 @ [*a*] @ u_2 *a* $\notin$ *set* u_1
shows *Ind* {*a*} (*set* u_1)
proof −
have *1*: *indoc a* ([*a*] @ *u*) **by** *simp*
have *2*: *indoc a* (u_1 @ [*a*] @ u_2) **using** *eq-fin-indoc assms*(*1*) *1* **by** *this*
show *?thesis* **using** *assms*(*2*) *2* **by** *blast*
qed
lemma *eq-fin-ind*:
assumes *u* @ *v* $=_F$ *v* @ *u* *set u* $\cap$ *set v* = {}
shows *Ind* (*set u*) (*set v*)
using *assms*
proof (*induct u*)
case *Nil*
show *?case* **by** *simp*
next
case (*Cons a u*)
have *1*: *Ind* {*a*} (*set v*)
proof (*rule eq-fin-ind′*)
show [*a*] @ *u* @ *v* $=_F$ *v* @ [*a*] @ *u* **using** *Cons*(*2*) **by** *simp*
show *a* $\notin$ *set v* **using** *Cons*(*3*) **by** *simp*
qed
have *2*: *Ind* (*set* [*a*]) (*set v*) **using** *1* **by** *simp*
have *4*: *Ind* (*set u*) (*set v*)
proof (*rule Cons*(*1*))
have [*a*] @ *u* @ *v* = (*a* # *u*) @ *v* **by** *simp*
also have ... $=_F$ *v* @ *a* # *u* **using** *Cons*(*2*) **by** *this*
also have ... = (*v* @ [*a*]) @ *u* **by** *simp*
also have ... $=_F$ ([*a*] @ *v*) @ *u* **using** *2* **by** *blast*

also have $\ldots = [a]\ @\ v\ @\ u$ **by** *simp*
finally show $u\ @\ v =_F v\ @\ u$ **by** *blast*
show $set\ u \cap set\ v = \{\}$ **using** $Cons(3)$ **by** *auto*
qed
show *?case* **using** *1 4* **by** *auto*
qed

lemma *le-fin-member'*:
assumes $[a] \preceq_F u\ @\ v\ a \in set\ u$
shows $[a] \preceq_F u$
proof $-$
obtain w **where** 1: $[a]\ @\ w =_F u\ @\ v$ **using** $assms(1)$ **by** *rule*
obtain $u_1\ u_2$ **where** 2: $u = u_1\ @\ [a]\ @\ u_2\ a \notin set\ u_1$
using $split\text{-}list\text{-}first[OF\ assms(2)]$ **by** *auto*
have 3: $Ind\ \{a\}\ (set\ u_1)$
proof $(rule\ eq\text{-}fin\text{-}ind')$
show $[a]\ @\ w =_F u_1\ @\ [a]\ @\ u_2\ @\ v$ **using** *1* **unfolding** $2(1)$ **by** *simp*
show $a \notin set\ u_1$ **using** $2(2)$ **by** *this*
qed
have 4: $Ind\ (set\ [a])\ (set\ u_1)$ **using** *3* **by** *simp*
have $[a] \le [a]\ @\ u_1\ @\ u_2$ **by** *auto*
also have $\ldots = ([a]\ @\ u_1)\ @\ u_2$ **by** *simp*
also have $\ldots =_F (u_1\ @\ [a])\ @\ u_2$ **using** *4* **by** *blast*
also have $\ldots = u$ **unfolding** $2(1)$ **by** *simp*
finally show *?thesis* **by** *this*
qed
lemma *le-fin-not-member'*:
assumes $[a] \preceq_F u\ @\ v\ a \notin set\ u$
shows $[a] \preceq_F v$
proof $-$
obtain w **where** 1: $[a]\ @\ w =_F u\ @\ v$ **using** $assms(1)$ **by** *rule*
have 3: $a \in set\ v$ **using** *assms* **by** *auto*
obtain $v_1\ v_2$ **where** 4: $v = v_1\ @\ [a]\ @\ v_2\ a \notin set\ v_1$ **using** $split\text{-}list\text{-}first[OF$
$3]$ **by** *auto*
have 5: $[a]\ @\ w =_F u\ @\ v_1\ @\ [a]\ @\ v_2$ **using** *1* **unfolding** $4(1)$ **by** *this*
have 6: $Ind\ \{a\}\ (set\ (u\ @\ v_1))$
proof $(rule\ eq\text{-}fin\text{-}ind')$
show $[a]\ @\ w =_F (u\ @\ v_1)\ @\ [a]\ @\ v_2$ **using** *5* **by** *simp*
show $a \notin set\ (u\ @\ v_1)$ **using** $assms(2)\ 4(2)$ **by** *auto*
qed
have 9: $Ind\ (set\ [a])\ (set\ v_1)$ **using** *6* **by** *auto*
have $[a] \le [a]\ @\ v_1\ @\ v_2$ **by** *auto*
also have $\ldots = ([a]\ @\ v_1)\ @\ v_2$ **by** *simp*
also have $\ldots =_F (v_1\ @\ [a])\ @\ v_2$ **using** *9* **by** *blast*
also have $\ldots = v_1\ @\ [a]\ @\ v_2$ **by** *simp*
also have $\ldots = v$ **unfolding** $4(1)$ **by** *rule*
finally show *?thesis* **by** *this*
qed
lemma *le-fininf-not-member'*:

assumes $[a] \preceq_{FI} u$ @− v $a \notin$ *set* u
shows $[a] \preceq_{FI} v$
proof −
obtain v_2 **where** *1*: $v_2 \leq_{FI} u$ @− v $[a] \preceq_F v_2$ **using** *le-fininfE assms*(*1*) **by** *this*
show *?thesis*
using *1*(*1*)
proof (*cases rule*: *prefix-fininf-append*)
case *absorb*
have $[a] \preceq_F v_2$ **using** *1*(*2*) **by** *this*
also have $\ldots \leq u$ **using** *absorb* **by** *this*
finally have *2*: $a \in$ *set* u **by** *force*
show *?thesis* **using** *assms*(*2*) *2* **by** *simp*
next
case (*extend* z)
have $[a] \preceq_F v_2$ **using** *1*(*2*) **by** *this*
also have $\ldots = u$ @ z **using** *extend*(*1*) **by** *this*
finally have *2*: $[a] \preceq_F u$ @ z **by** *this*
have $[a] \preceq_F z$ **using** *le-fin-not-member′* *2* *assms*(*2*) **by** *this*
also have $\ldots \leq_{FI} v$ **using** *extend*(*2*) **by** *this*
finally show *?thesis* **by** *this*
qed
qed

lemma *le-fin-ind′′*:
assumes $[a] \preceq_F w$ $[b] \preceq_F w$ $a \neq b$
shows *ind* a b
proof −
obtain u **where** *1*: $[a]$ @ $u =_F w$ **using** *assms*(*1*) **by** *rule*
obtain v **where** *2*: $[b]$ @ $v =_F w$ **using** *assms*(*2*) **by** *rule*
have *3*: $[a]$ @ $u =_F [b]$ @ v **using** *1* *2*[*symmetric*] **by** *auto*
have *4*: $a \in$ *set* v **using** *3* *assms*(*3*)
by (*metis append-Cons append-Nil eq-fin-range list.set-intros*(*1*) *set-ConsD*)
obtain v_1 v_2 **where** *5*: $v = v_1$ @ $[a]$ @ v_2 $a \notin$ *set* v_1 **using** *split-list-first*[*OF* *4*] **by** *auto*
have *7*: *Ind* $\{a\}$ (*set* ($[b]$ @ v_1))
proof (*rule eq-fin-ind′*)
show $[a]$ @ $u =_F$ ($[b]$ @ v_1) @ $[a]$ @ v_2 **using** *3* **unfolding** *5*(*1*) **by** *simp*
show $a \notin$ *set* ($[b]$ @ v_1) **using** *assms*(*3*) *5*(*2*) **by** *auto*
qed
show *?thesis* **using** *7* **by** *auto*
qed
lemma *le-fin-ind′*:
assumes $[a] \preceq_F w$ $v \preceq_F w$ $a \notin$ *set* v
shows *Ind* $\{a\}$ (*set* v)
using *assms*
proof (*induct* v *arbitrary*: w)
case *Nil*
show *?case* **by** *simp*

next
  case (*Cons b v*)
  have *1*: *ind a b*
  proof (*rule le-fin-ind''*)
    show $[a] \preceq_F w$ **using** *Cons*(*2*) **by** *this*
    show $[b] \preceq_F w$ **using** *Cons*(*3*) **by** *auto*
    show $a \neq b$ **using** *Cons*(*4*) **by** *auto*
  qed
  obtain w' **where** *2*: $[b]$ @ $w' =_F w$ **using** *Cons*(*3*) **by** *auto*
  have *3*: *Ind* $\{a\}$ (*set v*)
  proof (*rule Cons*(*1*))
    show $[a] \preceq_F w'$
    proof (*rule le-fin-not-member'*)
      show $[a] \preceq_F [b]$ @ w' **using** *Cons*(*2*) *2* **by** *auto*
      show $a \notin$ *set* $[b]$ **using** *Cons*(*4*) **by** *auto*
    qed
    have $[b]$ @ $v = b \# v$ **by** *simp*
    also have $\ldots \preceq_F w$ **using** *Cons*(*3*) **by** *this*
    also have $\ldots =_F [b]$ @ w' **using** *2* **by** *auto*
    finally show $v \preceq_F w'$ **by** *blast*
    show $a \notin$ *set v* **using** *Cons*(*4*) **by** *auto*
  qed
  show *?case* **using** *1 3* **by** *auto*
qed
lemma *le-fininf-ind''*:
  assumes $[a] \preceq_{FI} w$ $[b] \preceq_{FI} w$ $a \neq b$
  shows *ind a b*
  using *subsume-fin le-fin-ind'' assms* **by** *metis*
lemma *le-fininf-ind'*:
  assumes $[a] \preceq_{FI} w$ $v \preceq_{FI} w$ $a \notin$ *set v*
  shows *Ind* $\{a\}$ (*set v*)
  using *subsume-fin le-fin-ind' assms* **by** *metis*

lemma *indoc-alt-def*: *indoc a v* $\longleftrightarrow v =_F [a]$ @ *remove1 a v*
proof
  assume *0*: *indoc a v*
  obtain v_1 v_2 **where** *1*: $v = v_1$ @ $[a]$ @ v_2 $a \notin$ *set* v_1 *Ind* $\{a\}$ (*set* v_1)
    using *0* **unfolding** *indoc-def* **by** *blast*
  have *2*: *Ind* (*set* $[a]$) (*set* v_1) **using** *1*(*3*) **by** *simp*
  have $v = v_1$ @ $[a]$ @ v_2 **using** *1*(*1*) **by** *this*
  also have $\ldots = (v_1$ @ $[a])$ @ v_2 **by** *simp*
  also have $\ldots =_F ([a]$ @ $v_1)$ @ v_2 **using** *2* **by** *blast*
  also have $\ldots = [a]$ @ v_1 @ v_2 **by** *simp*
  also have $\ldots = [a]$ @ *remove1 a v* **unfolding** *1*(*1*) *remove1-append* **using** *1*(*2*) **by** *auto*
  finally show $v =_F [a]$ @ *remove1 a v* **by** *this*
next
  assume *0*: $v =_F [a]$ @ *remove1 a v*
  have *1*: *indoc a* ($[a]$ @ *remove1 a v*) **by** *simp*

show *indoc a v* **using** *eq-fin-indoc 0 1* **by** *blast*
qed

lemma *levi-lemma*:
assumes t @ $u =_F v$ @ w
obtains $p\ r\ s\ q$
where $t =_F p$ @ r $u =_F s$ @ q $v =_F p$ @ s $w =_F r$ @ q *Ind* (*set* r) (*set* s)
using *assms*
proof (*induct t arbitrary*: *thesis v w*)
case *Nil*
show *?case*
proof (*rule Nil*(*1*))
show [] $=_F$ [] @ [] **by** *simp*
show $v =_F$ [] @ v **by** *simp*
show $u =_F v$ @ w **using** *Nil*(*2*) **by** *simp*
show $w =_F$ [] @ w **by** *simp*
show *Ind* (*set* []) (*set* v) **by** *simp*
qed
next
case (*Cons a* t')
have *1*: [a] $\preceq_F v$ @ w **using** *Cons*(*3*) **by** *blast*
show *?case*
proof (*cases* $a \in$ *set* v)
case *False*
have *2*: [a] $\preceq_F w$ **using** *le-fin-not-member*$'$ *1 False* **by** *this*
obtain w' **where** *3*: $w =_F$ [a] @ w' **using** *2* **by** *blast*
have *4*: $v \preceq_F v$ @ w **by** *auto*
have *5*: *Ind* (*set* [a]) (*set* v) **using** *le-fin-ind*$'$[*OF 1 4*] *False* **by** *simp*
have [a] @ t' @ u = (a # t') @ u **by** *simp*
also have … $=_F v$ @ w **using** *Cons*(*3*) **by** *this*
also have … $=_F v$ @ [a] @ w' **using** *3* **by** *blast*
also have … = (v @ [a]) @ w' **by** *simp*
also have … $=_F$ ([a] @ v) @ w' **using** *5* **by** *blast*
also have … = [a] @ v @ w' **by** *simp*
finally have *6*: t' @ $u =_F v$ @ w' **by** *blast*
obtain $p\ r'\ s\ q$ **where** *7*: $t' =_F p$ @ r' $u =_F s$ @ q $v =_F p$ @ s $w' =_F r'$
@ q
Ind (*set* r') (*set* s) **using** *Cons*(*1*)[*OF - 6*] **by** *this*
have *8*: *set* v = *set* $p \cup$ *set* s **using** *eq-fin-range 7*(*3*) **by** *auto*
have *9*: *Ind* (*set* [a]) (*set* p) **using** *5 8* **by** *auto*
have *10*: *Ind* (*set* [a]) (*set* s) **using** *5 8* **by** *auto*
show *?thesis*
proof (*rule Cons*(*2*))
have a # t' = [a] @ t' **by** *simp*
also have … $=_F$ [a] @ p @ r' **using** *7*(*1*) **by** *blast*
also have … = ([a] @ p) @ r' **by** *simp*
also have … $=_F$ (p @ [a]) @ r' **using** *9* **by** *blast*
also have … = p @ [a] @ r' **by** *simp*
finally show a # $t' =_F p$ @ [a] @ r' **by** *this*

show $u =_F s$ @ q **using** $7(2)$ **by** *this*
show $v =_F p$ @ s **using** $7(3)$ **by** *this*
have $w =_F [a]$ @ w' **using** *3* **by** *this*
also have $\ldots =_F [a]$ @ r' @ q **using** $7(4)$ **by** *blast*
also have $\ldots = ([a]$ @ $r')$ @ q **by** *simp*
finally show $w =_F ([a]$ @ $r')$ @ q **by** *this*
show *Ind* (*set* ([*a*] @ *r*′)) (*set s*) **using** $7(5)$ *10* **by** *auto*
qed
next
case *True*
have *2*: $[a] \preceq_F v$ **using** *le-fin-member′ 1 True* **by** *this*
obtain v' **where** *3*: $v =_F [a]$ @ v' **using** *2* **by** *blast*
have $[a]$ @ t' @ $u = (a \# t')$ @ u **by** *simp*
also have $\ldots =_F v$ @ w **using** *Cons*(*3*) **by** *this*
also have $\ldots =_F ([a]$ @ $v')$ @ w **using** *3* **by** *blast*
also have $\ldots = [a]$ @ v' @ w **by** *simp*
finally have *4*: t' @ $u =_F v'$ @ w **by** *blast*
obtain p' r s q **where** *7*: $t' =_F p'$ @ r $u =_F s$ @ q $v' =_F p'$ @ s $w =_F r$
@ q
Ind (*set r*) (*set s*) **using** *Cons*(*1*)[*OF* - *4*] **by** *this*
show *?thesis*
proof (*rule Cons*(*2*))
have $a \# t' = [a]$ @ t' **by** *simp*
also have $\ldots =_F [a]$ @ p' @ r **using** $7(1)$ **by** *blast*
also have $\ldots = ([a]$ @ $p')$ @ r **by** *simp*
finally show $a \# t' =_F ([a]$ @ $p')$ @ r **by** *this*
show $u =_F s$ @ q **using** $7(2)$ **by** *this*
have $v =_F [a]$ @ v' **using** *3* **by** *this*
also have $\ldots =_F [a]$ @ p' @ s **using** $7(3)$ **by** *blast*
also have $\ldots = ([a]$ @ $p')$ @ s **by** *simp*
finally show $v =_F ([a]$ @ $p')$ @ s **by** *this*
show $w =_F r$ @ q **using** $7(4)$ **by** *this*
show *Ind* (*set r*) (*set s*) **using** $7(5)$ **by** *this*
qed
qed
qed

end

end

9 Transition Systems and Trace Theory

theory *Transition-System-Traces*
imports
Transition-System-Extensions
Traces
begin

lemma (**in** *transition-system*) *words-infI-construct*[*rule-format*, *intro?*]:
  assumes $\forall\ v.\ v \leq_{FI} w \longrightarrow path\ v\ p$
  shows *run w p*
  using *assms* **by** *coinduct auto*

lemma (**in** *transition-system*) *words-infI-construct'*:
  assumes $\bigwedge k.\ \exists\ v.\ v \leq_{FI} w \wedge k < length\ v \wedge path\ v\ p$
  shows *run w p*
proof
  fix *u*
  assume *1*: $u \leq_{FI} w$
  obtain *v* **where** *2*: $v \leq_{FI} w$ *length u* $<$ *length v path v p* **using** *assms*(*1*) **by** *auto*
  have *3*: *length u* $\leq$ *length v* **using** *2*(*2*) **by** *simp*
  have *4*: $u \leq v$ **using** *prefix-fininf-length 1 2*(*1*) *3* **by** *this*
  show *path u p* **using** *4 2*(*3*) **by** *auto*
qed

lemma (**in** *transition-system*) *words-infI-construct-chain*[*intro*]:
  assumes *chain w* $\bigwedge k.$ *path* (*w k*) *p*
  shows *run* (*limit w*) *p*
proof (*rule words-infI-construct'*)
  fix *k*
  obtain *l* **where** *1*: $k <$ *length* (*w l*) **using** *assms*(*1*) **by** *rule*
  show $\exists\ v.\ v \leq_{FI} limit\ w \wedge k < length\ v \wedge path\ v\ p$
  proof (*intro exI conjI*)
    show $w\ l \leq_{FI} limit\ w$ **using** *chain-prefix-limit assms*(*1*) **by** *this*
    show $k <$ *length* (*w l*) **using** *1* **by** *this*
    show *path* (*w l*) *p* **using** *assms*(*2*) **by** *this*
  qed
qed

lemma (**in** *transition-system*) *words-fin-blocked*:
  assumes $\bigwedge w.\ path\ w\ p \Longrightarrow A \cap set\ w = \{\} \Longrightarrow A \cap \{a.\ enabled\ a\ (target\ w\ p)\} \subseteq A \cap \{a.\ enabled\ a\ p\}$
  assumes *path w p* $A \cap \{a.\ enabled\ a\ p\} \cap set\ w = \{\}$
  shows $A \cap set\ w = \{\}$
  using *assms* **by** (*induct w rule*: *rev-induct*, *auto*)

locale *transition-system-traces* =
  transition-system ex en +
  traces ind
  for *ex* :: $'action \Rightarrow\ 'state \Rightarrow\ 'state$
  and *en* :: $'action \Rightarrow\ 'state \Rightarrow bool$
  and *ind* :: $'action \Rightarrow\ 'action \Rightarrow bool$
  +
  assumes *en*: $ind\ a\ b \Longrightarrow en\ a\ p \Longrightarrow en\ b\ p \longleftrightarrow en\ b\ (ex\ a\ p)$
  assumes *ex*: $ind\ a\ b \Longrightarrow en\ a\ p \Longrightarrow en\ b\ p \Longrightarrow ex\ b\ (ex\ a\ p) = ex\ a\ (ex\ b\ p)$
begin

lemma *diamond-bottom*:
assumes *ind a b*
assumes *en a p en b p*
shows *en a* (*ex b p*) *en b* (*ex a p*) *ex b* (*ex a p*) = *ex a* (*ex b p*)
using *assms independence-symmetric en ex* **by** *metis+*
lemma *diamond-right*:
assumes *ind a b*
assumes *en a p en b* (*ex a p*)
shows *en a* (*ex b p*) *en b p ex b* (*ex a p*) = *ex a* (*ex b p*)
using *assms independence-symmetric en ex* **by** *metis+*
lemma *diamond-left*:
assumes *ind a b*
assumes *en a* (*ex b p*) *en b p*
shows *en a p en b* (*ex a p*) *ex b* (*ex a p*) = *ex a* (*ex b p*)
using *assms independence-symmetric en ex* **by** *metis+*

lemma *eq-swap-word*:
assumes $w_1 =_S w_2$ *path* w_1 *p*
shows *path* w_2 *p*
using *assms diamond-right* **by** (*induct*, *auto*)
lemma *eq-fin-word*:
assumes $w_1 =_F w_2$ *path* w_1 *p*
shows *path* w_2 *p*
using *assms eq-swap-word* **by** (*induct*, *auto*)
lemma *le-fin-word*:
assumes $w_1 \preceq_F w_2$ *path* w_2 *p*
shows *path* w_1 *p*
using *assms eq-fin-word* **by** *blast*
lemma *le-fininf-word*:
assumes $w_1 \preceq_{FI} w_2$ *run* w_2 *p*
shows *path* w_1 *p*
using *assms le-fin-word* **by** *blast*
lemma *le-inf-word*:
assumes $w_2 \preceq_I w_1$ *run* w_1 *p*
shows *run* w_2 *p*
using *assms le-fininf-word* **by** (*blast intro*: *words-infI-construct*)
lemma *eq-inf-word*:
assumes $w_1 =_I w_2$ *run* w_1 *p*
shows *run* w_2 *p*
using *assms le-inf-word* **by** *auto*

lemma *eq-swap-execute*:
assumes *path* w_1 *p* $w_1 =_S w_2$
shows *fold ex* w_1 *p* = *fold ex* w_2 *p*
using *assms*(*2*, *1*) *diamond-right* **by** (*induct*, *auto*)
lemma *eq-fin-execute*:
assumes *path* w_1 *p* $w_1 =_F w_2$
shows *fold ex* w_1 *p* = *fold ex* w_2 *p*

```
    using assms(2, 1) eq-fin-word eq-swap-execute by (induct, auto)

  lemma diamond-fin-word-step:
    assumes Ind {a} (set v) en a p path v p
    shows path v (ex a p)
    using diamond-bottom assms by (induct v arbitrary: p, auto, metis)
  lemma diamond-inf-word-step:
    assumes Ind {a} (sset w) en a p run w p
    shows run w (ex a p)
    using diamond-fin-word-step assms by (fast intro: words-infI-construct)
  lemma diamond-fin-word-inf-word:
    assumes Ind (set v) (sset w) path v p run w p
    shows run w (fold ex v p)
    using diamond-inf-word-step assms by (induct v arbitrary: p, auto)
  lemma diamond-fin-word-inf-word':
    assumes Ind (set v) (sset w) path (u @ v) p run (u @− w) p
    shows run (u @− v @− w) p
    using assms diamond-fin-word-inf-word by auto

 end

end
```

