Relative Security

Andrei Popescu Jamie Wright

June 5, 2024

Abstract

This entry formalizes the notion of relative security, which can be used to model transient execution vulnerabilities in the style of Spectre and Meltdown. The notion was introduced in the CSF 2023 paper "Relative Security: Formally Modeling and (Dis)Proving Resilience Against Semantic Optimization Vulnerabilities" by Brijesh Dongol, Matt Griffin, Andrei Popescu and Jamie Wright [1].

It defines two versions of relative security: a finitary one (restricted to finite traces), and an infinitary one (working with both finite and infinite traces). It formalizes unwinding methods for verifying relative security in both the finitary and infinitary versions, and proves their soundness. The proof of soudness in the infinitary case is a substantial application of Isabelle's corecursion and coinduction infrastructure.

Contents

1 Finitary Relative Security

This theory formalizes the finitary version of relative security, more precisely the notion expressed in terms of finite traces.

theory *Relative-Security-fin*
imports *Preliminaries/Transition-System*
begin

declare *Let-def*[*simp*]

no-notation *relcomp* (**infixr** *O 75*)
no-notation *relcompp* (**infixr** *OO 75*)

1.1 Finite-trace versions of leakage models and attacker models

locale *Leakage-Mod-fin* = *System-Mod istate validTrans final*
for *istate* :: *'state* ⇒ *bool* **and** *validTrans* :: *'state* × *'state* ⇒ *bool* **and** *final* :: *'state* ⇒ *bool*
\+
fixes *S* :: *'state list* ⇒ *'secret list*
and *A* :: *'state trace* ⇒ *'act list*
and *O* :: *'state trace* ⇒ *'obs list*
and *leakVia* :: *'state list* ⇒ *'state list* ⇒ *'leak* ⇒ *bool*

locale *Attacker-Mod-fin* = *System-Mod istate validTrans final*
for *istate* :: *'state* ⇒ *bool* **and** *validTrans* :: *'state* × *'state* ⇒ *bool* **and** *final* :: *'state* ⇒ *bool*
\+
fixes *S* :: *'state list* ⇒ *'secret list*
and *A* :: *'state trace* ⇒ *'act list*
and *O* :: *'state trace* ⇒ *'obs list*
begin

fun *leakVia* :: *'state list* ⇒ *'state list* ⇒ *'secret list* × *'secret list* ⇒ *bool*
where
$leakVia\ tr\ tr'\ (sl,sl') = (S\ tr = sl \wedge S\ tr' = sl' \wedge A\ tr = A\ tr' \wedge O\ tr \neq O\ tr')$

lemmas *leakVia-def* = *leakVia.simps*

end

sublocale *Attacker-Mod-fin* < *Leakage-Mod-fin*

where *leakVia* = *leakVia*
by *standard*

1.2 Locales for increasingly concrete notions of finitary relative security

locale *Relative-Security''-fin* =
Van: *Leakage-Mod-fin istateV validTransV finalV SV AV OV leakViaV*
+
Opt: *Leakage-Mod-fin istateO validTransO finalO SO AO OO leakViaO*
for *validTransV* :: *'stateV* × *'stateV* ⇒ *bool*
and *istateV* :: *'stateV* ⇒ *bool* **and** *finalV* :: *'stateV* ⇒ *bool*
and *SV* :: *'stateV list* ⇒ *'secret list*
and *AV* :: *'stateV trace* ⇒ *'actV list*
and *OV* :: *'stateV trace* ⇒ *'obsV list*
and *leakViaV* :: *'stateV list* ⇒ *'stateV list* ⇒ *'leak* ⇒ *bool*

and *validTransO* :: *'stateO* × *'stateO* ⇒ *bool*
and *istateO* :: *'stateO* ⇒ *bool* **and** *finalO* :: *'stateO* ⇒ *bool*
and *SO* :: *'stateO list* ⇒ *'secret list*
and *AO* :: *'stateO trace* ⇒ *'actO list*
and *OO* :: *'stateO trace* ⇒ *'obsO list*
and *leakViaO* :: *'stateO list* ⇒ *'stateO list* ⇒ *'leak* ⇒ *bool*

and *corrState* :: *'stateV* ⇒ *'stateO* ⇒ *bool*
begin

definition *rsecure* :: *bool* **where**
rsecure ≡ ∀ *l s1 tr1 s2 tr2*.
istateO s1 ∧ *Opt.validFromS s1 tr1* ∧ *Opt.completedFrom s1 tr1* ∧
istateO s2 ∧ *Opt.validFromS s2 tr2* ∧ *Opt.completedFrom s2 tr2* ∧
leakViaO tr1 tr2 l
⟶
(∃ *sv1 trv1 sv2 trv2*.
istateV sv1 ∧ *istateV sv2* ∧ *corrState sv1 s1* ∧ *corrState sv2 s2* ∧
Van.validFromS sv1 trv1 ∧ *Van.completedFrom sv1 trv1* ∧
Van.validFromS sv2 trv2 ∧ *Van.completedFrom sv2 trv2* ∧
leakViaV trv1 trv2 l)

end

locale *Relative-Security'-fin* =
Van: *Attacker-Mod-fin istateV validTransV finalV SV AV OV*
+
Opt: *Attacker-Mod-fin istateO validTransO finalO SO AO OO*
for *validTransV* :: *'stateV* × *'stateV* ⇒ *bool*
and *istateV* :: *'stateV* ⇒ *bool* **and** *finalV* :: *'stateV* ⇒ *bool*
and *SV* :: *'stateV list* ⇒ *'secret list*

and AV :: $'stateV$ $trace \Rightarrow$ $'actV$ $list$
and OV :: $'stateV$ $trace \Rightarrow$ $'obsV$ $list$