10 Functions

```
theory Functions
imports ../Extensions/Set-Extensions
begin

 locale bounded-function =
  fixes A :: 'a set
  fixes B :: 'b set
  fixes f :: 'a ⇒ 'b
  assumes wellformed[intro?, simp]: x ∈ A ⟹ f x ∈ B

 locale bounded-function-pair =
  f: bounded-function A B f +
  g: bounded-function B A g
  for A :: 'a set
  and B :: 'b set
  and f :: 'a ⇒ 'b
  and g :: 'b ⇒ 'a

 locale injection = bounded-function-pair +
  assumes left-inverse[simp]: x ∈ A ⟹ g (f x) = x
 begin

  lemma inj-on[intro]: inj-on f A using inj-onI left-inverse by metis
```

lemma *injective-on*:
assumes $x \in A$ $y \in A$ $f\,x = f\,y$
shows $x = y$
using *assms left-inverse* **by** *metis*

end

locale *injective* = *bounded-function* +
assumes *injection*: $\exists$ *g*. *injection A B f g*
begin

definition $g \equiv$ *SOME g*. *injection A B f g*

sublocale *injection A B f g* **unfolding** *g-def* **using** *someI-ex*[*OF injection*] **by** *this*

end

locale *surjection* = *bounded-function-pair* +
assumes *right-inverse*[*simp*]: $y \in B \Longrightarrow f\ (g\ y) = y$
begin

lemma *image-superset*[*intro*]: f ‘ $A \supseteq B$
using *g.wellformed image-iff right-inverse subsetI* **by** *metis*

lemma *image-eq*[*simp*]: f ‘ $A = B$ **using** *image-superset* **by** *auto*

end

locale *surjective* = *bounded-function* +
assumes *surjection*: $\exists$ *g*. *surjection A B f g*
begin

definition $g \equiv$ *SOME g*. *surjection A B f g*

sublocale *surjection A B f g* **unfolding** *g-def* **using** *someI-ex*[*OF surjection*] **by** *this*

end

locale *bijection* = *injection* + *surjection*

lemma *inj-on-bijection*:
assumes *inj-on f A*
shows *bijection A* (f ‘ A) *f* (*inv-into A f*)
proof
show $\bigwedge x.\ x \in A \Longrightarrow f\,x \in f$ ‘ A **using** *imageI* **by** *this*
show $\bigwedge y.\ y \in f$ ‘ $A \Longrightarrow$ *inv-into A f y* $\in A$ **using** *inv-into-into* **by** *this*
show $\bigwedge x.\ x \in A \Longrightarrow$ *inv-into A f* ($f\,x$) $= x$ **using** *inv-into-f-f assms* **by** *this*

show $\bigwedge y.\ y \in f\ `\ A \Longrightarrow f\ (\mathit{inv\text{-}into}\ A\ f\ y) = y$ **using** *f-inv-into-f* **by** *this*
qed

end

11 Extended Natural Numbers

theory *ENat-Extensions*
imports
Coinductive.Coinductive-Nat
begin

declare *eSuc-enat*[*simp*]
declare *iadd-Suc*[*simp*] *iadd-Suc-right*[*simp*]
declare *enat-0*[*simp*] *enat-1*[*simp*] *one-eSuc*[*simp*]
declare *enat-0-iff*[*iff*] *enat-1-iff*[*iff*]
declare *Suc-ile-eq*[*iff*]

lemma *enat-Suc0*[*simp*]: $\mathit{enat}\ (\mathit{Suc}\ 0) = \mathit{eSuc}\ 0$ **by** (*metis One-nat-def one-eSuc one-enat-def*)

lemma *le-epred*[*iff*]: $l < \mathit{epred}\ k \longleftrightarrow \mathit{eSuc}\ l < k$
by (*metis eSuc-le-iff epred-eSuc epred-le-epredI less-le-not-le not-le*)

lemma *eq-infI*[*intro*]:
assumes $\bigwedge n.\ \mathit{enat}\ n \le m$
shows $m = \infty$
using *assms* **by** (*metis enat-less-imp-le enat-ord-simps*(*5*) *less-le-not-le*)

end

12 Chain-Complete Partial Orders

theory *CCPO-Extensions*
imports
HOL−Library.Complete-Partial-Order2
ENat-Extensions
Set-Extensions
begin

lemma *chain-split*[*dest*]:
assumes *Complete-Partial-Order.chain ord C* $x \in C$
shows $C = \{y \in C.\ \mathit{ord}\ x\ y\} \cup \{y \in C.\ \mathit{ord}\ y\ x\}$
proof −
have *1*: $\bigwedge y.\ y \in C \Longrightarrow \mathit{ord}\ x\ y \lor \mathit{ord}\ y\ x$ **using** *chainD assms* **by** *this*
show *?thesis* **using** *1* **by** *blast*
qed

lemma *infinite-chain-below*[*dest*]:
  assumes *Complete-Partial-Order.chain ord C infinite C* $x \in C$
  assumes *finite* $\{y \in C.\ ord\ x\ y\}$
  shows *infinite* $\{y \in C.\ ord\ y\ x\}$
proof –
  have *1*: $C = \{y \in C.\ ord\ x\ y\} \cup \{y \in C.\ ord\ y\ x\}$ **using** *assms*(*1*, *3*) **by** *rule*
  show *?thesis* **using** *finite-Un assms*(*2*, *4*) *1* **by** (*metis* (*poly-guards-query*))
qed
lemma *infinite-chain-above*[*dest*]:
  assumes *Complete-Partial-Order.chain ord C infinite C* $x \in C$
  assumes *finite* $\{y \in C.\ ord\ y\ x\}$
  shows *infinite* $\{y \in C.\ ord\ x\ y\}$
proof –
  have *1*: $C = \{y \in C.\ ord\ x\ y\} \cup \{y \in C.\ ord\ y\ x\}$ **using** *assms*(*1*, *3*) **by** *rule*
  show *?thesis* **using** *finite-Un assms*(*2*, *4*) *1* **by** (*metis* (*poly-guards-query*))
qed

lemma (**in** *ccpo*) *ccpo-Sup-upper-inv*:
  assumes *Complete-Partial-Order.chain less-eq C* $x > \bigsqcup\ C$
  shows $x \notin C$
  using *assms ccpo-Sup-upper* **by** *fastforce*
lemma (**in** *ccpo*) *ccpo-Sup-least-inv*:
  assumes *Complete-Partial-Order.chain less-eq C* $\bigsqcup\ C > x$
  obtains y
  where $y \in C\ \neg\ y \leq x$
  using *assms ccpo-Sup-least that* **by** *fastforce*

lemma *ccpo-Sup-least-inv'*:
  fixes C :: *'a* :: {*ccpo*, *linorder*} *set*
  assumes *Complete-Partial-Order.chain less-eq C* $\bigsqcup\ C > x$
  obtains y
  where $y \in C\ y > x$
proof –
  obtain y **where** *1*: $y \in C\ \neg\ y \leq x$ **using** *ccpo-Sup-least-inv assms* **by** *this*
  show *?thesis* **using** *that 1* **by** *simp*
qed

lemma *mcont2mcont-lessThan*[*THEN lfp.mcont2mcont*, *simp*, *cont-intro*]:
  shows *mcont-lessThan*: *mcont Sup less-eq Sup less-eq*
    (*lessThan* :: *'a* :: {*ccpo*, *linorder*} ⇒ *'a set*)
proof
  show *monotone less-eq less-eq* (*lessThan* :: *'a* ⇒ *'a set*) **by** (*rule*, *auto*)
  show *cont Sup less-eq Sup less-eq* (*lessThan* :: *'a* ⇒ *'a set*)
  proof
    fix C :: *'a set*
    assume *1*: *Complete-Partial-Order.chain less-eq C*
    show $\{..< \bigsqcup\ C\} = \bigcup\ (lessThan\ `\ C)$
    proof (*intro equalityI subsetI*)
      fix A

```
      assume 2: A ∈ {..< ⨆ C}
      obtain B where 3: B ∈ C B > A using ccpo-Sup-least-inv' 1 2 by blast
      show A ∈ ⋃ (lessThan ‘ C) using 3 by auto
    next
      fix A
      assume 2: A ∈ ⋃ (lessThan ‘ C)
      show A ∈ {..< ⨆ C} using ccpo-Sup-upper 2 by force
    qed
  qed
qed

class esize =
  fixes esize :: 'a ⇒ enat

class esize-order = esize + order +
  assumes esize-finite[dest]: esize x ≠ ∞ ⟹ finite {y. y ≤ x}
  assumes esize-mono[intro]: x ≤ y ⟹ esize x ≤ esize y
  assumes esize-strict-mono[intro]: esize x ≠ ∞ ⟹ x < y ⟹ esize x < esize y
begin

  lemma infinite-chain-eSuc-esize[dest]:
    assumes Complete-Partial-Order.chain less-eq C infinite C x ∈ C
    obtains y
    where y ∈ C esize y ≥ eSuc (esize x)
  proof (cases esize x)
    case (enat k)
    have 1: finite {y ∈ C. y ≤ x} using esize-finite enat by simp
    have 2: infinite {y ∈ C. y ≥ x} using assms 1 by rule
    have 3: {y ∈ C. y > x} = {y ∈ C. y ≥ x} − {x} by auto
    have 4: infinite {y ∈ C. y > x} using 2 unfolding 3 by simp
    obtain y where 5: y ∈ C y > x using 4 by auto
    have 6: esize y > esize x using esize-strict-mono enat 5(2) by blast
    show ?thesis using that 5(1) 6 ileI1 by simp
  next
    case (infinity)
    show ?thesis using that infinity assms(3) by simp
  qed

  lemma infinite-chain-arbitrary-esize[dest]:
    assumes Complete-Partial-Order.chain less-eq C infinite C
    obtains x
    where x ∈ C esize x ≥ enat n
  proof (induct n arbitrary: thesis)
    case 0
    show ?case using assms(2) 0 by force
  next
    case (Suc n)
    obtain x where 1: x ∈ C esize x ≥ enat n using Suc(1) by blast
    obtain y where 2: y ∈ C esize y ≥ eSuc (esize x) using assms 1(1) by rule
```

```
    show ?case using gfp.leq-trans Suc(2) 1(2) 2 by fastforce
  qed

end

class esize-ccpo = esize-order + ccpo
begin

  lemma esize-cont[dest]:
    assumes Complete-Partial-Order.chain less-eq C C ≠ {}
    shows esize (⨆ C) = ⨆ (esize ‘ C)
  proof (cases finite C)
    case False
    have 1: esize (⨆ C) = ∞
    proof
      fix n
      obtain A where 1: A ∈ C esize A ≥ enat n using assms(1) False by rule
      have 2: A ≤ ⨆ C using ccpo-Sup-upper assms(1) 1(1) by this
      have enat n ≤ esize A using 1(2) by this
      also have ... ≤ esize (⨆ C) using 2 by rule
      finally show enat n ≤ esize (⨆ C) by this
    qed
    have 2: (⨆ A ∈ C. esize A) = ∞
    proof
      fix n
      obtain A where 1: A ∈ C esize A ≥ enat n using assms(1) False by rule
      show enat n ≤ (⨆ A ∈ C. esize A) using SUP-upper2 1 by this
    qed
    show ?thesis using 1 2 by simp
  next
    case True
    have 1: esize (⨆ C) = (⨆ x ∈ C. esize x)
    proof (intro order-class.order.antisym SUP-upper SUP-least esize-mono)
      show ⨆ C ∈ C using in-chain-finite assms(1) True assms(2) by this
      show ⋀ x. x ∈ C ⟹ x ≤ ⨆ C using ccpo-Sup-upper assms(1) by this
    qed
    show ?thesis using 1 by simp
  qed

  lemma esize-mcont: mcont Sup less-eq Sup less-eq esize
    by (blast intro: mcontI monotoneI contI)

  lemmas mcont2mcont-esize = esize-mcont[THEN lfp.mcont2mcont, simp, cont-intro]

end

end
```

13 Sets and Extended Natural Numbers

```
theory ESet-Extensions
imports
  ../Basics/Functions
  Basic-Extensions
  CCPO-Extensions
begin

  lemma card-lessThan-enat[simp]: card {..< enat k} = card {..< k}
  proof –
    have 1: {..< enat k} = enat ‘ {..< k}
      unfolding lessThan-def image-Collect using enat-iless by force
    have card {..< enat k} = card (enat ‘ {..< k}) unfolding 1 by rule
    also have … = card {..< k} using card-image inj-enat by metis
    finally show ?thesis by this
  qed
  lemma card-atMost-enat[simp]: card {.. enat k} = card {.. k}
  proof –
    have 1: {.. enat k} = enat ‘ {.. k}
      unfolding atMost-def image-Collect using enat-ile by force
    have card {.. enat k} = card (enat ‘ {.. k}) unfolding 1 by rule
    also have … = card {.. k} using card-image inj-enat by metis
    finally show ?thesis by this
  qed

  lemma enat-Collect:
    assumes ∞ ∉ A
    shows {i. enat i ∈ A} = the-enat ‘ A
    using assms by (safe, force) (metis enat-the-enat)

  lemma Collect-lessThan: {i. enat i < n} = the-enat ‘ {..< n}
  proof –
    have 1: ∞ ∉ {..< n} by simp
    have {i. enat i < n} = {i. enat i ∈ {..< n}} by simp
    also have … = the-enat ‘ {..< n} using enat-Collect 1 by this
    finally show ?thesis by this
  qed

  instantiation set :: (type) esize-ccpo
  begin

   function esize-set where finite A ⟹ esize A = enat (card A) | infinite A ⟹
esize A = ∞
      by auto termination by lexicographic-order

    lemma esize-iff-empty[iff]: esize A = 0 ⟷ A = {} by (cases finite A, auto)
    lemma esize-iff-infinite[iff]: esize A = ∞ ⟷ infinite A by force
    lemma esize-singleton[simp]: esize {a} = eSuc 0 by simp
```

lemma *esize-infinite-enat*[*dest*, *simp*]: *infinite* $A \Longrightarrow$ *enat* $k <$ *esize* A **by** *force*

instance
proof
  fix A :: $'a$ *set*
  assume *1*: *esize* $A \neq \infty$
  show *finite* $\{B.\ B \subseteq A\}$ **using** *1* **by** *simp*
next
  fix A B :: $'a$ *set*
  assume *1*: $A \subseteq B$
  show *esize* $A \leq$ *esize* B
  proof (*cases finite* B)
    case *False*
    show *?thesis* **using** *False* **by** *auto*
  next
    case *True*
    have *2*: *finite* A **using** *True 1* **by** *auto*
    show *?thesis* **using** *card-mono True 1 2* **by** *auto*
  qed
next
  fix A B :: $'a$ *set*
  assume *1*: *esize* $A \neq \infty$ $A \subset B$
  show *esize* $A <$ *esize* B **using** *psubset-card-mono 1* **by** (*cases finite* B, *auto*)
qed

end

lemma *esize-image*[*simp*, *intro*]:
  assumes *inj-on* f A
  shows *esize* (f ‘ A) = *esize* A
  using *card-image finite-imageD assms* **by** (*cases finite* A, *auto*)
lemma *esize-insert1*[*simp*]: $a \notin A \Longrightarrow$ *esize* (*insert* a A) = *eSuc* (*esize* A)
  by (*cases finite* A, *force*+)
lemma *esize-insert2*[*simp*]: $a \in A \Longrightarrow$ *esize* (*insert* a A) = *esize* A
  using *insert-absorb* **by** *metis*
lemma *esize-remove1*[*simp*]: $a \notin A \Longrightarrow$ *esize* ($A - \{a\}$) = *esize* A
  by (*cases finite* A, *force*+)
lemma *esize-remove2*[*simp*]: $a \in A \Longrightarrow$ *esize* ($A - \{a\}$) = *epred* (*esize* A)
  by (*cases finite* A, *force*+)
lemma *esize-union-disjoint*[*simp*]:
  assumes $A \cap B = \{\}$
  shows *esize* ($A \cup B$) = *esize* A + *esize* B
proof (*cases finite* ($A \cup B$))
  case *True*
  show *?thesis* **using** *card-Un-disjoint assms True* **by** *auto*
next
  case *False*
  show *?thesis* **using** *False* **by** (*cases finite* A, *auto*)
qed

lemma *esize-lessThan*[*simp*]: *esize* $\{..< n\} = n$
proof (*cases n*)
  case (*enat k*)
    have *1*: *finite* $\{..< n\}$ **unfolding** *enat* **by** (*metis finite-lessThan-enat-iff not-enat-eq*)
  show *?thesis* **using** *1* **unfolding** *enat* **by** *simp*
next
  case (*infinity*)
  have *1*: *infinite* $\{..< n\}$ **unfolding** *infinity* **using** *infinite-lessThan-infty* **by** *simp*
  show *?thesis* **using** *1* **unfolding** *infinity* **by** *simp*
qed
lemma *esize-atMost*[*simp*]: *esize* $\{.. n\}$ = *eSuc n*
proof (*cases n*)
  case (*enat k*)
  have *1*: *finite* $\{.. n\}$ **unfolding** *enat* **by** (*metis atMost-iff finite-enat-bounded*)
  show *?thesis* **using** *1* **unfolding** *enat* **by** *simp*
next
  case (*infinity*)
  have *1*: *infinite* $\{.. n\}$
    unfolding *infinity*
      by (*metis atMost-iff enat-ord-code*(*3*) *infinite-lessThan-infty infinite-super subsetI*)
  show *?thesis* **using** *1* **unfolding** *infinity* **by** *simp*
qed

lemma *least-eSuc*[*simp*]:
  assumes $A \neq \{\}$
  shows *least* (*eSuc* ‘ *A*) = *eSuc* (*least A*)
proof (*rule antisym*)
  obtain *k* **where** *10*: $k \in A$ **using** *assms* **by** *blast*
  have *11*: *eSuc k* $\in$ *eSuc* ‘ *A* **using** *10* **by** *auto*
  have *20*: *least A* $\in$ *A* **using** *10 LeastI* **by** *metis*
  have *21*: *least* (*eSuc* ‘ *A*) $\in$ *eSuc* ‘ *A* **using** *11 LeastI* **by** *metis*
  have *30*: $\bigwedge$ *l*. $l \in A \Longrightarrow$ *least A* $\leq l$ **using** *10 Least-le* **by** *metis*
  have *31*: $\bigwedge$ *l*. $l \in$ *eSuc* ‘ *A* $\Longrightarrow$ *least* (*eSuc* ‘ *A*) $\leq l$ **using** *11 Least-le* **by** *metis*
  show *least* (*eSuc* ‘ *A*) $\leq$ *eSuc* (*least A*) **using** *20 31* **by** *auto*
  show *eSuc* (*least A*) $\leq$ *least* (*eSuc* ‘ *A*) **using** *21 30* **by** *auto*
qed

lemma *Inf-enat-eSuc*[*simp*]: $\bigsqcap$ (*eSuc* ‘ *A*) = *eSuc* ($\bigsqcap$ *A*) **unfolding** *Inf-enat-def* **by** *simp*

definition *lift* :: *nat set* $\Rightarrow$ *nat set*
  where *lift A* $\equiv$ *insert 0* (*Suc* ‘ *A*)

lemma *liftI-0*[*intro*, *simp*]: *0* $\in$ *lift A* **unfolding** *lift-def* **by** *auto*
lemma *liftI-Suc*[*intro*]: $a \in A \Longrightarrow$ *Suc a* $\in$ *lift A* **unfolding** *lift-def* **by** *auto*
lemma *liftE*[*elim*]:

assumes $b \in \mathit{lift}\ A$
obtains $(\mathit{0})\ b = \mathit{0} \mid (\mathit{Suc})\ a$ **where** $b = \mathit{Suc}\ a\ a \in A$
using *assms* **unfolding** *lift-def* **by** *auto*

lemma *lift-esize*[*simp*]: $\mathit{esize}\ (\mathit{lift}\ A) = \mathit{eSuc}\ (\mathit{esize}\ A)$ **unfolding** *lift-def* **by** *auto*

lemma *lift-least*[*simp*]: $\mathit{least}\ (\mathit{lift}\ A) = \mathit{0}$ **unfolding** *lift-def* **by** *auto*

primrec $\mathit{nth\text{-}least} :: 'a\ set \Rightarrow nat \Rightarrow 'a :: wellorder$
where $\mathit{nth\text{-}least}\ A\ \mathit{0} = \mathit{least}\ A \mid \mathit{nth\text{-}least}\ A\ (\mathit{Suc}\ n) = \mathit{nth\text{-}least}\ (A - \{\mathit{least}\ A\})\ n$

lemma *nth-least-wellformed*[*intro?*, *simp*]:
assumes $\mathit{enat}\ n < \mathit{esize}\ A$
shows $\mathit{nth\text{-}least}\ A\ n \in A$
using *assms*
proof (*induct n arbitrary*: *A*)
case *0*
show *?case* **using** *0* **by** *simp*
next
case (*Suc n*)
have *1*: $A \neq \{\}$ **using** *Suc*(*2*) **by** *auto*
have *2*: $\mathit{enat}\ n < \mathit{esize}\ (A - \{\mathit{least}\ A\})$ **using** *Suc*(*2*) *1* **by** *simp*
have *3*: $\mathit{nth\text{-}least}\ (A - \{\mathit{least}\ A\})\ n \in A - \{\mathit{least}\ A\}$ **using** *Suc*(*1*) *2* **by** *this*
show *?case* **using** *3* **by** *simp*
qed

lemma *card-wellformed*[*intro?*, *simp*]:
fixes $k :: 'a :: wellorder$
assumes $k \in A$
shows $\mathit{enat}\ (\mathit{card}\ \{i \in A.\ i < k\}) < \mathit{esize}\ A$
proof (*cases finite A*)
case *False*
show *?thesis* **using** *False* **by** *simp*
next
case *True*
have *1*: $\mathit{esize}\ \{i \in A.\ i < k\} < \mathit{esize}\ A$ **using** *True assms* **by** *fastforce*
show *?thesis* **using** *True 1* **by** *simp*
qed

lemma *nth-least-strict-mono*:
assumes $\mathit{enat}\ l < \mathit{esize}\ A\ k < l$
shows $\mathit{nth\text{-}least}\ A\ k < \mathit{nth\text{-}least}\ A\ l$
using *assms*
proof (*induct k arbitrary*: *A l*)
case *0*
obtain l' **where** *1*: $l = \mathit{Suc}\ l'$ **using** *0*(*2*) **by** (*metis gr0-conv-Suc*)
have *2*: $A \neq \{\}$ **using** *0*(*1*) **by** *auto*
have *3*: $\mathit{enat}\ l' < \mathit{esize}\ (A - \{\mathit{least}\ A\})$ **using** *0*(*1*) *2* **unfolding** *1* **by** *simp*

```
    have 4: nth-least (A − {least A}) l′ ∈ A − {least A} using 3 by rule
    show ?case using 1 4 by (auto intro: le-neq-trans)
  next
    case (Suc k)
    obtain l′ where 1: l = Suc l′ using Suc(3) by (metis Suc-lessE)
    have 2: A ≠ {} using Suc(2) by auto
    show ?case using Suc 2 unfolding 1 by simp
  qed

  lemma nth-least-mono[intro, simp]:
    assumes enat l < esize A k ≤ l
    shows nth-least A k ≤ nth-least A l
    using nth-least-strict-mono le-less assms by metis

  lemma card-nth-least[simp]:
    assumes enat n < esize A
    shows card {k ∈ A. k < nth-least A n} = n
  using assms
  proof (induct n arbitrary: A)
    case 0
    have 1: {k ∈ A. k < least A} = {} using least-not-less by auto
    show ?case using nth-least.simps(1) card.empty 1 by metis
  next
    case (Suc n)
    have 1: A ≠ {} using Suc(2) by auto
    have 2: enat n < esize (A − {least A}) using Suc(2) 1 by simp
    have 3: nth-least A 0 < nth-least A (Suc n) using nth-least-strict-mono Suc(2)
by blast
    have 4: {k ∈ A. k < nth-least A (Suc n)} =
      {least A} ∪ {k ∈ A − {least A}. k < nth-least (A − {least A}) n} using 1 3
by auto
    have 5: card {k ∈ A − {least A}. k < nth-least (A − {least A}) n} = n using
Suc(1) 2 by this
    have 6: finite {k ∈ A − {least A}. k < nth-least (A − {least A}) n}
        using 5 Collect-empty-eq card.infinite infinite-imp-nonempty least-not-less
nth-least.simps(1)
      by (metis (no-types, lifting))
    have card {k ∈ A. k < nth-least A (Suc n)} =
      card ({least A} ∪ {k ∈ A − {least A}. k < nth-least (A − {least A}) n})
using 4 by simp
    also have . . . = card {least A} + card {k ∈ A − {least A}. k < nth-least (A
− {least A}) n}
      using 6 by simp
    also have . . . = Suc n using 5 by simp
    finally show ?case by this
  qed

  lemma card-nth-least-le[simp]:
    assumes enat n < esize A
```

shows *card* $\{k \in A.\ k \le$ *nth-least A n*$\}$ = *Suc n*
proof –
have *1*: $\{k \in A.\ k \le$ *nth-least A n*$\}$ = $\{$*nth-least A n*$\} \cup \{k \in A.\ k <$ *nth-least A n*$\}$
using *assms* **by** *auto*
have *2*: *card* $\{k \in A.\ k <$ *nth-least A n*$\}$ = *n* **using** *assms* **by** *simp*
have *3*: *finite* $\{k \in A.\ k <$ *nth-least A n*$\}$
using *2 Collect-empty-eq card.infinite infinite-imp-nonempty least-not-less nth-least.simps*(*1*)
by (*metis* (*no-types*, *lifting*))
have *card* $\{k \in A.\ k \le$ *nth-least A n*$\}$ = *card* ($\{$*nth-least A n*$\} \cup \{k \in A.\ k <$ *nth-least A n*$\}$)
unfolding *1* **by** *rule*
also have ... = *card* $\{$*nth-least A n*$\}$ + *card* $\{k \in A.\ k <$ *nth-least A n*$\}$ **using** *3* **by** *simp*
also have ... = *Suc n* **using** *assms* **by** *simp*
finally show *?thesis* **by** *this*
qed

lemma *nth-least-card*:
fixes *k* :: *nat*
assumes $k \in A$
shows *nth-least A* (*card* $\{i \in A.\ i < k\}$) = *k*
proof (*rule nat-set-card-equality-less*)
have *1*: *enat* (*card* $\{l \in A.\ l < k\}$) < *esize A*
proof (*cases finite A*)
case *False*
show *?thesis* **using** *False* **by** *simp*
next
case *True*
have *1*: $\{l \in A.\ l < k\} \subset A$ **using** *assms* **by** *blast*
have *2*: *card* $\{l \in A.\ l < k\}$ < *card A* **using** *psubset-card-mono True 1* **by** *this*
show *?thesis* **using** *True 2* **by** *simp*
qed
show *nth-least A* (*card* $\{l \in A.\ l < k\}$) $\in A$ **using** *1* **by** *rule*
show $k \in A$ **using** *assms* **by** *this*
show *card* $\{z \in A.\ z <$ *nth-least A* (*card* $\{i \in A.\ i < k\}$)$\}$ = *card* $\{z \in A.\ z < k\}$ **using** *1* **by** *simp*
qed

interpretation *nth-least*:
bounded-function-pair $\{i.$ *enat* $i <$ *esize A*$\}$ *A nth-least A* λ *k*. *card* $\{i \in A.\ i < k\}$
using *nth-least-wellformed card-wellformed* **by** (*unfold-locales*, *blast*+)

interpretation *nth-least*:
injection $\{i.$ *enat* $i <$ *esize A*$\}$ *A nth-least A* λ *k*. *card* $\{i \in A.\ i < k\}$
using *card-nth-least* **by** (*unfold-locales*, *blast*)

interpretation *nth-least*:
surjection $\{i.\ enat\ i < esize\ A\}$ *A nth-least A* $\lambda\ k.\ card\ \{i \in A.\ i < k\}$
for A :: *nat set*
using *nth-least-card* **by** (*unfold-locales*, *blast*)

interpretation *nth-least*:
bijection $\{i.\ enat\ i < esize\ A\}$ *A nth-least A* $\lambda\ k.\ card\ \{i \in A.\ i < k\}$
for A :: *nat set*
by *unfold-locales*

lemma *nth-least-strict-mono-inverse*:
fixes A :: *nat set*
assumes *enat* $k <$ *esize A enat* $l <$ *esize A nth-least A* $k <$ *nth-least A* l
shows $k < l$
using *assms* **by** (*metis not-less-iff-gr-or-eq nth-least-strict-mono*)

lemma *nth-least-less-card-less*:
fixes k :: *nat*
shows *enat* $n <$ *esize A* $\wedge$ *nth-least A* $n < k \longleftrightarrow n <$ *card* $\{i \in A.\ i < k\}$
proof *safe*
assume *1*: *enat* $n <$ *esize A nth-least A* $n < k$
have *2*: *nth-least A* $n \in A$ **using** *1*(*1*) **by** *rule*
have $n =$ *card* $\{i \in A.\ i <$ *nth-least A* $n\}$ **using** *1* **by** *simp*
also have $\ldots <$ *card* $\{i \in A.\ i < k\}$ **using** *1*(*2*) *2* **by** *simp*
finally show $n <$ *card* $\{i \in A.\ i < k\}$ **by** *this*
next
assume *1*: $n <$ *card* $\{i \in A.\ i < k\}$
have *enat* $n <$ *enat* (*card* $\{i \in A.\ i < k\}$) **using** *1* **by** *simp*
also have $\ldots =$ *esize* $\{i \in A.\ i < k\}$ **by** *simp*
also have $\ldots \leq$ *esize A* **by** *blast*
finally show *2*: *enat* $n <$ *esize A* **by** *this*
have *3*: $n =$ *card* $\{i \in A.\ i <$ *nth-least A* $n\}$ **using** *2* **by** *simp*
have *4*: *card* $\{i \in A.\ i <$ *nth-least A* $n\} <$ *card* $\{i \in A.\ i < k\}$ **using** *1 2* **by** *simp*
have *5*: *nth-least A* $n \in A$ **using** *2* **by** *rule*
show *nth-least A* $n < k$ **using** *4 5* **by** *simp*
qed

lemma *nth-least-less-esize-less*:
enat $n <$ *esize A* $\wedge$ *enat* (*nth-least A* n) $< k \longleftrightarrow$ *enat* $n <$ *esize* $\{i \in A.\ enat\ i < k\}$
using *nth-least-less-card-less* **by** (*cases* k, *simp+*)

lemma *nth-least-le*:
assumes *enat* $n <$ *esize A*
shows $n \leq$ *nth-least A* n
using *assms*
proof (*induct* n)

```
    case 0
    show ?case using 0 by simp
  next
    case (Suc n)
    have n ≤ nth-least A n using Suc by (metis Suc-ile-eq less-imp-le)
     also have … < nth-least A (Suc n) using nth-least-strict-mono Suc(2) by
blast
    finally show ?case by simp
  qed

  lemma nth-least-eq:
    assumes enat n < esize A enat n < esize B
    assumes ⋀ i. i ≤ nth-least A n ⟹ i ≤ nth-least B n ⟹ i ∈ A ⟷ i ∈ B
    shows nth-least A n = nth-least B n
  using assms
  proof (induct n arbitrary: A B)
    case 0
    have 1: least A = least B
    proof (rule least-eq)
      show A ≠ {} using 0(1) by simp
      show B ≠ {} using 0(2) by simp
    next
      fix i
      assume 2: i ≤ least A i ≤ least B
      show i ∈ A ⟷ i ∈ B using 0(3) 2 unfolding nth-least.simps by this
    qed
    show ?case using 1 by simp
  next
    case (Suc n)
    have 1: A ≠ {} B ≠ {} using Suc(2, 3) by auto
    have 2: least A = least B
    proof (rule least-eq)
      show A ≠ {} using 1(1) by this
      show B ≠ {} using 1(2) by this
    next
      fix i
      assume 3: i ≤ least A i ≤ least B
      have 4: nth-least A 0 ≤ nth-least A (Suc n) using Suc(2) by blast
      have 5: nth-least B 0 ≤ nth-least B (Suc n) using Suc(3) by blast
     have 6: i ≤ nth-least A (Suc n) i ≤ nth-least B (Suc n) using 3 4 5 by auto
      show i ∈ A ⟷ i ∈ B using Suc(4) 6 by this
    qed
    have 3: nth-least (A − {least A}) n = nth-least (B − {least B}) n
    proof (rule Suc(1))
      show enat n < esize (A − {least A}) using Suc(2) 1(1) by simp
      show enat n < esize (B − {least B}) using Suc(3) 1(2) by simp
    next
      fix i
      assume 3: i ≤ nth-least (A − {least A}) n i ≤ nth-least (B − {least B}) n
```

have *4*: $i \le$ *nth-least* A (*Suc* n) $i \le$ *nth-least* B (*Suc* n) **using** *3* **by** *simp+*
have *5*: $i \in A \longleftrightarrow i \in B$ **using** *Suc*(*4*) *4* **by** *this*
show $i \in A - \{$*least* $A\} \longleftrightarrow i \in B - \{$*least* $B\}$ **using** *2 5* **by** *auto*
qed
show *?case* **using** *3* **by** *simp*
qed

lemma *nth-least-restrict*[*simp*]:
assumes *enat* $i <$ *esize* $\{i \in s.$ *enat* $i < k\}$
shows *nth-least* $\{i \in s.$ *enat* $i < k\}$ $i =$ *nth-least* s i
proof (*rule nth-least-eq*)
show *enat* $i <$ *esize* $\{i \in s.$ *enat* $i < k\}$ **using** *assms* **by** *this*
show *enat* $i <$ *esize* s **using** *nth-least-less-esize-less assms* **by** *auto*
next
fix l
assume *1*: $l \le$ *nth-least* $\{i \in s.$ *enat* $i < k\}$ i
have *2*: *nth-least* $\{i \in s.$ *enat* $i < k\}$ $i \in \{i \in s.$ *enat* $i < k\}$ **using** *assms* **by** *rule*
have *enat* $l \le$ *enat* (*nth-least* $\{i \in s.$ *enat* $i < k\}$ i) **using** *1* **by** *simp*
also have $\ldots < k$ **using** *2* **by** *simp*
finally show $l \in \{i \in s.$ *enat* $i < k\} \longleftrightarrow l \in s$ **by** *auto*
qed

lemma *least-nth-least*[*simp*]:
assumes $A \ne \{\}$ $\bigwedge i.\ i \in A \Longrightarrow$ *enat* $i <$ *esize* B
shows *least* (*nth-least* B ‘ A) $=$ *nth-least* B (*least* A)
using *assms* **by** *simp*

lemma *nth-least-nth-least*[*simp*]:
assumes *enat* $n <$ *esize* A $\bigwedge i.\ i \in A \Longrightarrow$ *enat* $i <$ *esize* B
shows *nth-least* B (*nth-least* A n) $=$ *nth-least* (*nth-least* B ‘ A) n
using *assms*
proof (*induct n arbitrary*: A)
case *0*
show *?case* **using** *0* **by** *simp*
next
case (*Suc* n)
have *1*: $A \ne \{\}$ **using** *Suc*(*2*) **by** *auto*
have *2*: *nth-least* B ‘ ($A - \{$*least* $A\}$) $=$ *nth-least* B ‘ $A -$ *nth-least* B ‘ $\{$*least* $A\}$
proof (*rule inj-on-image-set-diff*)
show *inj-on* (*nth-least* B) $\{i.$ *enat* $i <$ *esize* $B\}$ **using** *nth-least.inj-on* **by** *this*
show $A - \{$*least* $A\} \subseteq \{i.$ *enat* $i <$ *esize* $B\}$ **using** *Suc*(*3*) **by** *blast*
show $\{$*least* $A\} \subseteq \{i.$ *enat* $i <$ *esize* $B\}$ **using** *Suc*(*3*) *1* **by** *force*
qed
have *nth-least* B (*nth-least* A (*Suc* n)) $=$ *nth-least* B (*nth-least* ($A - \{$*least* $A\}$) n) **by** *simp*
also have $\ldots =$ *nth-least* (*nth-least* B ‘ ($A - \{$*least* $A\}$)) n **using** *Suc 1* **by** *force*

also have ... = *nth-least* (*nth-least B* ' *A* − *nth-least B* ' {*least A*}) *n* **unfolding** *2* **by** *rule*
also have ... = *nth-least* (*nth-least B* ' *A* − {*nth-least B* (*least A*)}) *n* **by** *simp*
also have ... = *nth-least* (*nth-least B* ' *A* − {*least* (*nth-least B* ' *A*)}) *n* **using** *Suc*(*3*) *1* **by** *auto*
also have ... = *nth-least* (*nth-least B* ' *A*) (*Suc n*) **by** *simp*
finally show *?case* **by** *this*
qed

lemma *nth-least-Max*[*simp*]:
assumes *finite A A* ≠ {}
shows *nth-least A* (*card A* − *1*) = *Max A*
using *assms*
proof (*induct card A* − *1 arbitrary*: *A*)
case *0*
have *1*: *card A* = *1* **using** *0* **by** (*metis One-nat-def Suc-diff-1 card-gt-0-iff*)
obtain *a* **where** *2*: *A* = {*a*} **using** *1* **by** *rule*
show *?case* **unfolding** *2* **by** (*simp del*: *insert-iff*)
next
case (*Suc n*)
have *1*: *least A* ∈ *A* **using** *Suc*(*4*) **by** *rule*
have *2*: *card* (*A* − {*least A*}) = *Suc n* **using** *Suc*(*2*, *3*) *1* **by** *simp*
have *3*: *A* − {*least A*} ≠ {} **using** *2 Suc*(*3*) **by** *fastforce*
have *nth-least A* (*card A* − *1*) = *nth-least A* (*Suc n*) **unfolding** *Suc*(*2*) **by** *rule*
also have ... = *nth-least* (*A* − {*least A*}) *n* **by** *simp*
also have ... = *nth-least* (*A* − {*least A*}) (*card* (*A* − {*least A*}) − *1*) **unfolding** *2* **by** *simp*
also have ... = *Max* (*A* − {*least A*})
proof (*rule Suc*(*1*))
show *n* = *card* (*A* − {*least A*}) − *1* **unfolding** *2* **by** *simp*
show *finite* (*A* − {*least A*}) **using** *Suc*(*3*) **by** *simp*
show *A* − {*least A*} ≠ {} **using** *3* **by** *this*
qed
also have ... = *Max A* **using** *Suc*(*3*) *3* **by** *simp*
finally show *?case* **by** *this*
qed

lemma *nth-least-le-Max*:
assumes *finite A A* ≠ {} *enat n* < *esize A*
shows *nth-least A n* ≤ *Max A*
proof −
have *nth-least A n* ≤ *nth-least A* (*card A* − *1*)
proof (*rule nth-least-mono*)
show *enat* (*card A* − *1*) < *esize A* **by** (*metis Suc-diff-1 Suc-ile-eq assms*(*1*) *assms*(*2*)
card-eq-0-iff esize-set.simps(*1*) *not-gr0 order-refl*)
show *n* ≤ *card A* − *1* **by** (*metis Suc-diff-1 Suc-leI antisym-conv assms*(*1*) *assms*(*3*)

enat-ord-simps(*2*) *esize-set.simps*(*1*) *le-less neq-iff not-gr0*)
qed
also have . . . = *Max A* **using** *nth-least-Max assms*(*1*, *2*) **by** *this*
finally show *?thesis* **by** *this*
qed

lemma *nth-least-not-contains*:
fixes *k* :: *nat*
assumes *enat* (*Suc n*) < *esize A nth-least A n* < *k k* < *nth-least A* (*Suc n*)
shows *k* ∉ *A*
proof
assume *1*: *k* ∈ *A*
have *2*: *nth-least A* (*card* {*i* ∈ *A*. *i* < *k*}) = *k* **using** *nth-least.right-inverse 1* **by** *this*
have *3*: *n* < *card* {*i* ∈ *A*. *i* < *k*}
proof (*rule nth-least-strict-mono-inverse*)
show *enat n* < *esize A* **using** *assms*(*1*) **by** *auto*
show *enat* (*card* {*i* ∈ *A*. *i* < *k*}) < *esize A* **using** *nth-least.g.wellformed 1* **by** *simp*
show *nth-least A n* < *nth-least A* (*card* {*i* ∈ *A*. *i* < *k*}) **using** *assms*(*2*) *2* **by** *simp*
qed
have *4*: *card* {*i* ∈ *A*. *i* < *k*} < *Suc n*
proof (*rule nth-least-strict-mono-inverse*)
show *enat* (*card* {*i* ∈ *A*. *i* < *k*}) < *esize A* **using** *nth-least.g.wellformed 1* **by** *simp*
show *enat* (*Suc n*) < *esize A* **using** *assms*(*1*) **by** *this*
show *nth-least A* (*card* {*i* ∈ *A*. *i* < *k*}) < *nth-least A* (*Suc n*) **using** *assms*(*3*) *2* **by** *simp*
qed
show *False* **using** *3 4* **by** *auto*
qed

lemma *nth-least-Suc*[*simp*]:
assumes *enat n* < *esize A*
shows *nth-least* (*Suc* ‘ *A*) *n* = *Suc* (*nth-least A n*)
using *assms*
proof (*induct n arbitrary*: *A*)
case (*0*)
have *1*: *A* ≠ {} **using** *0* **by** *auto*
show *?case* **using** *1* **by** *simp*
next
case (*Suc n*)
have *1*: *enat n* < *esize* (*A* − {*least A*})
proof −
have *2*: *A* ≠ {} **using** *Suc*(*2*) **by** *auto*
have *3*: *least A* ∈ *A* **using** *LeastI 2* **by** *fast*
have *4*: *A* = *insert* (*least A*) *A* **using** *3* **by** *auto*
have *eSuc* (*enat n*) = *enat* (*Suc n*) **by** *simp*

also have ... < *esize A* **using** *Suc*(*2*) **by** *this*
also have ... = *esize* (*insert* (*least A*) *A*) **using** *4* **by** *simp*
also have ... = *eSuc* (*esize* (*A* − {*least A*})) **using** *3 2* **by** *simp*
finally show *?thesis* **using** *Extended-Nat.eSuc-mono* **by** *metis*
qed
have *nth-least* (*Suc* ‘ *A*) (*Suc n*) = *nth-least* (*Suc* ‘ *A* − {*least* (*Suc* ‘ *A*)}) *n*
by *simp*
also have ... = *nth-least* (*Suc* ‘ (*A* − {*least A*})) *n* **by** *simp*
also have ... = *Suc* (*nth-least* (*A* − {*least A*}) *n*) **using** *Suc*(*1*) *1* **by** *this*
also have ... = *Suc* (*nth-least A* (*Suc n*)) **by** *simp*
finally show *?case* **by** *this*
qed

lemma *nth-least-lift*[*simp*]:
nth-least (*lift A*) *0* = *0*
enat n < *esize A* ⟹ *nth-least* (*lift A*) (*Suc n*) = *Suc* (*nth-least A n*)
unfolding *lift-def* **by** *simp*+

lemma *nth-least-list-card*[*simp*]:
assumes *enat n* ≤ *esize A*
shows *card* {*k* ∈ *A*. *k* < *nth-least* (*lift A*) *n*} = *n*
using *less-Suc-eq-le assms* **by** (*cases n*, *auto simp del*: *nth-least.simps*)

end

14 Coinductive Lists

theory *Coinductive-List-Extensions*
imports
Coinductive.Coinductive-List
Coinductive.Coinductive-List-Prefix
Coinductive.Coinductive-Stream
../Extensions/List-Extensions
../Extensions/ESet-Extensions
begin

hide-const (**open**) *Sublist.prefix*
hide-const (**open**) *Sublist.suffix*

declare *list-of-lappend*[*simp*]
declare *lnth-lappend1*[*simp*]
declare *lnth-lappend2*[*simp*]
declare *lprefix-llength-le*[*dest*]
declare *Sup-llist-def*[*simp*]
declare *length-list-of*[*simp*]
declare *llast-linfinite*[*simp*]
declare *lnth-ltake*[*simp*]
declare *lappend-assoc*[*simp*]
declare *lprefix-lappend*[*simp*]

lemma *lprefix-lSup-revert*: *lSup* = *Sup lprefix* = *less-eq* **by** *auto*
lemma *admissible-lprefixI*[*cont-intro*]:
  assumes *mcont lub ord lSup lprefix f*
  assumes *mcont lub ord lSup lprefix g*
  shows *ccpo.admissible lub ord* (λ *x*. *lprefix* (*f x*) (*g x*))
  using *ccpo-class.admissible-leI assms* **unfolding** *lprefix-lSup-revert* **by** *this*
lemma *llist-lift-admissible*:
  assumes *ccpo.admissible lSup lprefix P*
  assumes ⋀ *u*. *u* ≤ *v* ⟹ *lfinite u* ⟹ *P u*
  shows *P v*
 using *assms* **by** (*metis LNil-lprefix le-llist-conv-lprefix lfinite.simps llist-gen-induct*)

abbreviation *linfinite w* ≡ ¬ *lfinite w*

notation *LNil* (<>)
notation *LCons* (**infixr** % *65*)
notation *lzip* (**infixr** || *51*)
notation *lappend* (**infixr** $ *65*)
notation *lnth* (**infixl** ?! *100*)

syntax *-llist* :: *args* ⇒ ′*a llist* (<->)
translations
  <*a*, *x*> ⇌ *a* % <*x*>
  <*a*> ⇌ *a* % <>

lemma *eq-LNil-conv-lnull*[*simp*]: *w* = <> ⟷ *lnull w* **by** *auto*
lemma *Collect-lnull*[*simp*]: {*w*. *lnull w*} = {<>} **by** *auto*

lemma *inj-on-ltake*: *inj-on* (λ *k*. *ltake k w*) {.. *llength w*}
  by (*rule inj-onI*, *auto*, *metis llength-ltake min-def*)

lemma *lnth-inf-llist*′[*simp*]: *lnth* (*inf-llist f*) = *f* **by** *auto*

lemma *not-lnull-lappend-startE*[*elim*]:
  assumes ¬ *lnull w*
  obtains *a v*
  where *w* = <*a*> $ *v*
  using *not-lnull-conv assms* **by** (*simp*, *metis*)
lemma *not-lnull-lappend-endE*[*elim*]:
  assumes ¬ *lnull w*
  obtains *a v*
  where *w* = *v* $ <*a*>
proof (*cases lfinite w*)
  case *False*
  show *?thesis*
  proof
    show *w* = *w* $ <*a*> **using** *lappend-inf False* **by** *force*
  qed

```
next
  case True
  show ?thesis
  using True assms that
  proof (induct arbitrary: thesis)
    case (lfinite-LNil)
    show ?case using lfinite-LNil by auto
  next
    case (lfinite-LConsI w a)
    show ?case
    proof (cases lnull w)
      case False
      obtain b v where 1: w = v $ <b> using lfinite-LConsI(2) False by this
      show ?thesis
      proof (rule lfinite-LConsI(4))
        show a % w = (a % v) $ <b> unfolding 1 by simp
      qed
    next
      case True
      show ?thesis
      proof (rule lfinite-LConsI(4))
        show a % w = <> $ <a> using True by simp
      qed
    qed
  qed
qed

lemma llength-lappend-startE[elim]:
  assumes llength w ≥ eSuc n
  obtains a v
  where w = <a> $ v llength v ≥ n
proof -
  have 1: ¬ lnull w using assms by auto
  show ?thesis using assms 1 that by auto
qed
lemma llength-lappend-endE[elim]:
  assumes llength w ≥ eSuc n
  obtains a v
  where w = v $ <a> llength v ≥ n
proof -
  have 1: ¬ lnull w using assms by auto
  show ?thesis using assms 1 that by auto
qed

lemma llength-lappend-start'E[elim]:
  assumes llength w = enat (Suc n)
  obtains a v
  where w = <a> $ v llength v = enat n
proof -
```

have *1*: *llength w* ≥ *eSuc* (*enat n*) **using** *assms* **by** *simp*
obtain *a v* **where** *2*: *w* = <*a*> $ *v* **using** *1* **by** *blast*
show *?thesis*
proof
show *w* = <*a*> $ *v* **using** *2*(*1*) **by** *this*
show *llength v* = *enat n* **using** *assms* **unfolding** *2*(*1*) **by** (*simp*, *metis eSuc-enat eSuc-inject*)
qed
qed
lemma *llength-lappend-end′E*[*elim*]:
assumes *llength w* = *enat* (*Suc n*)
obtains *a v*
where *w* = *v* $ <*a*> *llength v* = *enat n*
proof −
have *1*: *llength w* ≥ *eSuc* (*enat n*) **using** *assms* **by** *simp*
obtain *a v* **where** *2*: *w* = *v* $ <*a*> **using** *1* **by** *blast*
show *?thesis*
proof
show *w* = *v* $ <*a*> **using** *2*(*1*) **by** *this*
show *llength v* = *enat n* **using** *assms* **unfolding** *2*(*1*) **by** (*simp*, *metis eSuc-enat eSuc-inject*)
qed
qed

lemma *ltake-llast*[*simp*]:
assumes *enat k* < *llength w*
shows *llast* (*ltake* (*enat* (*Suc k*)) *w*) = *w ?! k*
proof −
have *1*: *llength* (*ltake* (*enat* (*Suc k*)) *w*) = *eSuc* (*enat k*)**using** *min.absorb-iff1 assms* **by** *auto*
have *llast* (*ltake* (*enat* (*Suc k*)) *w*) = *ltake* (*enat* (*Suc k*)) *w ?! k*
using *llast-conv-lnth 1* **by** *this*
also have … = *w ?! k* **by** (*rule lnth-ltake*, *simp*)
finally show *?thesis* **by** *this*
qed

lemma *linfinite-llength*[*dest*, *simp*]:
assumes *linfinite w*
shows *enat k* < *llength w*
using *assms not-lfinite-llength* **by** *force*

lemma *llist-nth-eqI*[*intro*]:
assumes *llength u* = *llength v*
assumes ⋀ *i*. *enat i* < *llength u* ⟹ *enat i* < *llength v* ⟹ *u ?! i* = *v ?! i*
shows *u* = *v*
using *assms*
proof (*coinduction arbitrary*: *u v*)
case *Eq-llist*
have *10*: *llength u* = *llength v* **using** *Eq-llist* **by** *auto*

```
  have 11: ⋀ i. enat i < llength u ⟹  enat i < llength v ⟹ u ?! i = v ?! i
    using Eq-llist by auto
  show ?case
  proof (intro conjI impI exI allI)
    show lnull u ⟷ lnull v using 10 by auto
  next
    assume 20: ¬ lnull u ¬ lnull v
    show lhd u = lhd v using lhd-conv-lnth enat-0 11 20 by force
  next
    show ltl u = ltl u by rule
  next
    show ltl v = ltl v by rule
  next
    assume 30: ¬ lnull u ¬ lnull v
    show llength (ltl u) = llength (ltl v) using 10 30 by force
  next
    fix i
    assume 40: ¬ lnull u ¬ lnull v enat i < llength (ltl u) enat i < llength (ltl v)
    have 41: u ?! Suc i = v ?! Suc i
    proof (rule 11)
      show enat (Suc i) < llength u using Suc-ile-eq 40(1) 40(3) by auto
      show enat (Suc i) < llength v using Suc-ile-eq 40(2) 40(4) by auto
    qed
    show ltl u ?! i = ltl v ?! i using lnth-ltl 40(1−2) 41 by metis
  qed
qed

primcorec lscan :: ('a ⇒ 'b ⇒ 'b) ⇒ 'a llist ⇒ 'b ⇒ 'b llist
  where lscan f w a = (case w of <> ⇒ <a> | x % xs ⇒ a % lscan f xs (f x a))

lemma lscan-simps[simp]:
  lscan f <> a = <a>
  lscan f (x % xs) a = a % lscan f xs (f x a)
  by (metis llist.simps(4) lscan.code, metis llist.simps(5) lscan.code)

lemma lscan-lfinite[iff]: lfinite (lscan f w a) ⟷ lfinite w
proof
  assume lfinite (lscan f w a)
  thus lfinite w
  proof (induct lscan f w a arbitrary: w a rule: lfinite-induct)
    case LNil
    show ?case using LNil by simp
  next
    case LCons
    show ?case by (cases w, simp, simp add: LCons(3))
  qed
next
  assume lfinite w
  thus lfinite (lscan f w a) by (induct arbitrary: a, auto)
```

```
qed
lemma lscan-llength[simp]: llength (lscan f w a) = eSuc (llength w)
proof (cases lfinite w)
  case False
  have 1: llength (lscan f w a) = ∞ using not-lfinite-llength False by auto
  have 2: llength w = ∞ using not-lfinite-llength False by auto
  show ?thesis using 1 2 by simp
next
  case True
  show ?thesis using True by (induct arbitrary: a, auto)
qed

function lfold :: ('a ⇒ 'b ⇒ 'b) ⇒ 'a llist ⇒ 'b ⇒ 'b
 where lfinite w ⟹ lfold f w = fold f (list-of w) | linfinite w ⟹ lfold f w = id
  by (auto, metis) termination by lexicographic-order

lemma lfold-llist-of[simp]: lfold f (llist-of xs) = fold f xs by simp

lemma finite-UNIV-llength-eq:
  assumes finite (UNIV :: 'a set)
  shows finite {w :: 'a llist. llength w = enat n}
proof (induct n)
  case (0)
  show ?case by simp
next
  case (Suc n)
  have 1: finite ({v. llength v = enat n} × UNIV :: ('a llist × 'a) set)
    using Suc assms by simp
   have 2: finite ((λ (v, a). v $ <a> :: 'a llist ) ‘ ({v. llength v = enat n} ×
UNIV))
    using 1 by auto
  have 3: finite {v $ <a> :: 'a llist |v a. llength v = enat n}
  proof –
    have 0: {v $ <a> :: 'a llist  |v a. llength v = enat n} =
      (λ (v, a). v $ <a> :: 'a llist ) ‘ ({v. llength v = enat n} × UNIV) by auto
    show ?thesis using 2 unfolding 0 by this
  qed
  have 4: finite {w :: 'a llist . llength w = enat (Suc n)}
  proof –
    have 0: {w :: 'a llist . llength w = enat (Suc n)} =
      {v $ <a> :: 'a llist  |v a. llength v = enat n} by force
    show ?thesis using 3 unfolding 0 by this
  qed
  show ?case using 4 by this
qed
lemma finite-UNIV-llength-le:
  assumes finite (UNIV :: 'a set)
  shows finite {w :: 'a llist. llength w ≤ enat n}
proof –
```

have *1*: $\{w.\ llength\ w \leq enat\ n\} = (\bigcup\ k \leq n.\ \{w.\ llength\ w = enat\ k\})$
by (*auto*, *metis atMost-iff enat-ile enat-ord-simps*(*1*))
show *?thesis* **unfolding** *1* **using** *finite-UNIV-llength-eq assms* **by** *auto*
qed

lemma *lprefix-ltake*[*dest*]: $u \leq v \Longrightarrow u = ltake\ (llength\ u)\ v$
by (*metis le-llist-conv-lprefix lprefix-conv-lappend ltake-all ltake-lappend1 order-refl*)
lemma *prefixes-set*: $\{v.\ v \leq w\} = \{ltake\ k\ w\ |k.\ k \leq llength\ w\}$ **by** *fastforce*
lemma *esize-prefixes*[*simp*]: $esize\ \{v.\ v \leq w\} = eSuc\ (llength\ w)$
proof −
have $esize\ \{v.\ v \leq w\} = esize\ \{ltake\ k\ w\ |k.\ k \leq llength\ w\}$ **unfolding** *prefixes-set* **by** *rule*
also have $\ldots = esize\ ((\lambda\ k.\ ltake\ k\ w)\ `\ \{..\ llength\ w\})$
unfolding *atMost-def image-Collect* **by** *rule*
also have $\ldots = esize\ \{..\ llength\ w\}$ **using** *inj-on-ltake esize-image* **by** *blast*
also have $\ldots = eSuc\ (llength\ w)$ **by** *simp*
finally show *?thesis* **by** *this*
qed
lemma *prefix-subsume*: $v \leq w \Longrightarrow u \leq w \Longrightarrow llength\ v \leq llength\ u \Longrightarrow v \leq u$
by (*metis le-llist-conv-lprefix lprefix-conv-lappend*
lprefix-ltake ltake-is-lprefix ltake-lappend1)

lemma *ltake-infinite*[*simp*]: $ltake\ \infty\ w = w$ **by** (*metis enat-ord-code*(*3*) *ltake-all*)

lemma *lprefix-infinite*:
assumes $u \leq v$ *linfinite u*
shows $u = v$
proof −
have *1*: $llength\ u = \infty$ **using** *not-lfinite-llength assms*(*2*) **by** *this*
have $u = ltake\ (llength\ u)\ v$ **using** *lprefix-ltake assms*(*1*) **by** *this*
also have $\ldots = v$ **using** *1* **by** *simp*
finally show *?thesis* **by** *this*
qed

instantiation *llist* :: (*type*) *esize-order*
begin

definition [*simp*]: $esize \equiv llength$

instance
proof
fix w :: *'a llist*
assume *1*: $esize\ w \neq \infty$
show *finite* $\{v.\ v \leq w\}$
using *esize-prefixes 1* **by** (*metis eSuc-eq-infinity-iff esize-set.simps*(*2*) *esize-llist-def*)
next
fix $u\ v$ :: *'a llist*

assume *1*: $u \leq v$
show *esize* $u \leq$ *esize* v **using** *lprefix-llength-le 1* **by** *auto*
next
fix $u\ v$ :: *'a llist*
assume *1*: $u < v$
show *esize* $u <$ *esize* v **using** *lstrict-prefix-llength-less 1* **by** *auto*
qed

end

14.1 Index Sets

definition *liset* :: *'a set* ⇒ *'a llist* ⇒ *nat set*
where *liset A w* ≡ {*i. enat i* < *llength w* ∧ *w ?! i* ∈ *A*}

lemma *lisetI*[*intro*]:
assumes *enat i* < *llength w w ?! i* ∈ *A*
shows *i* ∈ *liset A w*
using *assms* **unfolding** *liset-def* **by** *auto*
lemma *lisetD*[*dest*]:
assumes *i* ∈ *liset A w*
shows *enat i* < *llength w w ?! i* ∈ *A*
using *assms* **unfolding** *liset-def* **by** *auto*

lemma *liset-finite*:
assumes *lfinite w*
shows *finite* (*liset A w*)
proof
show *liset A w* ⊆ {*i. enat i* < *llength w*} **by** *auto*
show *finite* {*i. enat i* < *llength w*} **using** *lfinite-finite-index assms* **by** *this*
qed

lemma *liset-nil*[*simp*]: *liset A* <> = {} **by** *auto*
lemma *liset-cons-not-member*[*simp*]:
assumes *a* ∉ *A*
shows *liset A* (*a* % *w*) = *Suc* ‘ *liset A w*
proof −
have *liset A* (*a* % *w*) = {*i. enat i* < *llength* (*a* % *w*) ∧ (*a* % *w*) *?! i* ∈ *A*} **by** *auto*
also have ... = *Suc* ‘ {*i. enat* (*Suc i*) < *llength* (*a* % *w*) ∧ (*a* % *w*) *?! Suc i* ∈ *A*}
using *Collect-split-Suc*(*1*) *assms* **by** *simp*
also have ... = *Suc* ‘ {*i. enat i* < *llength w* ∧ *w ?! i* ∈ *A*} **using** *Suc-ile-eq* **by** *simp*
also have ... = *Suc* ‘ *liset A w* **by** *auto*
finally show *?thesis* **by** *this*
qed
lemma *liset-cons-member*[*simp*]:
assumes *a* ∈ *A*

shows *liset A* (*a* % *w*) = {*0*} ∪ *Suc* ' *liset A w*
proof −
have *liset A* (*a* % *w*) = {*i*. *enat i* < *llength* (*a* % *w*) ∧ (*a* % *w*) *?! i* ∈ *A*} **by** *auto*
also have ... = {*0*} ∪ *Suc* ' {*i*. *enat* (*Suc i*) < *llength* (*a* % *w*) ∧ (*a* % *w*) *?! Suc i* ∈ *A*}
using *Collect-split-Suc*(*2*) *assms* **by** *simp*
also have ... = {*0*} ∪ *Suc* ' {*i*. *enat i* < *llength w* ∧ *w ?! i* ∈ *A*} **using** *Suc-ile-eq* **by** *simp*
also have ... = {*0*} ∪ *Suc* ' *liset A w* **by** *auto*
finally show *?thesis* **by** *this*
qed

lemma *liset-prefix*:
assumes *i* ∈ *liset A v u* ≤ *v enat i* < *llength u*
shows *i* ∈ *liset A u*
unfolding *liset-def*
proof (*intro CollectI conjI*)
have *1*: *v ?! i* ∈ *A* **using** *assms*(*1*) **by** *auto*
show *enat i* < *llength u* **using** *assms*(*3*) **by** *this*
show *u ?! i* ∈ *A* **using** *lprefix-lnthD assms*(*2*, *3*) *1* **by** *force*
qed
lemma *liset-suffix*:
assumes *i* ∈ *liset A u u* ≤ *v*
shows *i* ∈ *liset A v*
unfolding *liset-def*
proof (*intro CollectI conjI*)
have *1*: *enat i* < *llength u u ?! i* ∈ *A* **using** *assms*(*1*) **by** *auto*
show *enat i* < *llength v* **using** *lprefix-llength-le 1*(*1*) *assms*(*2*) **by** *fastforce*
show *v ?! i* ∈ *A* **using** *lprefix-lnthD assms*(*2*) *1* **by** *force*
qed

lemma *liset-ltake*[*simp*]: *liset A* (*ltake* (*enat k*) *w*) = *liset A w* ∩ {..< *k*}
proof (*intro equalityI subsetI*)
fix *i*
assume *1*: *i* ∈ *liset A* (*ltake* (*enat k*) *w*)
have *2*: *enat i* < *enat k* **using** *1* **by** *auto*
have *3*: *ltake* (*enat k*) *w ?! i* = *w ?! i* **using** *lnth-ltake 2* **by** *this*
show *i* ∈ *liset A w* ∩ {..< *k*} **using** *1 3* **by** *fastforce*
next
fix *i*
assume *1*: *i* ∈ *liset A w* ∩ {..< *k*}
have *2*: *enat i* < *enat k* **using** *1* **by** *auto*
have *3*: *ltake* (*enat k*) *w ?! i* = *w ?! i* **using** *lnth-ltake 2* **by** *this*
show *i* ∈ *liset A* (*ltake* (*enat k*) *w*) **using** *1 3* **by** *fastforce*
qed

lemma *liset-mono*[*dest*]: *u* ≤ *v* ⟹ *liset A u* ⊆ *liset A v*
unfolding *liset-def* **using** *lprefix-lnthD* **by** *fastforce*

lemma *liset-cont*[*dest*]:
assumes *Complete-Partial-Order.chain less-eq C* $C \neq \{\}$
shows *liset A* $(\bigsqcup\ C) = (\bigcup\ w \in C.$ *liset A w*$)$
proof *safe*
fix *i*
assume *1*: $i \in$ *liset A* $(\bigsqcup\ C)$
show $i \in (\bigcup\ w \in C.$ *liset A w*$)$
proof (*cases finite C*)
case *False*
obtain *w* **where** *2*: $w \in C$ *enat i* $<$ *llength w*
using *esize-llist-def infinite-chain-arbitrary-esize assms*(*1*) *False Suc-ile-eq*
by *metis*
have *3*: $w \leq \bigsqcup\ C$ **using** *chain-lprefix-lSup assms*(*1*) *2*(*1*) **by** *simp*
have *4*: $i \in$ *liset A w* **using** *liset-prefix 1 3 2*(*2*) **by** *this*
show *?thesis* **using** *2*(*1*) *4* **by** *auto*
next
case *True*
have *2*: $\bigsqcup\ C \in C$ **using** *in-chain-finite assms*(*1*) *True assms*(*2*) **by** *this*
show *?thesis* **using** *1 2* **by** *auto*
qed
next
fix *w i*
assume *1*: $w \in C$ $i \in$ *liset A w*
have *2*: $w \leq \bigsqcup\ C$ **using** *chain-lprefix-lSup assms*(*1*) *1*(*1*) **by** *simp*
show $i \in$ *liset A* $(\bigsqcup\ C)$ **using** *liset-suffix 1*(*2*) *2* **by** *this*
qed

lemma *liset-mcont*: *Complete-Partial-Order2.mcont lSup lprefix Sup less-eq*
(*liset A*)
unfolding *lprefix-lSup-revert* **by** (*blast intro*: *mcontI monotoneI contI*)

lemmas *mcont2mcont-liset* = *liset-mcont*[*THEN lfp.mcont2mcont*, *simp*, *cont-intro*]

14.2 Selections

abbreviation *lproject A* $\equiv$ *lfilter* $(\lambda\ a.\ a \in A)$
abbreviation *lselect s w* $\equiv$ *lnths w s*

lemma *lselect-to-lproject*: *lselect s w* = *lmap fst* (*lproject* (*UNIV* $\times$ *s*) (*w* || *iterates Suc 0*))
proof −
have *1*: $\{(x, y).\ y \in s\}$ = *UNIV* $\times$ *s* **by** *auto*
have *lselect s w* = *lmap fst* (*lproject* $\{(x, y).\ y \in s\}$ (*w* || *iterates Suc 0*))
unfolding *lnths-def* **by** *simp*
also have . . . = *lmap fst* (*lproject* (*UNIV* $\times$ *s*) (*w* || *iterates Suc 0*)) **unfolding**
1 **by** *rule*
finally show *?thesis* **by** *this*
qed
lemma *lproject-to-lselect*: *lproject A w* = *lselect* (*liset A w*) *w*

unfolding *lfilter-conv-lnths liset-def* **by** *rule*

lemma *lproject-llength*[*simp*]: *llength* (*lproject A w*) = *esize* (*liset A w*)
by (*induct rule*: *llist-induct*) (*auto*)
lemma *lproject-lfinite*[*simp*]: *lfinite* (*lproject A w*) ⟷ *finite* (*liset A w*)
using *lproject-llength esize-iff-infinite llength-eq-infty-conv-lfinite* **by** *metis*

lemma *lselect-restrict-indices*[*simp*]: *lselect* {*i* ∈ *s*. *enat i* < *llength w*} *w* = *lselect s w*
proof (*rule lnths-cong*)
show *w* = *w* **by** *rule*
next
fix *n*
assume *1*: *enat n* < *llength w*
show *n* ∈ {*i* ∈ *s*. *enat i* < *llength w*} ⟷ *n* ∈ *s* **using** *1* **by** *blast*
qed

lemma *lselect-llength*: *llength* (*lselect s w*) = *esize* {*i* ∈ *s*. *enat i* < *llength w*}
proof −
have *1*: ⋀ *i*. *enat i* < *llength w* ⟹ (*w* || *iterates Suc 0*) ?! *i* = (*w* ?! *i*, *i*)
by (*metis Suc-funpow enat.distinct*(*1*) *enat-ord-simps*(*4*) *llength-iterates lnth-iterates*
lnth-lzip monoid-add-class.add.right-neutral)
have *2*: {*i*. *enat i* < *llength w* ∧ (*w* || *iterates Suc 0*) ?! *i* ∈ *UNIV* × *s*} =
{*i* ∈ *s*. *enat i* < *llength w*} **using** *1* **by** *auto*
have *llength* (*lselect s w*) = *esize* (*liset* (*UNIV* × *s*) (*w* || *iterates Suc 0*))
unfolding *lselect-to-lproject* **by** *simp*
also have . . . = *esize* {*i*. *enat i* < *llength w* ∧ (*w* || *iterates Suc 0*) ?! *i* ∈ *UNIV* × *s*}
unfolding *liset-def* **by** *simp*
also have . . . = *esize* {*i* ∈ *s*. *enat i* < *llength w*} **unfolding** *2* **by** *rule*
finally show *?thesis* **by** *this*
qed
lemma *lselect-llength-le*[*simp*]: *llength* (*lselect s w*) ≤ *esize s*
proof −
have *llength* (*lselect s w*) = *esize* {*i* ∈ *s*. *enat i* < *llength w*}
unfolding *lselect-llength* **by** *rule*
also have . . . = *esize* (*s* ∩ {*i*. *enat i* < *llength w*}) **unfolding** *Collect-conj-eq* **by** *simp*
also have . . . ≤ *esize s* **by** *blast*
finally show *?thesis* **by** *this*
qed
lemma *least-lselect-llength*:
assumes ¬ *lnull* (*lselect s w*)
shows *enat* (*least s*) < *llength w*
proof −
have *0*: *llength* (*lselect s w*) > *0* **using** *assms* **by** *auto*
have *1*: ⋀ *i*. *i* ∈ *s* ⟹ *least s* ≤ *i* **using** *Least-le 0* **by** *fast*
obtain *i* **where** *2*: *i* ∈ *s enat i* < *llength w* **using** *0* **unfolding** *lselect-llength*

by *auto*
have *enat* (*least s*) $\leq$ *enat i* **using** *1 2*(*1*) **by** *auto*
also have ... < *llength w* **using** *2*(*2*) **by** *this*
finally show *enat* (*least s*) < *llength w* **by** *this*
qed
lemma *lselect-lnull*: *lnull* (*lselect s w*) $\longleftrightarrow$ ($\forall$ *i* $\in$ *s*. *enat i* $\geq$ *llength w*)
unfolding *llength-eq-0*[*symmetric*] *lselect-llength* **by** *auto*

lemma *lselect-discard-start*:
assumes $\bigwedge$ *i*. *i* $\in$ *s* $\Longrightarrow$ *k* $\leq$ *i*
shows *lselect* {*i*. *k* + *i* $\in$ *s*} (*ldropn k w*) = *lselect s w*
proof −
have *1*: *lselect s* (*ltake* (*enat k*) *w*) = <>
using *assms* **by** (*fastforce simp add*: *lselect-lnull min-le-iff-disj*)
have *lselect* {*m*. *k* + *m* $\in$ *s*} (*ldropn k w*) =
lselect s (*ltake* (*enat k*) *w*) $ *lselect* {*m*. *k* + *m* $\in$ *s*} (*ldropn k w*) **unfolding** *1* **by** *simp*
also have ... = *lselect s w* **using** *lnths-split* **by** *rule*
finally show *?thesis* **by** *this*
qed
lemma *lselect-discard-end*:
assumes $\bigwedge$ *i*. *i* $\in$ *s* $\Longrightarrow$ *i* < *k*
shows *lselect s* (*ltake* (*enat k*) *w*) = *lselect s w*
proof −
have *1*: *lselect* {*m*. *k* + *m* $\in$ *s*} (*ldropn k w*) = <>
using *assms* **by** (*fastforce simp add*: *lselect-lnull min-le-iff-disj*)
have *lselect s* (*ltake* (*enat k*) *w*) =
lselect s (*ltake* (*enat k*) *w*) $ *lselect* {*m*. *k* + *m* $\in$ *s*} (*ldropn k w*) **unfolding** *1* **by** *simp*
also have ... = *lselect s w* **using** *lnths-split* **by** *rule*
finally show *?thesis* **by** *this*
qed

lemma *lselect-least*:
assumes $\neg$ *lnull* (*lselect s w*)
shows *lselect s w* = *w* ?! *least s* % *lselect* (*s* − {*least s*}) *w*
proof −
have *0*: *s* $\neq$ {} **using** *assms* **by** *auto*
have *1*: *least s* $\in$ *s* **using** *LeastI 0* **by** *fast*
have *2*: $\bigwedge$ *i*. *i* $\in$ *s* $\Longrightarrow$ *least s* $\leq$ *i* **using** *Least-le 0* **by** *fast*
have *3*: $\bigwedge$ *i*. *i* $\in$ *s* − {*least s*} $\Longrightarrow$ *Suc* (*least s*) $\leq$ *i* **using** *least-unique 2* **by** *force*
have *4*: *insert* (*least s*) (*s* − {*least s*}) = *s* **using** *1* **by** *auto*
have *5*: *enat* (*least s*) < *llength w* **using** *least-lselect-llength assms* **by** *this*
have *6*: *lselect* (*s* − {*least s*}) (*ltake* (*enat* (*least s*)) *w*) = <>
by (*rule*, *auto simp*: *lselect-llength dest*: *least-not-less*)
have *7*: *lselect* {*i*. *Suc* (*least s*) + *i* $\in$ *s* − {*least s*}} (*ldropn* (*Suc* (*least s*)) *w*) =
lselect (*s* − {*least s*}) *w* **using** *lselect-discard-start 3* **by** *this*

have *lselect s w* = *lselect* (*insert* (*least s*) (*s* − {*least s*})) *w* **unfolding** *4* **by** *simp*
also have ... = *lselect* (*s* − {*least s*}) (*ltake* (*enat* (*least s*)) *w*) $ <w ?! least s> $
lselect {*m*. *Suc* (*least s*) + *m* ∈ *s* − {*least s*}} (*ldropn* (*Suc* (*least s*)) *w*)
unfolding *lnths-insert*[*OF 5*] **by** *simp*
also have ... = <w ?! least s> $
lselect {*m*. *Suc* (*least s*) + *m* ∈ *s* − {*least s*}} (*ldropn* (*Suc* (*least s*)) *w*)
unfolding *6* **by** *simp*
also have ... = *w* ?! (*least s*) % *lselect* (*s* − {*least s*}) *w* **unfolding** *7* **by** *simp*
finally show *?thesis* **by** *this*
qed

lemma *lselect-lnth*[*simp*]:
assumes *enat i* < *llength* (*lselect s w*)
shows *lselect s w* ?! *i* = *w* ?! *nth-least s i*
using *assms*
proof (*induct i arbitrary*: *s*)
case *0*
have *1*: ¬ *lnull* (*lselect s w*) **using** *0* **by** *auto*
show *?case* **using** *lselect-least 1* **by** *force*
next
case (*Suc i*)
have *1*: ¬ *lnull* (*lselect s w*) **using** *Suc*(*2*) **by** *auto*
have *2*: *lselect s w* = *w* ?! *least s* % *lselect* (*s* − {*least s*}) *w* **using** *lselect-least 1* **by** *this*
have *3*: *llength* (*lselect s w*) = *eSuc* (*llength* (*lselect* (*s* − {*least s*}) *w*)) **using** *2* **by** *simp*
have *4*: *enat i* < *llength* (*lselect* (*s* − {*least s*}) *w*) **using** *3 Suc*(*2*) **by** *simp*
have *lselect s w* ?! *Suc i* = (*w* ?! *least s* % *lselect* (*s* − {*least s*}) *w*) ?! *Suc i* **using** *2* **by** *simp*
also have ... = *lselect* (*s* − {*least s*}) *w* ?! *i* **by** *simp*
also have ... = *w* ?! *nth-least* (*s* − {*least s*}) *i* **using** *Suc*(*1*) *4* **by** *simp*
also have ... = *w* ?! *nth-least s* (*Suc i*) **by** *simp*
finally show *?case* **by** *this*
qed
lemma *lproject-lnth*[*simp*]:
assumes *enat i* < *llength* (*lproject A w*)
shows *lproject A w* ?! *i* = *w* ?! *nth-least* (*liset A w*) *i*
using *assms* **unfolding** *lproject-to-lselect* **by** *simp*

lemma *lproject-ltake*[*simp*]:
assumes *enat k* ≤ *llength* (*lproject A w*)
shows *lproject A* (*ltake* (*enat* (*nth-least* (*lift* (*liset A w*)) *k*)) *w*) =
ltake (*enat k*) (*lproject A w*)
proof
have *llength* (*lproject A* (*ltake* (*enat* (*nth-least* (*lift* (*liset A w*)) *k*)) *w*)) =
enat (*card* (*liset A w* ∩ {..< *nth-least* (*lift* (*liset A w*)) *k*})) **by** *simp*

also have ... = *enat* (*card* {*i* ∈ *liset A w*. *i* < *nth-least* (*lift* (*liset A w*)) *k*})
unfolding *lessThan-def Collect-conj-eq* **by** *simp*
also have ... = *enat k* **using** *assms* **by** *simp*
also have ... = *llength* (*ltake* (*enat k*) (*lproject A w*)) **using** *min-absorb1 assms* **by** *force*
finally show *llength* (*lproject A* (*ltake* (*enat* (*nth-least* (*lift* (*liset A w*)) *k*)) *w*)) =
llength (*ltake* (*enat k*) (*lproject A w*)) **by** *this*
next
fix *i*
assume *1*: *enat i* < *llength* (*lproject A* (*ltake* (*enat* (*nth-least* (*lift* (*liset A w*)) *k*)) *w*))
assume *2*: *enat i* < *llength* (*ltake* (*enat k*) (*lproject A w*))
obtain *k'* **where** *3*: *k* = *Suc k'* **using** *2 nat.exhaust* **by** *auto*
have *4*: *enat k'* < *llength* (*lproject A w*) **using** *assms 3* **by** *simp*
have *5*: *i* ≤ *k'* **using** *2 3* **by** *simp*
have *6*: *nth-least* (*lift* (*liset A w*)) *k* = *Suc* (*nth-least* (*liset A w*) *k'*)
using *3 4* **by** (*simp del*: *nth-least.simps*)
have *7*: *nth-least* (*liset A w*) *i* < *Suc* (*nth-least* (*liset A w*) *k'*)
proof −
have *nth-least* (*liset A w*) *i* ≤ *nth-least* (*liset A w*) *k'* **using** *4 5* **by** *simp*
also have ... < *Suc* (*nth-least* (*liset A w*) *k'*) **by** *simp*
finally show *?thesis* **by** *this*
qed
have *8*: *nth-least* (*liset A w* ∩ {..< *Suc* (*nth-least* (*liset A w*) *k'*)}) *i* =
nth-least (*liset A w*) *i*
proof (*rule nth-least-eq*)
show *enat i* < *esize* (*liset A w* ∩ {..< *Suc* (*nth-least* (*liset A w*) *k'*)}) **using** *1 6* **by** *simp*
have *enat i* ≤ *enat k'* **using** *5* **by** *simp*
also have *enat k'* < *esize* (*liset A w*) **using** *4* **by** *simp*
finally show *enat i* < *esize* (*liset A w*) **by** *this*
next
fix *j*
assume *1*: *j* ≤ *nth-least* (*liset A w*) *i*
show *j* ∈ *liset A w* ∩ {..< *Suc* (*nth-least* (*liset A w*) *k'*)} ⟷ *j* ∈ *liset A w*
using *1 7* **by** *simp*
qed
have *lproject A* (*ltake* (*enat* (*nth-least* (*lift* (*liset A w*)) *k*)) *w*) *?! i* =
ltake (*enat* (*Suc* (*nth-least* (*liset A w*) *k'*))) *w ?!*
nth-least (*liset A w* ∩ {..< *Suc* (*nth-least* (*liset A w*) *k'*)}) *i*
using *1 6* **by** *simp*
also have ... = *ltake* (*enat* (*Suc* (*nth-least* (*liset A w*) *k'*))) *w ?! nth-least* (*liset A w*) *i*
using *8* **by** *simp*
also have ... = *w ?! nth-least* (*liset A w*) *i* **using** *7* **by** *simp*
also have ... = *lproject A w ?! i* **using** *2* **by** *simp*
also have ... = *ltake* (*enat k*) (*lproject A w*) *?! i* **using** *2* **by** *simp*
finally show *lproject A* (*ltake* (*enat* (*nth-least* (*lift* (*liset A w*)) *k*)) *w*) *?! i* =

ltake (*enat k*) (*lproject A w*) *?! i* **by** *this*
qed

lemma *llength-less-llength-lselect-less*:
enat i < *esize s* ∧ *enat* (*nth-least s i*) < *llength w* ⟷ *enat i* < *llength* (*lselect s w*)
using *nth-least-less-esize-less* **unfolding** *lselect-llength* **by** *this*

lemma *lselect-lselect''*:
assumes ⋀ *i*. *i* ∈ *s* ⟹ *enat i* < *llength w*
assumes ⋀ *i*. *i* ∈ *t* ⟹ *enat i* < *llength* (*lselect s w*)
shows *lselect t* (*lselect s w*) = *lselect* (*nth-least s* ‘ *t*) *w*
proof
note *lselect-llength*[*simp*]
have *1*: ⋀ *i*. *i* ∈ *nth-least s* ‘ *t* ⟹ *enat i* < *llength w* **using** *assms* **by** *auto*
have *2*: *t* ⊆ {*i*. *enat i* < *esize s*}
using *assms*(*2*) *lselect-llength-le less-le-trans* **by** *blast*
have *3*: *inj-on* (*nth-least s*) *t* **using** *subset-inj-on nth-least.inj-on 2* **by** *this*
have *llength* (*lselect t* (*lselect s w*)) = *esize t* **using** *assms*(*2*) **by** *simp*
also have ... = *esize* (*nth-least s* ‘ *t*) **using** *3* **by** *auto*
also have ... = *llength* (*lselect* (*nth-least s* ‘ *t*) *w*) **using** *1* **by** *simp*
finally show *llength* (*lselect t* (*lselect s w*)) = *llength* (*lselect* (*nth-least s* ‘ *t*) *w*)
by *this*
next
fix *i*
assume *1*: *enat i* < *llength* (*lselect t* (*lselect s w*))
assume *2*: *enat i* < *llength* (*lselect* (*nth-least s* ‘ *t*) *w*)
have *3*: *enat i* < *esize t* **using** *less-le-trans 1 lselect-llength-le* **by** *this*
have *4*: ⋀ *i*. *i* ∈ *t* ⟹ *enat i* < *esize s*
using *assms*(*2*) *lselect-llength-le less-le-trans* **by** *blast*
have *lselect t* (*lselect s w*) *?! i* = *lselect s w ?! nth-least t i* **using** *1* **by** *simp*
also have ... = *w ?! nth-least s* (*nth-least t i*) **using** *assms*(*2*) *3* **by** *simp*
also have ... = *w ?! nth-least* (*nth-least s* ‘ *t*) *i* **using** *3 4* **by** *simp*
also have ... = *lselect* (*nth-least s* ‘ *t*) *w ?! i* **using** *2* **by** *simp*
finally show *lselect t* (*lselect s w*) *?! i* = *lselect* (*nth-least s* ‘ *t*) *w ?! i* **by** *this*
qed

lemma *lselect-lselect'*[*simp*]:
assumes ⋀ *i*. *i* ∈ *t* ⟹ *enat i* < *esize s*
shows *lselect t* (*lselect s w*) = *lselect* (*nth-least s* ‘ *t*) *w*
proof −
have *1*: *nth-least* {*i* ∈ *s*. *enat i* < *llength w*} ‘ {*i* ∈ *t*. *enat i* < *llength* (*lselect s w*)} =
{*i* ∈ *nth-least s* ‘ *t*. *enat i* < *llength w*}
unfolding *Compr-image-eq*
proof (*rule image-cong*)
show {*i* ∈ *t*. *enat i* < *llength* (*lselect s w*)} = {*i* ∈ *t*. *enat* (*nth-least s i*) < *llength w*}

using *llength-less-llength-lselect-less assms* **by** *blast*
next
fix i
assume *1*: $i \in \{i \in t.\ enat\ (nth\text{-}least\ s\ i) < llength\ w\}$
have *2*: $enat\ i < esize\ \{i \in s.\ enat\ i < llength\ w\}$
using *nth-least-less-esize-less assms 1* **by** *blast*
show $nth\text{-}least\ \{i \in s.\ enat\ i < llength\ w\}\ i = nth\text{-}least\ s\ i$ **using** *2* **by** *simp*
qed
have $lselect\ t\ (lselect\ s\ w) =$
$lselect\ \{i \in t.\ enat\ i < llength\ (lselect\ s\ w)\}\ (lselect\ \{i \in s.\ enat\ i < llength\ w\}\ w)$
by *simp*
also have $\ldots = lselect\ (nth\text{-}least\ \{i \in s.\ enat\ i < llength\ w\}$ ‘
$\{i \in t.\ enat\ i < llength\ (lselect\ s\ w)\})\ w$
by (*rule lselect-lselect*″, *auto simp*: *lselect-llength*)
also have $\ldots = lselect\ \{i \in nth\text{-}least\ s$ ‘ $t.\ enat\ i < llength\ w\}\ w$ **unfolding** *1* **by** *rule*
also have $\ldots = lselect\ (nth\text{-}least\ s$ ‘ $t)\ w$ **by** *simp*
finally show *?thesis* **by** *this*
qed

lemma *lselect-lselect*:
$lselect\ t\ (lselect\ s\ w) = lselect\ (nth\text{-}least\ s$ ‘ $\{i \in t.\ enat\ i < esize\ s\})\ w$
proof −
have $lselect\ t\ (lselect\ s\ w) = lselect\ \{i \in t.\ enat\ i < llength\ (lselect\ s\ w)\}\ (lselect\ s\ w)$
by *simp*
also have $\ldots = lselect\ (nth\text{-}least\ s$ ‘ $\{i \in t.\ enat\ i < llength\ (lselect\ s\ w)\})\ w$
using *lselect-llength-le less-le-trans* **by** (*blast intro*: *lselect-lselect*′)
also have $\ldots = lselect\ (nth\text{-}least\ s$ ‘ $\{i \in t.\ enat\ i < esize\ s\})\ w$
using *llength-less-llength-lselect-less* **by** (*auto intro*!: *lnths-cong*)
finally show *?thesis* **by** *this*
qed

lemma *lselect-lproject*′:
assumes $\bigwedge i.\ i \in s \Longrightarrow enat\ i < llength\ w$
shows $lproject\ A\ (lselect\ s\ w) = lselect\ (s \cap liset\ A\ w)\ w$
proof −
have *1*: $\bigwedge i.\ i \in liset\ A\ (lselect\ s\ w) \Longrightarrow enat\ i < esize\ s$ **using** *less-le-trans* **by** *force*
have *2*: $\{i \in liset\ A\ (lselect\ s\ w).\ enat\ i < esize\ s\} = liset\ A\ (lselect\ s\ w)$
using *1* **by** *auto*
have *3*: $nth\text{-}least\ s$ ‘ $liset\ A\ (lselect\ s\ w) = s \cap liset\ A\ w$
proof *safe*
fix k
assume *4*: $k \in liset\ A\ (lselect\ s\ w)$
show $nth\text{-}least\ s\ k \in s$ **using** *1 4* **by** *simp*
show $nth\text{-}least\ s\ k \in liset\ A\ w$

using *llength-less-llength-lselect-less 4* **unfolding** *liset-def* **by** *auto*
next
fix *k*
assume *1*: $k \in s$ $k \in$ *liset A w*
have *2*: *nth-least s* (*card* $\{i \in s.\ i < k\}$) $= k$ **using** *nth-least-card 1*(*1*) **by** *this*
have *3*: *enat* (*card* $\{i \in s.\ i < k\}$) $<$ *llength* (*lselect s w*)
unfolding *lselect-llength* **using** *assms 1*(*1*) **by** *simp*
show $k \in$ *nth-least s* ‘ *liset A* (*lselect s w*)
proof
show $k =$ *nth-least s* (*card* $\{i \in s.\ i < k\}$) **using** *2* **by** *simp*
show *card* $\{i \in s.\ i < k\} \in$ *liset A* (*lselect s w*) **using** *1*(*2*) *2 3* **by** *fastforce*
qed
qed
have *lproject A* (*lselect s w*) $=$ *lselect* (*liset A* (*lselect s w*)) (*lselect s w*)
unfolding *lproject-to-lselect* **by** *rule*
also have $\ldots =$ *lselect* (*nth-least s* ‘ $\{i \in$ *liset A* (*lselect s w*). *enat* $i <$ *esize* $s\}$) *w*
unfolding *lselect-lselect* **by** *rule*
also have $\ldots =$ *lselect* (*nth-least s* ‘ *liset A* (*lselect s w*)) *w* **unfolding** *2* **by** *rule*
also have $\ldots =$ *lselect* ($s \cap$ *liset A w*) *w* **unfolding** *3* **by** *rule*
finally show *?thesis* **by** *this*
qed

lemma *lselect-lproject*[*simp*]: *lproject A* (*lselect s w*) $=$ *lselect* ($s \cap$ *liset A w*) *w*
proof −
have *1*: $\{i \in s.$ *enat* $i <$ *llength* $w\} \cap$ *liset A w* $= s \cap$ *liset A w* **by** *auto*
have *lproject A* (*lselect s w*) $=$ *lproject A* (*lselect* $\{i \in s.$ *enat* $i <$ *llength* $w\}$ *w*) **by** *simp*
also have $\ldots =$ *lselect* ($\{i \in s.$ *enat* $i <$ *llength* $w\} \cap$ *liset A w*) *w*
by (*rule lselect-lproject′*, *simp*)
also have $\ldots =$ *lselect* ($s \cap$ *liset A w*) *w* **unfolding** *1* **by** *rule*
finally show *?thesis* **by** *this*
qed

lemma *lproject-lselect-subset*[*simp*]:
assumes *liset A w* $\subseteq s$
shows *lproject A* (*lselect s w*) $=$ *lproject A w*
proof −
have *1*: $s \cap$ *liset A w* $=$ *liset A w* **using** *assms* **by** *auto*
have *lproject A* (*lselect s w*) $=$ *lselect* ($s \cap$ *liset A w*) *w* **by** *simp*
also have $\ldots =$ *lselect* (*liset A w*) *w* **unfolding** *1* **by** *rule*
also have $\ldots =$ *lproject A w* **unfolding** *lproject-to-lselect* **by** *rule*
finally show *?thesis* **by** *this*
qed

lemma *lselect-prefix*[*intro*]:
assumes $u \le v$

shows *lselect s u* $\leq$ *lselect s v*
proof (*cases lfinite u*)
case *False*
show *?thesis* **using** *lprefix-infinite assms False* **by** *auto*
next
case *True*
obtain *k* **where** *1*: *llength u* = *enat k* **using** *True length-list-of* **by** *metis*
obtain *w* **where** *2*: *v* = *u* \$ *w* **using** *lprefix-conv-lappend assms* **by** *auto*
have *lselect s u* $\leq$ *lselect s u* \$ *lselect* {*n. n* + *k* $\in$ *s*} *w* **by** *simp*
also have ... = *lselect s* (*u* \$ *w*) **using** *lnths-lappend-lfinite*[*symmetric*] *1* **by** *this*
also have ... = *lselect s v* **unfolding** *2* **by** *rule*
finally show *?thesis* **by** *this*
qed
lemma *lproject-prefix*[*intro*]:
assumes *u* $\leq$ *v*
shows *lproject A u* $\leq$ *lproject A v*
using *lprefix-lfilterI assms* **by** *auto*

lemma *lproject-prefix-limit*[*intro?*]:
assumes $\bigwedge$ *v. v* $\leq$ *w* $\Longrightarrow$ *lfinite v* $\Longrightarrow$ *lproject A v* $\leq$ *x*
shows *lproject A w* $\leq$ *x*
proof −
have *1*: *ccpo.admissible lSup lprefix* (λ *v. lproject A v* $\leq$ *x*) **by** *simp*
show *?thesis* **using** *llist-lift-admissible 1 assms*(*1*) **by** *this*
qed
lemma *lproject-prefix-limit'*:
assumes $\bigwedge$ *k.* $\exists$ *v. v* $\leq$ *w* $\wedge$ *enat k* < *llength v* $\wedge$ *lproject A v* $\leq$ *x*
shows *lproject A w* $\leq$ *x*
proof (*rule lproject-prefix-limit*)
fix *u*
assume *1*: *u* $\leq$ *w lfinite u*
obtain *k* **where** *2*: *llength u* = *enat k* **using** *1*(*2*) **by** (*metis length-list-of*)
obtain *v* **where** *3*: *v* $\leq$ *w llength u* < *llength v lproject A v* $\leq$ *x*
unfolding *2* **using** *assms*(*1*) **by** *auto*
have *4*: *llength u* $\leq$ *llength v* **using** *3*(*2*) **by** *simp*
have *5*: *u* $\leq$ *v* **using** *prefix-subsume 1*(*1*) *3*(*1*) *4* **by** *this*
have *lproject A u* $\leq$ *lproject A v* **using** *5* **by** *rule*
also have ... $\leq$ *x* **using** *3*(*3*) **by** *this*
finally show *lproject A u* $\leq$ *x* **by** *this*
qed

end

15 Prefixes on Coinductive Lists

theory *LList-Prefixes*
imports
Word-Prefixes

../Extensions/Coinductive-List-Extensions
begin

lemma *unfold-stream-siterate-smap*: *unfold-stream f g* = *smap f* ∘ *siterate g*
by (*rule*, *coinduction*, *auto*) (*metis unfold-stream-eq-SCons*)+

lemma *lappend-stream-of-llist*:
assumes *lfinite u*
shows *stream-of-llist* (*u* \$ *v*) = *list-of u* @− *stream-of-llist v*
using *assms* **unfolding** *stream-of-llist-def* **by** *induct auto*

lemma *llist-of-inf-llist-prefix*[*intro*]: $u \le_{FI} v \Longrightarrow$ *llist-of u* ≤ *llist-of-stream v*
by (*metis lappend-llist-of-stream-conv-shift le-llist-conv-lprefix lprefix-lappend prefix-fininfE*)
lemma *prefix-llist-of-inf-llist*[*intro*]: *lfinite u* ⟹ *u* ≤ *v* ⟹ *list-of u* $\le_{FI}$ *stream-of-llist v*
by (*metis lappend-stream-of-llist le-llist-conv-lprefix lprefix-conv-lappend prefix-fininfI*)

lemma *lproject-prefix-limit-chain*:
assumes *chain w* ⋀ *k*. *lproject A* (*llist-of* (*w k*)) ≤ *x*
shows *lproject A* (*llist-of-stream* (*limit w*)) ≤ *x*
proof (*rule lproject-prefix-limit'*)
fix *k*
obtain *l* **where** *1*: *k* < *length* (*w l*) **using** *assms*(*1*) **by** *rule*
show ∃ *v* ≤ *llist-of-stream* (*limit w*). *enat k* < *llength v* ∧ *lproject A v* ≤ *x*
proof (*intro exI conjI*)
show *llist-of* (*w l*) ≤ *llist-of-stream* (*limit w*)
using *llist-of-inf-llist-prefix chain-prefix-limit assms*(*1*) **by** *this*
show *enat k* < *llength* (*llist-of* (*w l*)) **using** *1* **by** *simp*
show *lproject A* (*llist-of* (*w l*)) ≤ *x* **using** *assms*(*2*) **by** *this*
qed
qed
lemma *lproject-eq-limit-chain*:
assumes *chain u chain v* ⋀ *k*. *project A* (*u k*) = *project A* (*v k*)
shows *lproject A* (*llist-of-stream* (*limit u*)) = *lproject A* (*llist-of-stream* (*limit v*))
proof (*rule antisym*)
show *lproject A* (*llist-of-stream* (*limit u*)) ≤ *lproject A* (*llist-of-stream* (*limit v*))
proof (*rule lproject-prefix-limit-chain*)
show *chain u* **using** *assms*(*1*) **by** *this*
next
fix *k*
have *lproject A* (*llist-of* (*u k*)) = *lproject A* (*llist-of* (*v k*)) **using** *assms*(*3*) **by** *simp*
also have … ≤ *lproject A* (*llist-of-stream* (*limit v*)) **using** *chain-prefix-limit assms*(*2*) **by** *blast*
finally show *lproject A* (*llist-of* (*u k*)) ≤ *lproject A* (*llist-of-stream* (*limit v*))

```
by this
    qed
    show lproject A (llist-of-stream (limit v)) ≤ lproject A (llist-of-stream (limit
u))
    proof (rule lproject-prefix-limit-chain)
      show chain v using assms(2) by this
    next
      fix k
      have lproject A (llist-of (v k)) = lproject A (llist-of (u k)) using assms(3)
by simp
      also have ... ≤ lproject A (llist-of-stream (limit u)) using chain-prefix-limit
assms(1) by blast
      finally show lproject A (llist-of (v k)) ≤ lproject A (llist-of-stream (limit u))
by this
    qed
  qed

end
```

16 Stuttering

```
theory Stuttering
imports
  Stuttering-Equivalence.StutterEquivalence
  LList-Prefixes
begin

  function nth-least-ext :: nat set ⇒ nat ⇒ nat
    where
      enat k < esize A ⟹ nth-least-ext A k = nth-least A k |
      enat k ≥ esize A ⟹ nth-least-ext A k = Suc (Max A + (k − card A))
    by force+ termination by lexicographic-order

  lemma nth-least-ext-strict-mono:
    assumes k < l
    shows nth-least-ext s k < nth-least-ext s l
  proof (cases enat l < esize s)
    case True
    have 1: enat k < esize s using assms True by (metis enat-ord-simps(2)
less-trans)
    show ?thesis using nth-least-strict-mono assms True 1 by simp
  next
    case False
    have 1: finite s using False esize-infinite-enat by auto
    have 2: enat l ≥ esize s using False by simp
    have 3: l ≥ card s using 1 2 by simp
    show ?thesis
    proof (cases enat k < esize s)
      case True
```

have *4*: $s \neq \{\}$ **using** *True* **by** *auto*
have *nth-least-ext s k* = *nth-least s k* **using** *True* **by** *simp*
also have $\ldots \leq$ *Max s* **using** *nth-least-le-Max 1 4 True* **by** *this*
also have $\ldots <$ *Suc* (*Max s*) **by** *auto*
also have $\ldots \leq$ *Suc* (*Max s* + (*l* − *card s*)) **by** *auto*
also have *Suc* (*Max s* + (*l* − *card s*)) = *nth-least-ext s l* **using** *2* **by** *simp*
finally show *?thesis* **by** *this*
next
case *False*
have *4*: *enat k* $\geq$ *esize s* **using** *False* **by** *simp*
have *5*: *k* $\geq$ *card s* **using** *1 4* **by** *simp*
have *nth-least-ext s k* = *Suc* (*Max s* + (*k* − *card s*)) **using** *4* **by** *simp*
also have $\ldots <$ *Suc* (*Max s* + (*l* − *card s*)) **using** *assms 5* **by** *simp*
also have $\ldots =$ *nth-least-ext s l* **using** *2* **by** *simp*
finally show *?thesis* **by** *this*
qed
qed

definition *stutter-selection* :: *nat set* ⇒ *'a llist* ⇒ *bool*
where *stutter-selection s w* ≡ *0* ∈ *s* ∧
(∀ *k i*. *enat i* < *llength w* ⟶ *enat* (*Suc k*) < *esize s* ⟶
nth-least s k < *i* ⟶ *i* < *nth-least s* (*Suc k*) ⟶ *w ?! i* = *w ?! nth-least s k*) ∧
(∀ *i*. *enat i* < *llength w* ⟶ *finite s* ⟶ *Max s* < *i* ⟶ *w ?! i* = *w ?! Max s*)

lemma *stutter-selectionI*[*intro*]:
assumes *0* ∈ *s*
assumes ⋀ *k i*. *enat i* < *llength w* ⟹ *enat* (*Suc k*) < *esize s* ⟹
nth-least s k < *i* ⟹ *i* < *nth-least s* (*Suc k*) ⟹ *w ?! i* = *w ?! nth-least s k*
assumes ⋀ *i*. *enat i* < *llength w* ⟹ *finite s* ⟹ *Max s* < *i* ⟹ *w ?! i* = *w ?! Max s*
shows *stutter-selection s w*
using *assms* **unfolding** *stutter-selection-def* **by** *auto*
lemma *stutter-selectionD-0*[*dest*]:
assumes *stutter-selection s w*
shows *0* ∈ *s*
using *assms* **unfolding** *stutter-selection-def* **by** *auto*
lemma *stutter-selectionD-inside*[*dest*]:
assumes *stutter-selection s w*
assumes *enat i* < *llength w enat* (*Suc k*) < *esize s*
assumes *nth-least s k* < *i i* < *nth-least s* (*Suc k*)
shows *w ?! i* = *w ?! nth-least s k*
using *assms* **unfolding** *stutter-selection-def* **by** *auto*
lemma *stutter-selectionD-infinite*[*dest*]:
assumes *stutter-selection s w*
assumes *enat i* < *llength w finite s Max s* < *i*
shows *w ?! i* = *w ?! Max s*
using *assms* **unfolding** *stutter-selection-def* **by** *auto*

lemma *stutter-selection-stutter-sampler*[*intro*]:

assumes *linfinite w stutter-selection s w*
shows *stutter-sampler* (*nth-least-ext s*) (*lnth w*)
unfolding *stutter-sampler-def*
proof *safe*
show *nth-least-ext s 0 = 0* **using** *assms*(*2*) **by** (*cases enat 0 < esize s*, *auto*)
show *strict-mono* (*nth-least-ext s*) **using** *strict-monoI nth-least-ext-strict-mono* **by** *blast*
next
fix *k i*
assume *1*: *nth-least-ext s k < i i < nth-least-ext s* (*Suc k*)
show *w ?! i = w ?! nth-least-ext s k*
proof (*cases enat* (*Suc k*) *esize s rule*: *linorder-cases*)
case *less*
have *w ?! i = w ?! nth-least s k*
proof (*rule stutter-selectionD-inside*)
show *stutter-selection s w* **using** *assms*(*2*) **by** *this*
show *enat i < llength w* **using** *assms*(*1*) **by** *auto*
show *enat* (*Suc k*) *< esize s* **using** *less* **by** *this*
show *nth-least s k < i* **using** *1*(*1*) *less* **by** *auto*
show *i < nth-least s* (*Suc k*) **using** *1*(*2*) *less* **by** *simp*
qed
also have *w ?! nth-least s k = w ?! nth-least-ext s k* **using** *less* **by** *auto*
finally show *?thesis* **by** *this*
next
case *equal*
have *2*: *enat k < esize s* **using** *equal* **by** (*metis enat-ord-simps*(*2*) *lessI*)
have *3*: *finite s* **using** *equal* **by** (*metis esize-infinite-enat less-irrefl*)
have *4*: ⋀ *i. i > Max s* ⟹ *w ?! i = w ?! Max s* **using** *assms 3* **by** *auto*
have *5*: *k = card s − 1* **using** *equal 3* **by** (*metis diff-Suc-1 enat.inject esize-set.simps*(*1*))
have *Max s = nth-least s* (*card s − 1*) **using** *nth-least-Max 3 assms*(*2*) **by** *force*
also have ... = *nth-least s k* **unfolding** *5* **by** *rule*
also have ... = *nth-least-ext s k* **using** *2* **by** *simp*
finally have *6*: *Max s = nth-least-ext s k* **by** *this*
have *w ?! i = w ?! Max s* **using** *1*(*1*) *4 6* **by** *auto*
also have ... = *w ?! nth-least-ext s k* **unfolding** *6* **by** *rule*
finally show *?thesis* **by** *this*
next
case *greater*
have *2*: *enat k* ≥ *esize s* **using** *greater* **by** (*metis Suc-ile-eq not-le*)
have *3*: *finite s* **using** *greater* **by** (*metis esize-infinite-enat less-asym*)
have *4*: ⋀ *i. i > Max s* ⟹ *w ?! i = w ?! Max s* **using** *assms 3* **by** *auto*
have *w ?! i = w ?! Max s* **using** *1*(*1*) *2 4* **by** *auto*
also have ... = *w ?! Suc* (*Max s +* (*k − card s*)) **using** *4* **by** *simp*
also have ... = *w ?! nth-least-ext s k* **using** *2* **by** *simp*
finally show *?thesis* **by** *this*
qed
qed

lemma *stutter-equivI-selection*[*intro*]:
assumes *linfinite u linfinite v*
assumes *stutter-selection s u stutter-selection t v*
assumes *lselect s u* = *lselect t v*
shows *lnth u* ≈ *lnth v*
proof (*rule stutter-equivI*)
have *1*: *llength* (*lselect s u*) = *llength* (*lselect t v*) **unfolding** *assms*(*5*) **by** *rule*
have *2*: *esize s* = *esize t* **using** *1 assms*(*1*, *2*) **unfolding** *lselect-llength* **by** *simp*
show *stutter-sampler* (*nth-least-ext s*) (*lnth u*) **using** *assms*(*1*, *3*) **by** *rule*
show *stutter-sampler* (*nth-least-ext t*) (*lnth v*) **using** *assms*(*2*, *4*) **by** *rule*
show *lnth u* ∘ *nth-least-ext s* = *lnth v* ∘ *nth-least-ext t*
proof (*rule ext*, *unfold comp-apply*)
fix *i*
show *u* ?! *nth-least-ext s i* = *v* ?! *nth-least-ext t i*
proof (*cases enat i* < *esize s*)
case *True*
have *3*: *enat i* < *llength* (*lselect s u*) *enat i* < *llength* (*lselect t v*)
using *assms*(*1*, *2*) *2 True* **unfolding** *lselect-llength* **by** *auto*
have *u* ?! *nth-least-ext s i* = *u* ?! *nth-least s i* **using** *True* **by** *simp*
also have ... = *lselect s u* ?! *i* **using** *3*(*1*) **by** *simp*
also have ... = *lselect t v* ?! *i* **unfolding** *assms*(*5*) **by** *rule*
also have ... = *v* ?! *nth-least t i* **using** *3*(*2*) **by** *simp*
also have ... = *v* ?! *nth-least-ext t i* **using** *True* **unfolding** *2* **by** *simp*
finally show *u* ?! *nth-least-ext s i* = *v* ?! *nth-least-ext t i* **by** *this*
next
case *False*
have *3*: *s* ≠ {} *t* ≠ {} **using** *assms*(*3*, *4*) **by** *auto*
have *4*: *finite s finite t* **using** *esize-infinite-enat 2 False* **by** *metis+*
have *5*: ⋀ *i*. *i* > *Max s* ⟹ *u* ?! *i* = *u* ?! *Max s* **using** *assms*(*1*, *3*) *4*(*1*) **by** *auto*
have *6*: ⋀ *i*. *i* > *Max t* ⟹ *v* ?! *i* = *v* ?! *Max t* **using** *assms*(*2*, *4*) *4*(*2*) **by** *auto*
have *7*: *esize s* = *enat* (*card s*) *esize t* = *enat* (*card t*) **using** *4* **by** *auto*
have *8*: *card s* ≠ *0 card t* ≠ *0* **using** *3 4* **by** *auto*
have *9*: *enat* (*card s* − *1*) < *llength* (*lselect s u*)
using *assms*(*1*) *7*(*1*) *8*(*1*) **unfolding** *lselect-llength* **by** *simp*
have *10*: *enat* (*card t* − *1*) < *llength* (*lselect t v*)
using *assms*(*2*) *7*(*2*) *8*(*2*) **unfolding** *lselect-llength* **by** *simp*
have *u* ?! *nth-least-ext s i* = *u* ?! *Suc* (*Max s* + (*i* − *card s*)) **using** *False* **by** *simp*
also have ... = *u* ?! *Max s* **using** *5* **by** *simp*
also have ... = *u* ?! *nth-least s* (*card s* − *1*) **using** *nth-least-Max 4*(*1*) *3*(*1*) **by** *force*
also have ... = *lselect s u* ?! (*card s* − *1*) **using** *lselect-lnth 9* **by** *simp*
also have ... = *lselect s u* ?! (*card t* − *1*) **using** *2 4* **by** *simp*
also have ... = *lselect t v* ?! (*card t* − *1*) **unfolding** *assms*(*5*) **by** *rule*
also have ... = *v* ?! *nth-least t* (*card t* − *1*) **using** *lselect-lnth 10* **by** *simp*

also have $\ldots = v$?! *Max t* **using** *nth-least-Max 4*(*2*) *3*(*2*) **by** *force*
also have $\ldots = v$?! *Suc* (*Max t* + (*i* − *card t*)) **using** *6* **by** *simp*
also have $\ldots = v$?! *nth-least-ext t i* **using** *2 False* **by** *simp*
finally show *?thesis* **by** *this*
qed
qed
qed

definition *stuttering-invariant* :: *'a word set* $\Rightarrow$ *bool*
where *stuttering-invariant A* $\equiv \forall\ u\ v.\ u \approx v \longrightarrow u \in A \longleftrightarrow v \in A$

lemma *stuttering-invariant-complement*[*intro*!]:
assumes *stuttering-invariant A*
shows *stuttering-invariant* (− *A*)
using *assms* **unfolding** *stuttering-invariant-def* **by** *simp*

lemma *stutter-equiv-forw-subst*[*trans*]: $w_1 = w_2 \Longrightarrow w_2 \approx w_3 \Longrightarrow w_1 \approx w_3$ **by** *auto*

lemma *stutter-sampler-build*:
assumes *stutter-sampler f w*
shows *stutter-sampler* (*0* ## (*Suc* ∘ *f*)) (*a* ## *w*)
unfolding *stutter-sampler-def*
proof *safe*
have *0*: *f 0* = *0* **using** *assms* **unfolding** *stutter-sampler-def* **by** *auto*
have *1*: *f x* < *f y* **if** *x* < *y* **for** *x y*
using *assms that* **unfolding** *stutter-sampler-def strict-mono-def* **by** *auto*
have *2*: (*0* ## (*Suc* ∘ *f*)) *x* < (*0* ## (*Suc* ∘ *f*)) *y* **if** *x* < *y* **for** *x y*
using *1 that* **by** (*cases x*; *cases y*) (*auto*)
have *3*: *w n* = *w* (*f k*) **if** *f k* < *n n* < *f* (*Suc k*) **for** *k n*
using *assms that* **unfolding** *stutter-sampler-def* **by** *auto*
show (*0* ## (*Suc* ∘ *f*)) *0* = *0* **by** *simp*
show *strict-mono* (*0* ## (*Suc* ∘ *f*)) **using** *2* **by** *rule*
show (*a* ## *w*) *n* = (*a* ## *w*) ((*0* ## (*Suc* ∘ *f*)) *k*)
if (*0* ## (*Suc* ∘ *f*)) *k* < *n n* < (*0* ## (*Suc* ∘ *f*)) (*Suc k*) **for** *k n*
using *0 3 that* **by** (*cases k*; *cases n*) (*force*)+
qed
lemma *stutter-extend-build*:
assumes $u \approx v$
shows *a* ## $u \approx a$ ## *v*
proof −
obtain *f g* **where** *1*: *stutter-sampler f u stutter-sampler g v u* ∘ *f* = *v* ∘ *g*
using *stutter-equivE assms* **by** *this*
show *?thesis*
proof (*intro stutter-equivI ext*)
show *stutter-sampler* (*0* ## (*Suc* ∘ *f*)) (*a* ## *u*) **using** *stutter-sampler-build* *1*(*1*) **by** *this*
show *stutter-sampler* (*0* ## (*Suc* ∘ *g*)) (*a* ## *v*) **using** *stutter-sampler-build* *1*(*2*) **by** *this*

show (a ## $u \circ 0$ ## ($Suc \circ f$)) i = (a ## $v \circ 0$ ## ($Suc \circ g$)) i **for** i
using *fun-cong*[*OF 1*(*3*)] **by** (*cases i*) (*auto*)
qed
qed
lemma *stutter-extend-concat*:
assumes $u \approx v$
shows $w \frown u \approx w \frown v$
using *stutter-extend-build assms* **by** (*induct w*, *force+*)
lemma *build-stutter*: $w\ 0$ ## $w \approx w$
proof (*rule stutter-equivI*)
show *stutter-sampler* (*Suc* (*0* := *0*)) ($w\ 0$ ## w)
unfolding *stutter-sampler-def*
proof *safe*
show (*Suc* (*0* := *0*)) *0* = *0* **by** *simp*
show *strict-mono* (*Suc* (*0* := *0*)) **by** (*rule strict-monoI*, *simp*)
next
fix *k n*
assume *1*: (*Suc* (*0* := *0*)) $k < n$ $n <$ (*Suc* (*0* := *0*)) (*Suc k*)
show ($w\ 0$ ## w) n = ($w\ 0$ ## w) ((*Suc* (*0* := *0*)) k) **using** *1* **by** (*cases n*, *auto*)
qed
show *stutter-sampler id w* **by** *rule*
show $w\ 0$ ## $w \circ$ (*Suc* (*0* := *0*)) = $w \circ id$ **by** *auto*
qed
lemma *replicate-stutter*: *replicate n* ($v\ 0$) $\frown v \approx v$
proof (*induct n*)
case *0*
show *?case* **using** *stutter-equiv-refl* **by** *simp*
next
case (*Suc n*)
have *replicate* (*Suc n*) ($v\ 0$) $\frown v$ = $v\ 0$ ## *replicate n* ($v\ 0$) $\frown v$ **by** *simp*
also have ... = (*replicate n* ($v\ 0$) $\frown v$) *0* ## *replicate n* ($v\ 0$) $\frown v$ **by** (*cases n*, *auto*)
also have ... $\approx$ *replicate n* ($v\ 0$) $\frown v$ **using** *build-stutter* **by** *this*
also have ... $\approx v$ **using** *Suc* **by** *this*
finally show *?case* **by** *this*
qed

lemma *replicate-stutter'*: $u \frown$ *replicate n* ($v\ 0$) $\frown v \approx u \frown v$
using *stutter-extend-concat replicate-stutter* **by** *this*

end

17 Interpreted Transition Systems and Traces

theory *Transition-System-Interpreted-Traces*
imports
Transition-System-Traces
Basics/Stuttering

begin

locale *transition-system-interpreted-traces* =
transition-system-interpreted ex en int +
transition-system-traces ex en ind
for *ex* :: *'action* ⇒ *'state* ⇒ *'state*
and *en* :: *'action* ⇒ *'state* ⇒ *bool*
and *int* :: *'state* ⇒ *'interpretation*
and *ind* :: *'action* ⇒ *'action* ⇒ *bool*
+
assumes *independence-invisible*: $a \in visible \Longrightarrow b \in visible \Longrightarrow \neg$ *ind a b*
begin

lemma *eq-swap-lproject-visible*:
assumes $u =_S v$
shows *lproject visible* (*llist-of u*) = *lproject visible* (*llist-of v*)
using *assms independence-invisible* **by** (*induct*, *auto*)
lemma *eq-fin-lproject-visible*:
assumes $u =_F v$
shows *lproject visible* (*llist-of u*) = *lproject visible* (*llist-of v*)
using *assms eq-swap-lproject-visible* **by** (*induct*, *auto*)
lemma *le-fin-lproject-visible*:
assumes $u \preceq_F v$
shows *lproject visible* (*llist-of u*) ≤ *lproject visible* (*llist-of v*)
proof −
obtain *w* **where** *1*: u @ $w =_F v$ **using** *assms* **by** *rule*
have *lproject visible* (*llist-of u*) ≤
lproject visible (*llist-of u*) $ *lproject visible* (*llist-of w*) **by** *auto*
also have . . . = *lproject visible* (*llist-of* (*u* @ *w*)) **using** *lappend-llist-of-llist-of*
by *auto*
also have . . . = *lproject visible* (*llist-of v*) **using** *eq-fin-lproject-visible 1* **by**
this
finally show *?thesis* **by** *this*
qed
lemma *le-fininf-lproject-visible*:
assumes $u \preceq_{FI} v$
shows *lproject visible* (*llist-of u*) ≤ *lproject visible* (*llist-of-stream v*)
proof −
obtain *w* **where** *1*: $w \leq_{FI} v$ $u \preceq_F w$ **using** *assms* **by** *rule*
have *lproject visible* (*llist-of u*) ≤ *lproject visible* (*llist-of w*)
using *le-fin-lproject-visible 1*(*2*) **by** *this*
also have . . . ≤ *lproject visible* (*llist-of-stream v*) **using** *1*(*1*) **by** *blast*
finally show *?thesis* **by** *this*
qed
lemma *le-inf-lproject-visible*:
assumes $u \preceq_I v$
shows *lproject visible* (*llist-of-stream u*) ≤ *lproject visible* (*llist-of-stream v*)
proof (*rule lproject-prefix-limit*)
fix *w*

assume *1*: $w \le$ *llist-of-stream u lfinite w*
have *2*: *list-of w* $\le_{FI}$ *stream-of-llist* (*llist-of-stream u*) **using** *1* **by** *blast*
have *3*: *list-of w* $\preceq_{FI}$ *v* **using** *assms 2* **by** *auto*
have *lproject visible w* = *lproject visible* (*llist-of* (*list-of w*)) **using** *1*(*2*) **by** *simp*
also have ... $\le$ *lproject visible* (*llist-of-stream v*) **using** *le-fininf-lproject-visible 3* **by** *this*
finally show *lproject visible w* $\le$ *lproject visible* (*llist-of-stream v*) **by** *this*
qed
lemma *eq-inf-lproject-visible*:
assumes $u =_I v$
shows *lproject visible* (*llist-of-stream u*) = *lproject visible* (*llist-of-stream v*)
using *le-inf-lproject-visible assms* **by** (*metis antisym eq-infE*)

lemma *stutter-selection-lproject-visible*:
assumes *run u p*
shows *stutter-selection* (*lift* (*liset visible* (*llist-of-stream u*)))
(*llist-of-stream* (*smap int* (*p ## trace u p*)))
proof
show *0* $\in$ *lift* (*liset visible* (*llist-of-stream u*)) **by** *auto*
next
fix *k i*
assume *3*: *enat* (*Suc k*) < *esize* (*lift* (*liset visible* (*llist-of-stream u*)))
assume *4*: *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) *k* < *i*
assume *5*: *i* < *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) (*Suc k*)
have *6*: *int* ((*p ## trace u p*) !! *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) *k*) =
int ((*p ## trace u p*) !! *i*)
proof (*rule execute-inf-word-invisible*)
show *run u p* **using** *assms* **by** *this*
show *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) *k* $\le$ *i* **using** *4* **by** *auto*
next
fix *j*
assume *6*: *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) *k* $\le$ *j*
assume *7*: *j* < *i*
have *8*: *Suc j* $\notin$ *lift* (*liset visible* (*llist-of-stream u*))
proof (*rule nth-least-not-contains*)
show *enat* (*Suc k*) < *esize* (*lift* (*liset visible* (*llist-of-stream u*))) **using** *3* **by** *this*
show *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) *k* < *Suc j* **using** *6* **by** *auto*
show *Suc j* < *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) (*Suc k*) **using** *5 7* **by** *simp*
qed
have *9*: *j* $\notin$ *liset visible* (*llist-of-stream u*) **using** *8* **by** *auto*
show *u* !! *j* $\notin$ *visible* **using** *9* **by** *auto*
qed
show *llist-of-stream* (*smap int* (*p ## trace u p*)) ?! *i* = *llist-of-stream* (*smap int* (*p ## trace u p*)) ?!

nth-least (*lift* (*liset visible* (*llist-of-stream u*))) *k*
using *6* **by** (*metis lnth-list-of-stream snth-smap*)
next
fix *i*
assume *1*: *finite* (*lift* (*liset visible* (*llist-of-stream u*)))
assume *3*: *Max* (*lift* (*liset visible* (*llist-of-stream u*))) $<$ *i*
have *4*: *int* ((*p* ## *trace u p*) !! *Max* (*lift* (*liset visible* (*llist-of-stream u*)))) =
int ((*p* ## *trace u p*) !! *i*)
proof (*rule execute-inf-word-invisible*)
show *run u p* **using** *assms* **by** *this*
show *Max* (*lift* (*liset visible* (*llist-of-stream u*))) $\leq$ *i* **using** *3* **by** *auto*
next
fix *j*
assume *6*: *Max* (*lift* (*liset visible* (*llist-of-stream u*))) $\leq$ *j*
assume *7*: *j* $<$ *i*
have *8*: *Suc j* $\notin$ *lift* (*liset visible* (*llist-of-stream u*))
proof (*rule ccontr*)
assume *9*: $\neg$ *Suc j* $\notin$ *lift* (*liset visible* (*llist-of-stream u*))
have *10*: *Suc j* $\in$ *lift* (*liset visible* (*llist-of-stream u*)) **using** *9* **by** *simp*
have *11*: *Suc j* $\leq$ *Max* (*lift* (*liset visible* (*llist-of-stream u*))) **using** *Max-ge 1 10* **by** *this*
show *False* **using** *6 11* **by** *auto*
qed
have *9*: *j* $\notin$ *liset visible* (*llist-of-stream u*) **using** *8* **by** *auto*
show *u* !! *j* $\notin$ *visible* **using** *9* **by** *auto*
qed
show *llist-of-stream* (*smap int* (*p* ## *trace u p*)) ?! *i* = *llist-of-stream* (*smap int* (*p* ## *trace u p*)) ?!
Max (*lift* (*liset visible* (*llist-of-stream u*))) **using** *4* **by** (*metis lnth-list-of-stream snth-smap*)
qed

lemma *execute-fin-visible*:
assumes *path u q path v q u* $\preceq_{FI}$ *w v* $\preceq_{FI}$ *w*
assumes *project visible u* = *project visible v*
shows *int* (*target u q*) = *int* (*target v q*)
proof –
obtain *w'* **where** *1*: *u* $\preceq_F$ *w' v* $\preceq_F$ *w'* **using** *subsume-fin assms*(*3*, *4*) **by** *this*
obtain *u' v'* **where** *2*: *u* @ *u'* $=_F$ *w' v* @ *v'* $=_F$ *w'* **using** *1* **by** *blast*
have *u* @ *u'* $=_F$ *w'* **using** *2*(*1*) **by** *this*
also have . . . $=_F$ *v* @ *v'* **using** *2*(*2*) **by** *blast*
finally have *3*: *u* @ *u'* $=_F$ *v* @ *v'* **by** *this*
obtain s_1 s_2 s_3 **where** *4*: *u* $=_F$ s_1 @ s_2 *v* $=_F$ s_1 @ s_3 *Ind* (*set* s_2) (*set* s_3)
using *levi-lemma 3* **by** *this*
have *5*: *project visible* (s_1 @ s_2) = *project visible* (s_1 @ s_3)
using *eq-fin-lproject-visible assms*(*5*) *4*(*1*, *2*) **by** *auto*
have *6*: *project visible* s_2 = *project visible* s_3 **using** *5* **by** *simp*

have *7*: *set* (*project visible* s_2) = *set* (*project visible* s_3) **using** *6* **by** *simp*
have *8*: *set* s_2 ∩ *visible* = *set* s_3 ∩ *visible* **using** *7* **by** *auto*
have *9*: *set* s_2 ⊆ *invisible* ∨ *set* s_3 ⊆ *invisible* **using** *independence-invisible* *4*(*3*) **by** *auto*
have *10*: *set* s_2 ⊆ *invisible set* s_3 ⊆ *invisible* **using** *8 9* **by** *auto*
have *11*: *path* s_2 (*target* s_1 *q*) **using** *eq-fin-word 4*(*1*) *assms*(*1*) **by** *auto*
have *12*: *path* s_3 (*target* s_1 *q*) **using** *eq-fin-word 4*(*2*) *assms*(*2*) **by** *auto*
have *int* (*fold ex u q*) = *int* (*fold ex* (s_1 @ s_2) *q*) **using** *eq-fin-execute assms*(*1*) *4*(*1*) **by** *simp*
also have . . . = *int* (*fold ex* s_1 *q*) **using** *execute-fin-word-invisible 11 10*(*1*) **by** *simp*
also have . . . = *int* (*fold ex* (s_1 @ s_3) *q*) **using** *execute-fin-word-invisible 12* *10*(*2*) **by** *simp*
also have . . . = *int* (*fold ex v q*) **using** *eq-fin-execute assms*(*2*) *4*(*2*) **by** *simp*
finally show *?thesis* **by** *this*
qed
lemma *execute-inf-visible*:
assumes *run u q run v q u* $\preceq_I$ *w v* $\preceq_I$ *w*
assumes *lproject visible* (*llist-of-stream u*) = *lproject visible* (*llist-of-stream v*)
shows *snth* (*smap int* (*q ## trace u q*)) ≈ *snth* (*smap int* (*q ## trace v q*))
proof −
have *1*: *lnth* (*llist-of-stream* (*smap int* (*q ## trace u q*))) ≈
lnth (*llist-of-stream* (*smap int* (*q ## trace v q*)))
proof
show *linfinite* (*llist-of-stream* (*smap int* (*q ## trace u q*))) **by** *simp*
show *linfinite* (*llist-of-stream* (*smap int* (*q ## trace v q*))) **by** *simp*
show *stutter-selection* (*lift* (*liset visible* (*llist-of-stream u*))) (*llist-of-stream* (*smap int* (*q ## trace u q*)))
using *stutter-selection-lproject-visible assms*(*1*) **by** *this*
show *stutter-selection* (*lift* (*liset visible* (*llist-of-stream v*))) (*llist-of-stream* (*smap int* (*q ## trace v q*)))
using *stutter-selection-lproject-visible assms*(*2*) **by** *this*
show *lselect* (*lift* (*liset visible* (*llist-of-stream u*))) (*llist-of-stream* (*smap int* (*q ## trace u q*))) =
lselect (*lift* (*liset visible* (*llist-of-stream v*))) (*llist-of-stream* (*smap int* (*q ## trace v q*)))
proof
have *llength* (*lselect* (*lift* (*liset visible* (*llist-of-stream u*)))
(*llist-of-stream* (*smap int* (*q ## trace u q*)))) = *eSuc* (*llength* (*lproject visible* (*llist-of-stream u*)))
by (*simp add*: *lselect-llength*)
also have . . . = *eSuc* (*llength* (*lproject visible* (*llist-of-stream v*)))
unfolding *assms*(*5*) **by** *rule*
also have . . . = *llength* (*lselect* (*lift* (*liset visible* (*llist-of-stream v*)))
(*llist-of-stream* (*smap int* (*q ## trace v q*)))) **by** (*simp add*: *lselect-llength*)
finally show *llength* (*lselect* (*lift* (*liset visible* (*llist-of-stream u*)))
(*llist-of-stream* (*smap int* (*q ## trace u q*)))) = *llength* (*lselect* (*lift* (*liset visible* (*llist-of-stream v*)))
(*llist-of-stream* (*smap int* (*q ## trace v q*)))) **by** *this*

next
fix *i*
assume *1*:
enat i $<$ *llength* (*lselect* (*lift* (*liset visible* (*llist-of-stream u*)))
(*llist-of-stream* (*smap int* (*q* ## *trace u q*))))
enat i $<$ *llength* (*lselect* (*lift* (*liset visible* (*llist-of-stream v*)))
(*llist-of-stream* (*smap int* (*q* ## *trace v q*))))
have *2*:
enat i $\leq$ *llength* (*lproject visible* (*llist-of-stream u*))
enat i $\leq$ *llength* (*lproject visible* (*llist-of-stream v*))
using *1* **by** (*simp add*: *lselect-llength*)+
define *k* **where** *k* $\equiv$ *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) *i*
define *l* **where** *l* $\equiv$ *nth-least* (*lift* (*liset visible* (*llist-of-stream v*))) *i*
have *lselect* (*lift* (*liset visible* (*llist-of-stream u*))) (*llist-of-stream* (*smap int* (*q* ## *trace u q*))) ?! *i* =
int ((*q* ## *trace u q*) !! *nth-least* (*lift* (*liset visible* (*llist-of-stream u*))) *i*)
by (*metis 1*(*1*) *lnth-list-of-stream lselect-lnth snth-smap*)
also have ... = *int* ((*q* ## *trace u q*) !! *k*) **unfolding** *k-def* **by** *rule*
also have ... = *int* ((*q* ## *trace v q*) !! *l*)
unfolding *sscan-scons-snth*
proof (*rule execute-fin-visible*)
show *path* (*stake k u*) *q* **using** *assms*(*1*) **by** (*metis run-shift-elim stake-sdrop*)
show *path* (*stake l v*) *q* **using** *assms*(*2*) **by** (*metis run-shift-elim stake-sdrop*)
show *stake k u* $\preceq_{FI}$ *w stake l v* $\preceq_{FI}$ *w* **using** *assms*(*3*, *4*) **by** *auto*
have *project visible* (*stake k u*) =
list-of (*lproject visible* (*llist-of* (*stake k u*))) **by** *simp*
also have ... = *list-of* (*lproject visible* (*ltake* (*enat k*) (*llist-of-stream u*)))
by (*metis length-stake llength-llist-of llist-of-inf-llist-prefix lprefix-ltake prefix-fininf-prefix*)
also have ... = *list-of* (*ltake* (*enat i*) (*lproject visible* (*llist-of-stream u*)))
unfolding *k-def lproject-ltake*[*OF 2*(*1*)] **by** *rule*
also have ... = *list-of* (*ltake* (*enat i*) (*lproject visible* (*llist-of-stream v*)))
unfolding *assms*(*5*) **by** *rule*
also have ... = *list-of* (*lproject visible* (*ltake* (*enat l*) (*llist-of-stream v*)))
unfolding *l-def lproject-ltake*[*OF 2*(*2*)] **by** *rule*
also have ... = *project visible* (*stake l v*)
by (*metis length-stake lfilter-llist-of list-of-llist-of llength-llist-of llist-of-inf-llist-prefix lprefix-ltake prefix-fininf-prefix*)
finally show *project visible* (*stake k u*) = *project visible* (*stake l v*) **by** *this*
qed
also have ... = *int* ((*q* ## *trace v q*) !! *nth-least* (*lift* (*liset visible* (*llist-of-stream v*))) *i*)
unfolding *l-def* **by** *simp*
also have ... = *lselect* (*lift* (*liset visible* (*llist-of-stream v*)))
(*llist-of-stream* (*smap int* (*q* ## *trace v q*))) ?! *i*

```
      using 1 by (metis lnth-list-of-stream lselect-lnth snth-smap)
    finally show lselect (lift (liset visible (llist-of-stream u)))
        (llist-of-stream (smap int (q ## trace u q))) ?! i = lselect (lift (liset
visible (llist-of-stream v)))
      (llist-of-stream (smap int (q ## trace v q))) ?! i by this
   qed
  qed
  show ?thesis using 1 by simp
 qed

end

end
```

18 Abstract Theory of Ample Set Partial Order Reduction

```
theory Ample-Abstract
imports
  Transition-System-Interpreted-Traces
  Extensions/Relation-Extensions
begin

 locale ample-base =
  transition-system-interpreted-traces ex en int ind +
  wellfounded-relation src
  for ex :: 'action ⇒ 'state ⇒ 'state
  and en :: 'action ⇒ 'state ⇒ bool
  and int :: 'state ⇒ 'interpretation
  and ind :: 'action ⇒ 'action ⇒ bool
  and src :: 'state ⇒ 'state ⇒ bool
 begin

  definition ample-set :: 'state ⇒ 'action set ⇒ bool
   where ample-set q A ≡
    A ⊆ {a. en a q} ∧
    (A ⊂ {a. en a q} ⟶ A ≠ {}) ∧
    (∀ a. A ⊂ {a. en a q} ⟶ a ∈ A ⟶ src (ex a q) q) ∧
    (A ⊂ {a. en a q} ⟶ A ⊆ invisible) ∧
    (∀ w. A ⊂ {a. en a q} ⟶ path w q ⟶ A ∩ set w = {} ⟶ Ind A (set w))


  lemma ample-subset:
   assumes ample-set q A
   shows A ⊆ {a. en a q}
   using assms unfolding ample-set-def by auto
```

lemma *ample-nonempty*:
assumes *ample-set* q A $A \subset \{a.\ en\ a\ q\}$
shows $A \neq \{\}$
using *assms* **unfolding** *ample-set-def* **by** *auto*

lemma *ample-wellfounded*:
assumes *ample-set* q A $A \subset \{a.\ en\ a\ q\}$ $a \in A$
shows *src* $(ex\ a\ q)$ q
using *assms* **unfolding** *ample-set-def* **by** *auto*

lemma *ample-invisible*:
assumes *ample-set* q A $A \subset \{a.\ en\ a\ q\}$
shows $A \subseteq$ *invisible*
using *assms* **unfolding** *ample-set-def* **by** *auto*

lemma *ample-independent*:
assumes *ample-set* q A $A \subset \{a.\ en\ a\ q\}$ *path* w q $A \cap$ *set* $w = \{\}$
shows *Ind* A (*set* w)
using *assms* **unfolding** *ample-set-def* **by** *auto*

lemma *ample-en*[*intro*]: *ample-set* q $\{a.\ en\ a\ q\}$ **unfolding** *ample-set-def* **by** *blast*

end

locale *ample-abstract* =
S?: *transition-system-complete ex en init int* +
R: *transition-system-complete ex ren init int* +
ample-base ex en int ind src
for *ex* :: $'action \Rightarrow 'state \Rightarrow 'state$
and *en* :: $'action \Rightarrow 'state \Rightarrow bool$
and *init* :: $'state \Rightarrow bool$
and *int* :: $'state \Rightarrow 'interpretation$
and *ind* :: $'action \Rightarrow 'action \Rightarrow bool$
and *src* :: $'state \Rightarrow 'state \Rightarrow bool$
and *ren* :: $'action \Rightarrow 'state \Rightarrow bool$
+
assumes *reduction-ample*: $q \in nodes \Longrightarrow$ *ample-set* q $\{a.\ ren\ a\ q\}$
begin

lemma *reduction-words-fin*:
assumes $q \in$ *nodes* *R.path* w q
shows *S.path* w q
using *assms*(*2*, *1*) *ample-subset reduction-ample* **by** *induct auto*
lemma *reduction-words-inf*:
assumes $q \in$ *nodes* *R.run* w q
shows *S.run* w q
using *reduction-words-fin assms* **by** (*auto intro*: *words-infI-construct*)

```
lemma reduction-step:
  assumes q ∈ nodes run w q
  obtains
    (deferred) a where ren a q [a] ≼_FI w |
    (omitted) {a. ren a q} ⊆ invisible Ind {a. ren a q} (sset w)
proof (cases {a. en a q} = {a. ren a q})
  case True
  have 1: run (shd w ## sdrop 1 w) q using assms(2) by simp
  show ?thesis
  proof (rule deferred)
    show ren (shd w) q using True 1 by blast
    show [shd w] ≼_FI w by simp
  qed
next
  case False
  have 1: {a. ren a q} ⊂ {a. en a q} using ample-subset reduction-ample assms(1) False by auto
  show ?thesis
  proof (cases {a. ren a q} ∩ sset w = {})
    case True
    show ?thesis
    proof (rule omitted)
      show {a. ren a q} ⊆ invisible using ample-invisible reduction-ample assms(1) 1 by auto
      show Ind {a. ren a q} (sset w)
      proof safe
        fix a b
        assume 2: b ∈ sset w ren a q
        obtain u v where 3: w = u @- b ## v using split-stream-first' 2(1) by this
        have 4: Ind {a. ren a q} (set (u @ [b]))
        proof (rule ample-independent)
          show ample-set q {a. ren a q} using reduction-ample assms(1) by this
          show {a. ren a q} ⊂ {a. en a q} using 1 by this
          show path (u @ [b]) q using assms(2) 3 by blast
          show {a. ren a q} ∩ set (u @ [b]) = {} using True 3 by auto
        qed
        show ind a b using 2 3 4 by auto
      qed
    qed
  next
    case False
    obtain u a v where 2: w = u @- a ## v {a. ren a q} ∩ set u = {} ren a q
      using split-stream-first[OF False] by auto
    have 3: path u q using assms(2) unfolding 2(1) by auto
    have 4: Ind {a. ren a q} (set u)
```

using *ample-independent reduction-ample assms*(*1*) *1 3 2*(*2*) **by** *this*
have *5*: *Ind* (*set* [*a*]) (*set u*) **using** *4 2*(*3*) **by** *simp*
have *6*: $[a]$ @ $u =_F u$ @ $[a]$ **using** *5* **by** *blast*
show *?thesis*
proof (*rule deferred*)
show *ren a q* **using** *2*(*3*) **by** *this*
have $[a] \preceq_{FI} [a]$ @− u @− v **by** *blast*
also have $[a]$ @− u @− v = ($[a]$ @ u) @− v **by** *simp*
also have ($[a]$ @ u) @− $v =_I$ (u @ $[a]$) @− v **using** *6* **by** *blast*
also have (u @ $[a]$) @− v = u @− $[a]$ @− v **by** *simp*
also have … = w **unfolding** *2*(*1*) **by** *simp*
finally show $[a] \preceq_{FI} w$ **by** *this*
qed
qed
qed

lemma *reduction-chunk*:
assumes $q \in$ *nodes run* ($[a]$ @− v) q
obtains $b\ b_1\ b_2\ u$
where
R.path (b @ $[a]$) q
Ind $\{a\}$ (*set b*) *set* $b \subseteq$ *invisible*
$b =_F b_1$ @ b_2 b_1 @− $u =_I v$ *Ind* (*set* b_2) (*sset u*)
using *wellfounded assms*
proof (*induct q arbitrary*: *thesis v rule*: *wfP-induct-rule*)
case (*less q*)
show *?case*
proof (*cases ren a q*)
case (*True*)
show *?thesis*
proof (*rule less*(*2*))
show *R.path* ([] @ $[a]$) q **using** *True* **by** *auto*
show *Ind* $\{a\}$ (*set* []) **by** *auto*
show *set* [] $\subseteq$ *invisible* **by** *auto*
show $[] =_F []$ @ [] **by** *auto*
show [] @− $v =_I v$ **by** *auto*
show *Ind* (*set* []) (*sset v*) **by** *auto*
qed
next
case (*False*)
have *0*: $\{a.\ en\ a\ q\} \neq \{a.\ ren\ a\ q\}$ **using** *False less*(*4*) **by** *auto*
show *?thesis*
using *less*(*3*, *4*)
proof (*cases rule*: *reduction-step*)
case (*deferred c*)
have *1*: *ren c q* **using** *deferred*(*1*) **by** *simp*
have *2*: *ind a c*
proof (*rule le-fininf-ind*″)

show $[a] \preceq_{FI} [a]$ @− v **by** *blast*
show $[c] \preceq_{FI} [a]$ @− v **using** *deferred*(*2*) **by** *this*
show $a \neq c$ **using** *False 1* **by** *auto*
qed
obtain v' **where** *3*: $[a]$ @− $v =_I [c]$ @− $[a]$ @− v'
proof −
have *10*: $[c] \preceq_{FI} v$
proof (*rule le-fininf-not-member'*)
show $[c] \preceq_{FI} [a]$ @− v **using** *deferred*(*2*) **by** *this*
show $c \notin$ *set* $[a]$ **using** *False 1* **by** *auto*
qed
obtain v' **where** *11*: $v =_I [c]$ @− v' **using** *10* **by** *blast*
have *12*: *Ind* (*set* $[a]$) (*set* $[c]$) **using** *2* **by** *auto*
have *13*: $[a]$ @ $[c] =_F [c]$ @ $[a]$ **using** *12* **by** *blast*
have $[a]$ @− $v =_I [a]$ @− $[c]$ @− v' **using** *11* **by** *blast*
also have ... = ($[a]$ @ $[c]$) @− v' **by** *simp*
also have ... $=_I$ ($[c]$ @ $[a]$) @− v' **using** *13* **by** *blast*
also have ... = $[c]$ @− $[a]$ @− v' **by** *simp*
finally show *?thesis* **using** *that* **by** *auto*
qed
have *4*: *run* ($[c]$ @− $[a]$ @− v') q **using** *eq-inf-word 3 less*(*4*) **by** *this*
show *?thesis*
proof (*rule less*(*1*))
show *src* (*ex c q*) q
using *ample-wellfounded ample-subset reduction-ample less*(*3*) *0 1* **by**
blast
have *100*: *en c q* **using** *less*(*4*) *deferred*(*2*) *le-fininf-word* **by** *auto*
show *ex c q* $\in$ *nodes* **using** *less*(*3*) *100* **by** *auto*
show *run* ($[a]$ @− v') (*ex c q*) **using** *4* **by** *auto*
next
fix b b_1 b_2 u
assume *5*: *R.path* (b @ $[a]$) (*ex c q*)
assume *6*: *Ind* $\{a\}$ (*set b*)
assume *7*: *set b* $\subseteq$ *invisible*
assume *8*: $b =_F b_1$ @ b_2
assume *9*: b_1 @− $u =_I v'$
assume *10*: *Ind* (*set* b_2) (*sset u*)
show *thesis*
proof (*rule less*(*2*))
show *R.path* ((($[c]$ @ b) @ $[a]$) q **using** *1 5* **by** *auto*
show *Ind* $\{a\}$ (*set* ($[c]$ @ b)) **using** *6 2* **by** *auto*
have *11*: $c \in$ *invisible*
using *ample-invisible ample-subset reduction-ample less*(*3*) *0 1* **by**
blast
show *set* ($[c]$ @ b) $\subseteq$ *invisible* **using** *7 11* **by** *auto*
have $[c]$ @ $b =_F [c]$ @ b_1 @ b_2 **using** *8* **by** *blast*
also have $[c]$ @ b_1 @ b_2 = ($[c]$ @ b_1) @ b_2 **by** *simp*
finally show $[c]$ @ $b =_F$ ($[c]$ @ b_1) @ b_2 **by** *this*
show ($[c]$ @ b_1) @− $u =_I v$

proof −
have *10*: *Ind* (*set* [*a*]) (*set* [*c*]) **using** *2* **by** *auto*
have *11*: [*a*] @ [*c*] $=_F$ [*c*] @ [*a*] **using** *10* **by** *blast*
have [*a*] @− *v* $=_I$ [*c*] @− [*a*] @− *v*′ **using** *3* **by** *this*
also have . . . = ([*c*] @ [*a*]) @− *v*′ **by** *simp*
also have . . . $=_I$ ([*a*] @ [*c*]) @− *v*′ **using** *11* **by** *blast*
also have . . . = [*a*] @− [*c*] @− *v*′ **by** *simp*
finally have *12*: [*a*] @− *v* $=_I$ [*a*] @− [*c*] @− *v*′ **by** *this*
have *12*: *v* $=_I$ [*c*] @− *v*′ **using** *12* **by** *blast*
have ([*c*] @ b_1) @− *u* = [*c*] @− b_1 @− *u* **by** *simp*
also have . . . $=_I$ [*c*] @− *v*′ **using** *9* **by** *blast*
also have . . . $=_I$ *v* **using** *12* **by** *blast*
finally show *?thesis* **by** *this*
qed
show *Ind* (*set* b_2) (*sset* *u*) **using** *10* **by** *this*
qed
qed
next
case (*omitted*)
have *1*: {*a*. *ren a q*} ⊆ *invisible* **using** *omitted*(*1*) **by** *simp*
have *2*: *Ind* {*a*. *ren a q*} (*sset* ([*a*] @− *v*)) **using** *omitted*(*2*) **by** *simp*
obtain *c* **where** *3*: *ren c q*
proof −
have *1*: *en a q* **using** *less*(*4*) **by** *auto*
show *?thesis* **using** *reduction-ample ample-nonempty less*(*3*) *1 that* **by** *blast*
qed
have *4*: *Ind* (*set* [*c*]) (*sset* ([*a*] @− *v*)) **using** *2 3* **by** *auto*
have *6*: *path* [*c*] *q* **using** *reduction-ample ample-subset less*(*3*) *3* **by** *auto*
have *7*: *run* ([*a*] @− *v*) (*target* [*c*] *q*) **using** *diamond-fin-word-inf-word 4 6 less*(*4*) **by** *this*
show *?thesis*
proof (*rule less*(*1*))
show *src* (*ex c q*) *q*
using *reduction-ample ample-wellfounded ample-subset less*(*3*) *0 3* **by** *blast*
show *ex c q* ∈ *nodes* **using** *less*(*3*) *6* **by** *auto*
show *run* ([*a*] @− *v*) (*ex c q*) **using** *7* **by** *auto*
next
fix *b s* b_1 b_2 *u*
assume *5*: *R.path* (*b* @ [*a*]) (*ex c q*)
assume *6*: *Ind* {*a*} (*set b*)
assume *7*: *set b* ⊆ *invisible*
assume *8*: *b* $=_F$ b_1 @ b_2
assume *9*: b_1 @− *u* $=_I$ *v*
assume *10*: *Ind* (*set* b_2) (*sset u*)
show *thesis*
proof (*rule less*(*2*))
show *R.path* (([*c*] @ *b*) @ [*a*]) *q* **using** *3 5* **by** *auto*

show *Ind* $\{a\}$ (*set* ([c] @ b))
proof −
have *1*: *ind c a* **using** *4* **by** *simp*
have *2*: *ind a c* **using** *independence-symmetric 1* **by** *this*
show *?thesis* **using** *6 2* **by** *auto*
qed
have *11*: $c \in$ *invisible* **using** *1 3* **by** *auto*
show *set* ([c] @ b) $\subseteq$ *invisible* **using** *7 11* **by** *auto*
have *12*: *Ind* (*set* [c]) (*set* b_1)
proof −
have *1*: *set* $b_1 \subseteq$ *sset v* **using** *9* **by** *force*
have *2*: *Ind* (*set* [c]) (*sset v*) **using** *4* **by** *simp*
show *?thesis* **using** *1 2* **by** *auto*
qed
have [c] @ $b =_F$ [c] @ b_1 @ b_2 **using** *8* **by** *blast*
also have ... = ([c] @ b_1) @ b_2 **by** *simp*
also have ... $=_F$ (b_1 @ [c]) @ b_2 **using** *12* **by** *blast*
also have ... = b_1 @ [c] @ b_2 **by** *simp*
finally show [c] @ $b =_F b_1$ @ [c] @ b_2 **by** *this*
show b_1 @− $u =_I v$ **using** *9* **by** *this*
have *13*: *Ind* (*set* [c]) (*sset u*)
proof −
have *1*: *sset u* $\subseteq$ *sset v* **using** *9* **by** *force*
have *2*: *Ind* (*set* [c]) (*sset v*) **using** *4* **by** *simp*
show *?thesis* **using** *1 2* **by** *blast*
qed
show *Ind* (*set* ([c] @ b_2)) (*sset u*) **using** *10 13* **by** *auto*
qed
qed
qed
qed
qed

inductive *reduced-run* :: *'state* ⇒ *'action list* ⇒ *'action stream* ⇒ *'action list* ⇒
'action list ⇒ *'action list* ⇒ *'action list* ⇒ *'action stream* ⇒ *bool*
where
init: *reduced-run q* [] *v* [] [] [] [] *v* |
absorb: *reduced-run q* v_1 ([a] @− v_2) *l w* w_1 w_2 $u \Longrightarrow a \in$ *set l* $\Longrightarrow$
reduced-run q (v_1 @ [a]) v_2 (*remove1 a l*) *w* w_1 w_2 *u* |
extend: *reduced-run q* v_1 ([a] @− v_2) *l w* w_1 w_2 $u \Longrightarrow a \notin$ *set l* $\Longrightarrow$
R.path (b @ [a]) (*target w q*) $\Longrightarrow$
Ind $\{a\}$ (*set b*) $\Longrightarrow$ *set b* $\subseteq$ *invisible* $\Longrightarrow$
$b =_F b_1$ @ $b_2 \Longrightarrow$ [a] @− b_1 @− $u' =_I u \Longrightarrow$ *Ind* (*set* b_2) (*sset* u') $\Longrightarrow$
reduced-run q (v_1 @ [a]) v_2 (l @ b_1) (w @ b @ [a]) (w_1 @ b_1 @ [a]) (w_2 @ b_2) u'

lemma *reduced-run-words-fin*:

assumes *reduced-run* $q\ v_1\ v_2\ l\ w\ w_1\ w_2\ u$
shows *R.path* $w\ q$
using *assms* **by** *induct auto*

lemma *reduced-run-invar-2*:
assumes *reduced-run* $q\ v_1\ v_2\ l\ w\ w_1\ w_2\ u$
shows $v_2 =_I l$ @− u
using *assms*
proof *induct*
case (*init* $q\ v$)
show *?case* **by** *simp*
next
case (*absorb* $q\ v_1\ a\ v_2\ l\ w\ w_1\ w_2\ u$)
obtain $l_1\ l_2$ **where** *10*: $l = l_1$ @ $[a]$ @ l_2 $a \notin$ *set* l_1
using *split-list-first*[*OF absorb*(*3*)] **by** *auto*
have *11*: *Ind* $\{a\}$ (*set* l_1)
proof (*rule le-fininf-ind′*)
show $[a] \preceq_{FI} l$ @− u **using** *absorb*(*2*) **by** *auto*
show $l_1 \preceq_{FI} l$ @− u **unfolding** *10*(*1*) **by** *auto*
show $a \notin$ *set* l_1 **using** *10*(*2*) **by** *this*
qed
have *12*: *Ind* (*set* $[a]$) (*set* l_1) **using** *11* **by** *auto*
have $[a]$ @ *remove1* $a\ l = [a]$ @ l_1 @ l_2 **unfolding** *10*(*1*) *remove1-append* **using** *10*(*2*) **by** *auto*
also have $\ldots =_F ([a]$ @ $l_1)$ @ l_2 **by** *simp*
also have $\ldots =_F (l_1$ @ $[a])$ @ l_2 **using** *12* **by** *blast*
also have $\ldots = l$ **unfolding** *10*(*1*) **by** *simp*
finally have *13*: $[a]$ @ *remove1* $a\ l =_F l$ **by** *this*
have $[a]$ @− *remove1* $a\ l$ @− $u = ([a]$ @ *remove1* $a\ l)$ @− u **unfolding** *conc-conc* **by** *simp*
also have $\ldots =_I l$ @− u **using** *13* **by** *blast*
also have $\ldots =_I [a]$ @− v_2 **using** *absorb*(*2*) **by** *auto*
finally show *?case* **by** *blast*
next
case (*extend* $q\ v_1\ a\ v_2\ l\ w\ w_1\ w_2\ u\ b\ b_1\ b_2\ u'$)
have *11*: *Ind* $\{a\}$ (*set* l)
proof (*rule le-fininf-ind′*)
show $[a] \preceq_{FI} l$ @− u **using** *extend*(*2*) **by** *auto*
show $l \preceq_{FI} l$ @− u **by** *auto*
show $a \notin$ *set* l **using** *extend*(*3*) **by** *this*
qed
have *11*: *Ind* (*set* $[a]$) (*set* l) **using** *11* **by** *auto*
have *12*: *eq-fin* ($[a]$ @ l) (l @ $[a]$) **using** *11* **by** *blast*
have *131*: *set* $b_1 \subseteq$ *set* b **using** *extend*(*7*) **by** *auto*
have *132*: *Ind* (*set* $[a]$) (*set* b) **using** *extend*(*5*) **by** *auto*
have *13*: *Ind* (*set* $[a]$) (*set* b_1) **using** *131 132* **by** *auto*
have *14*: *eq-fin* ($[a]$ @ b_1) (b_1 @ $[a]$) **using** *13* **by** *blast*
have $[a]$ @− ((l @ b_1) @− u') = ($[a]$ @ l) @− b_1 @− u' **by** *simp*
also have *eq-inf* $\ldots$ ((l @ $[a]$) @− b_1 @− u') **using** *12* **by** *blast*

also have $\ldots = l$ @− $[a]$ @− b_1 @− u' **by** *simp*
also have *eq-inf* $\ldots$ $(l$ @− $u)$ **using** *extend*$($*8*$)$ **by** *blast*
also have *eq-inf* $\ldots$ $([a]$ @− $v_2)$ **using** *extend*$($*2*$)$ **by** *auto*
finally show *?case* **by** *blast*
qed

lemma *reduced-run-invar-1*:
assumes *reduced-run* $q\ v_1\ v_2\ l\ w\ w_1\ w_2\ u$
shows v_1 @ $l =_F w_1$
using *assms*
proof *induct*
case $($*init* $q\ v)$
show *?case* **by** *simp*
next
case $($*absorb* $q\ v_1\ a\ v_2\ l\ w\ w_1\ w_2\ u)$
have *1*: $[a]$ @− $v_2 =_I l$ @− u **using** *reduced-run-invar-2 absorb*$($*1*$)$ **by** *this*
obtain $l_1\ l_2$ **where** *10*: $l = l_1$ @ $[a]$ @ l_2 $a \notin$ *set* l_1
using *split-list-first*$[$*OF absorb*$($*3*$)]$ **by** *auto*
have *11*: *Ind* $\{a\}$ $($*set* $l_1)$
proof $($*rule le-fininf-ind*$')$
show $[a] \preceq_{FI} l$ @− u **using** *1* **by** *auto*
show $l_1 \preceq_{FI} l$ @− u **unfolding** *10*$($*1*$)$ **by** *auto*
show $a \notin$ *set* l_1 **using** *10*$($*2*$)$ **by** *this*
qed
have *12*: *Ind* $($*set* $[a])$ $($*set* $l_1)$ **using** *11* **by** *auto*
have $[a]$ @ *remove1* $a\ l = [a]$ @ l_1 @ l_2 **unfolding** *10*$($*1*$)$ *remove1-append* **using** *10*$($*2*$)$ **by** *auto*
also have $\ldots =_F ([a]$ @ $l_1)$ @ l_2 **by** *simp*
also have $\ldots =_F (l_1$ @ $[a])$ @ l_2 **using** *12* **by** *blast*
also have $\ldots = l$ **unfolding** *10*$($*1*$)$ **by** *simp*
finally have *13*: $[a]$ @ *remove1* $a\ l =_F l$ **by** *this*
have $w_1 =_F v_1$ @ l **using** *absorb*$($*2*$)$ **by** *auto*
also have $\ldots =_F v_1$ @ $([a]$ @ *remove1* $a\ l)$ **using** *13* **by** *blast*
also have $\ldots = (v_1$ @ $[a])$ @ *remove1* $a\ l$ **by** *simp*
finally show *?case* **by** *auto*
next
case $($*extend* $q\ v_1\ a\ v_2\ l\ w\ w_1\ w_2\ u\ b\ b_1\ b_2\ u')$
have *1*: $[a]$ @− $v_2 =_I l$ @− u **using** *reduced-run-invar-2 extend*$($*1*$)$ **by** *this*
have *11*: *Ind* $\{a\}$ $($*set* $l)$
proof $($*rule le-fininf-ind*$')$
show $[a] \preceq_{FI} l$ @− u **using** *1* **by** *auto*
show $l \preceq_{FI} l$ @− u **by** *auto*
show $a \notin$ *set* l **using** *extend*$($*3*$)$ **by** *auto*
qed
have *11*: *Ind* $($*set* $[a])$ $($*set* $l)$ **using** *11* **by** *auto*
have *12*: *eq-fin* $([a]$ @ $l)$ $(l$ @ $[a])$ **using** *11* **by** *blast*
have *131*: *set* $b_1 \subseteq$ *set* b **using** *extend*$($*7*$)$ **by** *auto*
have *132*: *Ind* $($*set* $[a])$ $($*set* $b)$ **using** *extend*$($*5*$)$ **by** *auto*
have *13*: *Ind* $($*set* $[a])$ $($*set* $b_1)$ **using** *131 132* **by** *auto*

have *14*: *eq-fin* $([a] \mathbin{@} b_1)$ $(b_1 \mathbin{@} [a])$ **using** *13* **by** *blast*
have *eq-fin* $(w_1 \mathbin{@} b_1 \mathbin{@} [a])$ $(w_1 \mathbin{@} [a] \mathbin{@} b_1)$ **using** *14* **by** *blast*
also have *eq-fin* ... $((v_1 \mathbin{@} l) \mathbin{@} [a] \mathbin{@} b_1)$ **using** *extend(2)* **by** *blast*
also have *eq-fin* ... $(v_1 \mathbin{@} (l \mathbin{@} [a]) \mathbin{@} b_1)$ **by** *simp*
also have *eq-fin* ... $(v_1 \mathbin{@} ([a] \mathbin{@} l) \mathbin{@} b_1)$ **using** *12* **by** *blast*
also have ... $= (v_1 \mathbin{@} [a]) \mathbin{@} l \mathbin{@} b_1$ **by** *simp*
finally show *?case* **by** *auto*
qed

lemma *reduced-run-invisible*:
assumes *reduced-run* $q\ v_1\ v_2\ l\ w\ w_1\ w_2\ u$
shows *set* $w_2 \subseteq$ *invisible*
using *assms*
proof *induct*
case (*init q v*)
show *?case* **by** *simp*
next
case (*absorb* $q\ v_1\ a\ v_2\ l\ w\ w_1\ w_2\ u$)
show *?case* **using** *absorb(2)* **by** *this*
next
case (*extend* $q\ v_1\ a\ v_2\ l\ w\ w_1\ w_2\ u\ b\ b_1\ b_2\ u'$)
have *1*: *set* $b_2 \subseteq$ *set b* **using** *extend(7)* **by** *auto*
show *?case* **unfolding** *set-append* **using** *extend(2) extend(6) 1* **by** *blast*
qed

lemma *reduced-run-ind*:
assumes *reduced-run* $q\ v_1\ v_2\ l\ w\ w_1\ w_2\ u$
shows *Ind* (*set* w_2) (*sset u*)
using *assms*
proof *induct*
case (*init q v*)
show *?case* **by** *simp*
next
case (*absorb* $q\ v_1\ a\ v_2\ l\ w\ w_1\ w_2\ u$)
show *?case* **using** *absorb(2)* **by** *this*
next
case (*extend* $q\ v_1\ a\ v_2\ l\ w\ w_1\ w_2\ u\ b\ b_1\ b_2\ u'$)
have *1*: *sset* $u' \subseteq$ *sset u* **using** *extend(8)* **by** *force*
show *?case* **using** *extend(2) extend(9) 1* **unfolding** *set-append* **by** *blast*
qed

lemma *reduced-run-decompose*:
assumes *reduced-run* $q\ v_1\ v_2\ l\ w\ w_1\ w_2\ u$
shows $w =_F w_1 \mathbin{@} w_2$
using *assms*
proof *induct*
case (*init q v*)
show *?case* **by** *simp*
next

case (*absorb* q v_1 a v_2 l w w_1 w_2 u)
show *?case* **using** *absorb*(*2*) **by** *this*
next
case (*extend* q v_1 a v_2 l w w_1 w_2 u b b_1 b_2 u')
have *1*: *Ind* (*set* $[a]$) (*set* b_2) **using** *extend*(*5*) *extend*(*7*) **by** *auto*
have *2*: *Ind* (*set* w_2) (*set* (b_1 @ $[a]$))
proof −
have *1*: *Ind* (*set* w_2) (*sset* u) **using** *reduced-run-ind extend*(*1*) **by** *this*
have *2*: $u =_I [a]$ @− b_1 @− u' **using** *extend*(*8*) **by** *auto*
have *3*: *sset* u = *sset* ($[a]$ @− b_1 @− u') **using** *2* **by** *blast*
show *?thesis* **unfolding** *set-append* **using** *1 3* **by** *simp*
qed
have w @ b @ $[a]$ $=_F$ (w_1 @ w_2) @ b @ $[a]$ **using** *extend*(*2*) **by** *blast*
also have ... $=_F$ (w_1 @ w_2) @ (b_1 @ b_2) @ $[a]$ **using** *extend*(*7*) **by** *blast*
also have ... = w_1 @ w_2 @ b_1 @ (b_2 @ $[a]$) **by** *simp*
also have ... $=_F$ w_1 @ w_2 @ b_1 @ ($[a]$ @ b_2) **using** *1* **by** *blast*
also have ... $=_F$ w_1 @ (w_2 @ (b_1 @ $[a]$)) @ b_2 **by** *simp*
also have ... $=_F$ w_1 @ ((b_1 @ $[a]$) @ w_2) @ b_2 **using** *2* **by** *blast*
also have ... $=_F$ (w_1 @ b_1 @ $[a]$) @ w_2 @ b_2 **by** *simp*
finally show *?case* **by** *this*
qed

lemma *reduced-run-project*:
assumes *reduced-run* q v_1 v_2 l w w_1 w_2 u
shows *project visible* w_1 = *project visible* w
proof −
have *1*: w_1 @ w_2 $=_F$ w **using** *reduced-run-decompose assms* **by** *auto*
have *2*: *set* $w_2 \subseteq$ *invisible* **using** *reduced-run-invisible assms* **by** *this*
have *3*: *project visible* w_2 = [] **unfolding** *filter-empty-conv* **using** *2* **by** *auto*
have *project visible* w_1 = *project visible* w_1 @ *project visible* w_2 **using** *3* **by** *simp*
also have ... = *project visible* (w_1 @ w_2) **by** *simp*
also have ... = *list-of* (*lproject visible* (*llist-of* (w_1 @ w_2))) **by** *simp*
also have ... = *list-of* (*lproject visible* (*llist-of* w))
using *eq-fin-lproject-visible 1* **by** *metis*
also have ... = *project visible* w **by** *simp*
finally show *?thesis* **by** *this*
qed

lemma *reduced-run-length-1*:
assumes *reduced-run* q v_1 v_2 l w w_1 w_2 u
shows *length* $v_1 \leq$ *length* w_1
using *reduced-run-invar-1 assms* **by** *force*
lemma *reduced-run-length*:
assumes *reduced-run* q v_1 v_2 l w w_1 w_2 u
shows *length* $v_1 \leq$ *length* w
proof −
have *length* $v_1 \leq$ *length* w_1 **using** *reduced-run-length-1 assms* **by** *this*
also have ... $\leq$ *length* w **using** *reduced-run-decompose assms* **by** *force*

finally show *?thesis* **by** *this*
qed

lemma *reduced-run-step*:
assumes $q \in$ *nodes run* $(v_1$ @− $[a]$ @− $v_2)$ q
assumes *reduced-run* q v_1 $([a]$ @− $v_2)$ l w w_1 w_2 u
obtains l' w' $w_1{}'$ $w_2{}'$ u'
where *reduced-run* q $(v_1$ @ $[a])$ v_2 l' $(w$ @ $w')$ $(w_1$ @ $w_1{}')$ $(w_2$ @ $w_2{}')$ u'
proof (*cases* $a \in$ *set* l)
case *True*
show *?thesis*
proof (*rule that*, *rule absorb*)
show *reduced-run* q v_1 $([a]$ @− $v_2)$ l $(w$ @ $[])$ $(w_1$ @ $[])$ $(w_2$ @ $[])$ u **using** *assms*(*3*) **by** *simp*
show $a \in$ *set* l **using** *True* **by** *this*
qed
next
case *False*
have *1*: v_1 @ $l =_F w_1$ **using** *reduced-run-invar-1 assms*(*3*) **by** *this*
have *2*: $[a]$ @− $v_2 =_I l$ @− u **using** *reduced-run-invar-2 assms*(*3*) **by** *this*
have *3*: $w =_F w_1$ @ w_2 **using** *reduced-run-decompose assms*(*3*) **by** *this*
have v_1 @ l @ $w_2 = (v_1$ @ $l)$ @ w_2 **by** *simp*
also have $\ldots =_F w_1$ @ w_2 **using** *1* **by** *blast*
also have $\ldots =_F w$ **using** *3* **by** *blast*
finally have *4*: v_1 @ l @ $w_2 =_F w$ **by** *this*
have *5*: *run* $((v_1$ @ $l)$ @− w_2 @− $u)$ q
proof (*rule diamond-fin-word-inf-word′*)
show *Ind* (*set* w_2) (*sset* u) **using** *reduced-run-ind assms*(*3*) **by** *this*
have *6*: *R.path* w q **using** *reduced-run-words-fin assms*(*3*) **by** *this*
have *7*: *path* w q **using** *reduction-words-fin assms*(*1*) *6* **by** *auto*
show *path* $((v_1$ @ $l)$ @ $w_2)$ q **using** *eq-fin-word 4 7* **by** *auto*
have *8*: v_1 @− $[a]$ @− $v_2 =_I v_1$ @− l @− u **using** *2* **by** *blast*
show *run* $((v_1$ @ $l)$ @− $u)$ q **using** *eq-inf-word assms*(*2*) *8* **by** *auto*
qed
have *6*: *run* $(w$ @− $u)$ q **using** *eq-inf-word 4 5* **by** (*metis eq-inf-concat-end shift-append*)
have *7*: $[a] \preceq_{FI} l$ @− u **using** *2* **by** *blast*
have *8*: $[a] \preceq_{FI} u$ **using** *le-fininf-not-member′ 7 False* **by** *this*
obtain u' **where** *9*: $u =_I [a]$ @− u' **using** *8* **by** *rule*
have *101*: *target* w $q \in$ *nodes* **using** *assms*(*1*) *6* **by** *auto*
have *10*: *run* $([a]$ @− $u')$ (*target* w q) **using** *eq-inf-word 9 6* **by** *blast*
obtain b b_1 b_2 u'' **where** *11*:
R.path $(b$ @ $[a])$ (*target* w q)
Ind $\{a\}$ (*set* b) *set* $b \subseteq$ *invisible*
$b =_F b_1$ @ b_2 b_1 @− $u'' =_I u'$ *Ind* (*set* b_2) (*sset* u'')
using *reduction-chunk 101 10* **by** *this*
show *?thesis*
proof (*rule that*, *rule extend*)
show *reduced-run* q v_1 $([a]$ @− $v_2)$ l w w_1 w_2 u **using** *assms*(*3*) **by** *this*

show $a \notin$ *set l* **using** *False* **by** *this*
show *R.path* (*b* @ [*a*]) (*target w q*) **using** *11*(*1*) **by** *this*
show *Ind* {*a*} (*set b*) **using** *11*(*2*) **by** *this*
show *set b* $\subseteq$ *invisible* **using** *11*(*3*) **by** *this*
show $b =_F b_1$ @ b_2 **using** *11*(*4*) **by** *this*
show [*a*] @− b_1 @− $u'' =_I u$ **using** *9 11*(*5*) **by** *blast*
show *Ind* (*set* b_2) (*sset* u'') **using** *11*(*6*) **by** *this*
qed
qed

lemma *reduction-word*:
assumes $q \in$ *nodes run v q*
obtains *u w*
where
R.run w q
$v =_I u\ u \preceq_I w$
lproject visible (*llist-of-stream u*) = *lproject visible* (*llist-of-stream w*)
proof −
define *P* **where** $P \equiv \lambda\ k\ w\ w_1.\ \exists\ l\ w_2\ u.$ *reduced-run q* (*stake k v*) (*sdrop k v*) *l w* $w_1\ w_2$ *u*
obtain *w* w_1 **where** *1*: $\bigwedge k.$ *P k* (*w k*) (w_1 *k*) *chain w chain* w_1
proof (*rule chain-construct-2'*[*of P*])
show *P 0* [] [] **unfolding** *P-def* **using** *init* **by** *force*
next
fix *k w* w_1
assume *1*: *P k w* w_1
obtain *l* w_2 *u* **where** *2*: *reduced-run q* (*stake k v*) (*sdrop k v*) *l w* $w_1\ w_2$ *u*
using *1* **unfolding** *P-def* **by** *auto*
obtain $l'\ w'\ w_1'\ w_2'\ u'$ **where** *3*:
reduced-run q (*stake k v* @ [*v* !! *k*]) (*sdrop* (*Suc k*) *v*) l' (*w* @ w') (w_1 @ w_1') (w_2 @ w_2') u'
proof (*rule reduced-run-step*)
show $q \in$ *nodes* **using** *assms*(*1*) **by** *this*
show *run* (*stake k v* @− [*v* !! *k*] @− *sdrop* (*Suc k*) *v*) *q*
using *assms*(*2*) **by** (*metis shift-append stake-Suc stake-sdrop*)
show *reduced-run q* (*stake k v*) ([*v* !! *k*] @− *sdrop* (*Suc k*) *v*) *l w* $w_1\ w_2$ *u*
using *2* **by** (*metis sdrop-simps shift.simps stream.collapse*)
qed
show $\exists\ w'\ w_1'.$ *P* (*Suc k*) $w'\ w_1' \wedge w \le w' \wedge w_1 \le w_1'$
unfolding *P-def* **using** *3* **by** (*metis prefix-fin-append stake-Suc*)
show $k \le$ *length w* **using** *reduced-run-length 2* **by** *force*
show $k \le$ *length* w_1 **using** *reduced-run-length-1 2* **by** *force*
qed *rule*
obtain *l* w_2 *u* **where** *2*:
$\bigwedge k.$ *reduced-run q* (*stake k v*) (*sdrop k v*) (*l k*) (*w k*) (w_1 *k*) (w_2 *k*) (*u k*)
using *1*(*1*) **unfolding** *P-def* **by** *metis*
show *?thesis*
proof

show *R.run* (*Word-Prefixes.limit w*) *q* **using** *reduced-run-words-fin 1*(*2*) *2* **by** *blast*
show $v =_I$ *Word-Prefixes.limit* w_1
proof
show $v \preceq_I$ *Word-Prefixes.limit* w_1
proof (*rule le-infI-chain-right'*)
show *chain* w_1 **using** *1*(*3*) **by** *this*
show $\bigwedge$ *k. stake k v* $\preceq_F$ w_1 *k* **using** *reduced-run-invar-1*[*OF 2*] **by** *auto*
qed
show *Word-Prefixes.limit* w_1 $\preceq_I$ *v*
proof (*rule le-infI-chain-left*)
show *chain* w_1 **using** *1*(*3*) **by** *this*
next
fix *k*
have w_1 *k* $=_F$ *stake k v* @ *l k* **using** *reduced-run-invar-1 2* **by** *blast*
also have ... $\leq_{FI}$ *stake k v* @− *l k* @− *u k* **by** *auto*
also have ... $=_I$ *stake k v* @− *sdrop k v* **using** *reduced-run-invar-2*[*OF 2*] **by** *blast*
also have ... = *v* **by** *simp*
finally show w_1 *k* $\preceq_{FI}$ *v* **by** *this*
qed
qed
show *Word-Prefixes.limit* w_1 $\preceq_I$ *Word-Prefixes.limit w*
proof (*rule le-infI-chain-left*)
show *chain* w_1 **using** *1*(*3*) **by** *this*
next
fix *k*
have w_1 *k* $\preceq_F$ *w k* **using** *reduced-run-decompose*[*OF 2*] **by** *blast*
also have ... $\leq_{FI}$ *Word-Prefixes.limit w* **using** *chain-prefix-limit 1*(*2*) **by** *this*
finally show w_1 *k* $\preceq_{FI}$ *Word-Prefixes.limit w* **by** *this*
qed
show *lproject visible* (*llist-of-stream* (*Word-Prefixes.limit* w_1)) =
lproject visible (*llist-of-stream* (*Word-Prefixes.limit w*))
using *lproject-eq-limit-chain reduced-run-project 1* **unfolding** *P-def* **by** *metis*
qed
qed

lemma *reduction-equivalent*:
assumes *q* ∈ *nodes run u q*
obtains *v*
where *R.run v q snth* (*smap int* (*q* ## *trace u q*)) ≈ *snth* (*smap int* (*q* ## *trace v q*))
proof −
obtain *v w* **where** *1*: *R.run w q u* $=_I$ *v v* $\preceq_I$ *w*
lproject visible (*llist-of-stream v*) = *lproject visible* (*llist-of-stream w*)
using *reduction-word assms* **by** *this*

```
      show ?thesis
      proof
        show R.run w q using 1(1) by this
        show snth (smap int (q ## trace u q)) ≈ snth (smap int (q ## trace w q))
        proof (rule execute-inf-visible)
          show run u q using assms(2) by this
          show run w q using reduction-words-inf assms(1) 1(1) by auto
          have u =_I v using 1(2) by this
          also have ... ⪯_I w using 1(3) by this
          finally show u ⪯_I w by this
          show w ⪯_I w by simp
          have lproject visible (llist-of-stream u) = lproject visible (llist-of-stream v)
            using eq-inf-lproject-visible 1(2) by this
          also have ... = lproject visible (llist-of-stream w) using 1(4) by this
        finally show lproject visible (llist-of-stream u) = lproject visible (llist-of-stream
w) by this
        qed
      qed
    qed

    lemma reduction-language-subset: R.language ⊆ S.language
      unfolding S.language-def R.language-def using reduction-words-inf by blast

    lemma reduction-language-stuttering:
      assumes u ∈ S.language
      obtains v
      where v ∈ R.language snth u ≈ snth v
    proof −
      obtain q v where 1: u = smap int (q ## trace v q) init q S.run v q using
assms by rule
        obtain v' where 2: R.run v' q snth (smap int (q ## trace v q)) ≈ snth
(smap int (q ## trace v' q))
        using reduction-equivalent 1(2, 3) by blast
      show ?thesis
      proof (intro that R.languageI)
        show smap int (q ## trace v' q) = smap int (q ## trace v' q) by rule
        show init q using 1(2) by this
        show R.run v' q using 2(1) by this
        show snth u ≈ snth (smap int (q ## trace v' q)) unfolding 1(1) using
2(2) by this
      qed
    qed

  end

end
```

19 LTL Formulae

theory *Formula*
imports
Basics/Stuttering
Stuttering-Equivalence.PLTL
begin

locale *formula* =
fixes φ :: *'a pltl*
begin

definition *language* :: *'a stream set*
where *language* $\equiv$ {*w. snth w* $\models_p \varphi$}

lemma *language-entails*[*iff*]: $w \in$ *language* $\longleftrightarrow$ *snth w* $\models_p \varphi$ **unfolding** *language-def* **by** *simp*

end

locale *formula-next-free* =
formula φ
for φ :: *'a pltl*
+
assumes *next-free*: *next-free* φ
begin

lemma *stutter-equivalent-entails*[*dest*]: $u \approx v \Longrightarrow u \models_p \varphi \longleftrightarrow v \models_p \varphi$
using *next-free-stutter-invariant next-free* **by** *blast*

end

end

20 Correctness Theorem of Partial Order Reduction

theory *Ample-Correctness*
imports
Ample-Abstract
Formula
begin

locale *ample-correctness* =
S: *transition-system-complete ex en init int* +
R: *transition-system-complete ex ren init int* +
F: *formula-next-free* φ +
ample-abstract ex en init int ind src ren

for *ex* :: $'action \Rightarrow 'state \Rightarrow 'state$
and *en* :: $'action \Rightarrow 'state \Rightarrow bool$
and *init* :: $'state \Rightarrow bool$
and *int* :: $'state \Rightarrow 'interpretation$
and *ind* :: $'action \Rightarrow 'action \Rightarrow bool$
and *src* :: $'state \Rightarrow 'state \Rightarrow bool$
and *ren* :: $'action \Rightarrow 'state \Rightarrow bool$
and φ :: *'interpretation pltl*
begin

lemma *reduction-language-indistinguishable*:
assumes $R.language \subseteq F.language$
shows $S.language \subseteq F.language$
proof
fix *u*
assume *1*: $u \in S.language$
obtain *v* **where** *2*: $v \in R.language$ *snth u* $\approx$ *snth v* **using** *reduction-language-stuttering 1* **by** *this*
have *3*: $v \in F.language$ **using** *assms 2(1)* **by** *rule*
show $u \in F.language$ **using** *2(2) 3* **by** *auto*
qed

theorem *reduction-correct*: $S.language \subseteq F.language \longleftrightarrow R.language \subseteq F.language$
using *reduction-language-subset reduction-language-indistinguishable* **by** *blast*

end

end

21 Static Analysis for Partial Order Reduction

theory *Ample-Analysis*
imports
Ample-Abstract
begin

locale *transition-system-ample* =
transition-system-sticky ex en init int sticky +
transition-system-interpreted-traces ex en int ind
for *ex* :: $'action \Rightarrow 'state \Rightarrow 'state$
and *en* :: $'action \Rightarrow 'state \Rightarrow bool$
and *init* :: $'state \Rightarrow bool$
and *int* :: $'state \Rightarrow 'interpretation$
and *sticky* :: *'action set*
and *ind* :: $'action \Rightarrow 'action \Rightarrow bool$
begin

sublocale *ample-base ex en int ind* $scut^{-1-1}$ **by** *unfold-locales*

lemma *restrict-ample-set*:
assumes $s \in$ *nodes*
assumes $A \cap \{a.\ en\ a\ s\} \neq \{\}$ $A \cap \{a.\ en\ a\ s\} \cap sticky = \{\}$
assumes *Ind* $(A \cap \{a.\ en\ a\ s\})$ $(executable - A)$
assumes $\bigwedge$ $w.\ path\ w\ s \Longrightarrow A \cap \{a.\ en\ a\ s\} \cap set\ w = \{\} \Longrightarrow A \cap set\ w = \{\}$
shows *ample-set* s $(A \cap \{a.\ en\ a\ s\})$
unfolding *ample-set-def*
proof (*intro conjI allI impI*)
show $A \cap \{a.\ en\ a\ s\} \subseteq \{a.\ en\ a\ s\}$ **by** *simp*
next
show $A \cap \{a.\ en\ a\ s\} \neq \{\}$ **using** *assms*(*2*) **by** *this*
next
fix a
assume *1*: $a \in A \cap \{a.\ en\ a\ s\}$
show $scut^{-1-1}$ $(ex\ a\ s)$ s
proof (*rule no-cut-scut*)
show $s \in$ *nodes* **using** *assms*(*1*) **by** *this*
show *en a s* **using** *1* **by** *simp*
show $a \notin$ *sticky* **using** *assms*(*3*) *1* **by** *auto*
qed
next
have *1*: $A \cap \{a.\ en\ a\ s\} \subseteq$ *executable* **using** *executable assms*(*1*) **by** *blast*
show $A \cap \{a.\ en\ a\ s\} \subseteq$ *invisible* **using** *executable-visible-sticky assms*(*3*) *1* **by** *blast*
next
fix w
assume *1*: *path w s* $A \cap \{a.\ en\ a\ s\} \cap set\ w = \{\}$
have *2*: $A \cap set\ w = \{\}$ **using** *assms*(*5*) *1* **by** *this*
have *3*: $set\ w \subseteq$ *executable* **using** *assms*(*1*) *1*(*1*) **by** *rule*
show *Ind* $(A \cap \{a.\ en\ a\ s\})$ $(set\ w)$ **using** *assms*(*4*) *2 3* **by** *blast*
qed

end

locale *transition-system-concurrent* =
transition-system-initial ex en init
for *ex* :: $'action \Rightarrow 'state \Rightarrow 'state$
and *en* :: $'action \Rightarrow 'state \Rightarrow bool$
and *init* :: $'state \Rightarrow bool$
+
fixes *procs* :: $'state \Rightarrow 'process\ set$
fixes *pac* :: $'process \Rightarrow 'action\ set$
fixes *psen* :: $'process \Rightarrow 'state \Rightarrow 'action\ set$
assumes *procs-finite*: $s \in nodes \Longrightarrow finite\ (procs\ s)$
assumes *psen-en*: $s \in nodes \Longrightarrow pac\ p \cap \{a.\ en\ a\ s\} \subseteq psen\ p\ s$
assumes *psen-ex*: $s \in nodes \Longrightarrow a \in \{a.\ en\ a\ s\} - pac\ p \Longrightarrow psen\ p\ (ex\ a\ s) = psen\ p\ s$
begin

```
lemma psen-fin-word:
  assumes s ∈ nodes path w s pac p ∩ set w = {}
  shows psen p (target w s) = psen p s
using assms(2, 1, 3)
proof induct
  case (nil s)
  show ?case by simp
next
  case (cons a s w)
  have 1: ex a s ∈ nodes using cons(4, 1) by rule
  have psen p (target (a # w) s) = psen p (target w (ex a s)) by simp
  also have ... = psen p (ex a s) using cons 1 by simp
  also have ... = psen p s using psen-ex cons by simp
  finally show ?case by this
qed

lemma en-fin-word:
  assumes ⋀ r a b. r ∈ nodes ⟹ a ∈ psen p s − {a. en a s} ⟹ b ∈ {a. en
a r} − pac p ⟹
     en a (ex b r) ⟹ en a r
  assumes s ∈ nodes path w s pac p ∩ set w = {}
  shows pac p ∩ {a. en a (target w s)} ⊆ pac p ∩ {a. en a s}
using assms
proof (induct w rule: rev-induct)
  case Nil
  show ?case by simp
next
  case (snoc b w)
  show ?case
  proof (safe, rule ccontr)
    fix a
    assume 2: a ∈ pac p en a (target (w @ [b]) s) ¬ en a s
    have 3: a ∈ psen p s
    proof −
     have 3: psen p (target (w @ [b]) s) = psen p s using psen-fin-word snoc(3,
4, 5) by this
      have 4: target (w @ [b]) s ∈ nodes using snoc(3, 4) by rule
      have 5: a ∈ psen p (target (w @ [b]) s) using psen-en 4 2(1, 2) by auto
      show ?thesis using 2(1) 3 5 by auto
    qed
    have 4: en a (target w s)
    proof (rule snoc(2))
      show target w s ∈ nodes using snoc(3, 4) by auto
      show a ∈ psen p s − {a. en a s} using 2(3) 3 by simp
      show b ∈ {a. en a (target w s)} − pac p using snoc(4, 5) by auto
      show en a (ex b (target w s)) using 2(2) by simp
    qed
    have 5: pac p ∩ {a. en a (target w s)} ⊆ pac p ∩ {a. en a s}
```

proof $(rule\ snoc(1))$
show $\bigwedge r\ a\ b.\ r \in nodes \Longrightarrow a \in psen\ p\ s - \{a.\ en\ a\ s\} \Longrightarrow b \in \{a.\ en\ a\ r\} - pac\ p \Longrightarrow$
$en\ a\ (ex\ b\ r) \Longrightarrow en\ a\ r$ **using** $snoc(2)$ **by** *this*
show $s \in nodes$ **using** $snoc(3)$ **by** *this*
show $path\ w\ s$ **using** $snoc(4)$ **by** *auto*
show $pac\ p \cap set\ w = \{\}$ **using** $snoc(5)$ **by** *auto*
qed
have 6: $en\ a\ s$ **using** $2(1)\ 4\ 5$ **by** *auto*
show *False* **using** $2(3)\ 6$ **by** *simp*
qed
qed

lemma *pac-en-blocked*:
assumes $\bigwedge r\ a\ b.\ r \in nodes \Longrightarrow a \in psen\ p\ s - \{a.\ en\ a\ s\} \Longrightarrow b \in \{a.\ en\ a\ r\} - pac\ p \Longrightarrow$
$en\ a\ (ex\ b\ r) \Longrightarrow en\ a\ r$
assumes $s \in nodes\ path\ w\ s\ pac\ p \cap \{a.\ en\ a\ s\} \cap set\ w = \{\}$
shows $pac\ p \cap set\ w = \{\}$
using *words-fin-blocked en-fin-word assms* **by** *metis*

abbreviation $proc\ a \equiv \{p.\ a \in pac\ p\}$
abbreviation $Proc\ A \equiv \bigcup\ a \in A.\ proc\ a$

lemma *psen-simple*:
assumes $Proc\ (psen\ p\ s) = \{p\}$
assumes $\bigwedge r\ a\ b.\ r \in nodes \Longrightarrow a \in psen\ p\ s - \{a.\ en\ a\ s\} \Longrightarrow en\ b\ r \Longrightarrow$
$proc\ a \cap proc\ b = \{\} \Longrightarrow en\ a\ (ex\ b\ r) \Longrightarrow en\ a\ r$
shows $\bigwedge r\ a\ b.\ r \in nodes \Longrightarrow a \in psen\ p\ s - \{a.\ en\ a\ s\} \Longrightarrow b \in \{a.\ en\ a\ r\} - pac\ p \Longrightarrow$
$en\ a\ (ex\ b\ r) \Longrightarrow en\ a\ r$
using *assms* **by** *force*

end

end

References

[1] C.-T. Chou and D. Peled. Formal verification of a partial-order reduction technique for model checking. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 241–257. Springer Berlin Heidelberg, 1996.

[2] D. Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8(1):39–64, 1996.