and $validTransO$ :: $'stateO \times 'stateO \Rightarrow bool$
and $istateO$ :: $'stateO \Rightarrow bool$ **and** $finalO$ :: $'stateO \Rightarrow bool$
and SO :: $'stateO$ $list \Rightarrow$ $'secret$ $list$
and AO :: $'stateO$ $trace \Rightarrow$ $'actO$ $list$
and OO :: $'stateO$ $trace \Rightarrow$ $'obsO$ $list$
and $corrState$ :: $'stateV \Rightarrow 'stateO \Rightarrow bool$

sublocale *Relative-Security'-fin* < *Relative-Security''-fin*
where *leakViaV* = *Van.leakVia* **and** *leakViaO* = *Opt.leakVia*
by *standard*

context *Relative-Security'-fin*
begin

lemma *rsecure-def2*:
rsecure ⟷
(∀ *s1 tr1 s2 tr2*.
istateO s1 ∧ *Opt.validFromS s1 tr1* ∧ *Opt.completedFrom s1 tr1* ∧
istateO s2 ∧ *Opt.validFromS s2 tr2* ∧ *Opt.completedFrom s2 tr2* ∧
AO tr1 = *AO tr2* ∧ *OO tr1* ≠ *OO tr2*
⟶
(∃ *sv1 trv1 sv2 trv2*.
istateV sv1 ∧ *istateV sv2* ∧ *corrState sv1 s1* ∧ *corrState sv2 s2* ∧
Van.validFromS sv1 trv1 ∧ *Van.completedFrom sv1 trv1* ∧
Van.validFromS sv2 trv2 ∧ *Van.completedFrom sv2 trv2* ∧
SV trv1 = *SO tr1* ∧ *SV trv2* = *SO tr2* ∧
AV trv1 = *AV trv2* ∧ *OV trv1* ≠ *OV trv2*))
unfolding *rsecure-def*
unfolding *Van.leakVia-def Opt.leakVia-def*
by *auto metis*

end

locale *Statewise-Attacker-Mod* = *System-Mod istate validTrans final*
for *istate* :: $'state \Rightarrow bool$ **and** *validTrans* :: $'state \times 'state \Rightarrow bool$ **and** *final* :: $'state \Rightarrow bool$
+
fixes
isSec :: $'state \Rightarrow bool$ **and** *getSec* :: $'state \Rightarrow 'secret$
and
isInt :: $'state \Rightarrow bool$ **and** *getInt* :: $'state \Rightarrow 'act \times 'obs$
assumes *final-not-isInt*: ⋀*s. final s* ⟹ ¬ *isInt s*

and *final-not-isSec*: $\bigwedge s.\ final\ s \Longrightarrow \neg\ isSec\ s$
begin

definition $getAct :: 'state \Rightarrow 'act$ **where**
$getAct = fst\ o\ getInt$

definition $getObs :: 'state \Rightarrow 'obs$ **where**
$getObs = snd\ o\ getInt$

definition $eqObs\ trn1\ trn2 \equiv$
$(isInt\ trn1 \longleftrightarrow isInt\ trn2) \wedge (isInt\ trn1 \longrightarrow getObs\ trn1 = getObs\ trn2)$

definition $eqAct\ trn1\ trn2 \equiv$
$(isInt\ trn1 \longleftrightarrow isInt\ trn2) \wedge (isInt\ trn1 \longrightarrow getAct\ trn1 = getAct\ trn2)$

definition $A :: 'state\ trace \Rightarrow 'act\ list$ **where**
$A\ tr \equiv filtermap\ isInt\ getAct\ (butlast\ tr)$

sublocale *A*: *FiltermapBL isInt getAct A*
apply *standard* **unfolding** *A-def* **..**

definition $O :: 'state\ trace \Rightarrow 'obs\ list$ **where**
$O\ tr \equiv filtermap\ isInt\ getObs\ (butlast\ tr)$

sublocale *O*: *FiltermapBL isInt getObs O*
apply *standard* **unfolding** *O-def* **..**

definition $S :: 'state\ list \Rightarrow 'secret\ list$ **where**
$S\ tr \equiv filtermap\ isSec\ getSec\ (butlast\ tr)$

sublocale *S*: *FiltermapBL isSec getSec S*
apply *standard* **unfolding** *S-def* **..**

end

sublocale *Statewise-Attacker-Mod* < *Attacker-Mod-fin*
where $S = S$ **and** $A = A$ **and** $O = O$
by *standard*

locale *Rel-Sec* =

Van: *Statewise-Attacker-Mod istateV validTransV finalV isSecV getSecV isIntV getIntV*
+
Opt: *Statewise-Attacker-Mod istateO validTransO finalO isSecO getSecO isIntO getIntO*
for *validTransV* :: *'stateV* × *'stateV* ⇒ *bool*
and *istateV* :: *'stateV* ⇒ *bool* **and** *finalV* :: *'stateV* ⇒ *bool*
and *isSecV* :: *'stateV* ⇒ *bool* **and** *getSecV* :: *'stateV* ⇒ *'secret*
and *isIntV* :: *'stateV* ⇒ *bool* **and** *getIntV* :: *'stateV* ⇒ *'actV* × *'obsV*

and *validTransO* :: *'stateO* × *'stateO* ⇒ *bool*
and *istateO* :: *'stateO* ⇒ *bool* **and** *finalO* :: *'stateO* ⇒ *bool*
and *isSecO* :: *'stateO* ⇒ *bool* **and** *getSecO* :: *'stateO* ⇒ *'secret*
and *isIntO* :: *'stateO* ⇒ *bool* **and** *getIntO* :: *'stateO* ⇒ *'actO* × *'obsO*

and *corrState* :: *'stateV* ⇒ *'stateO* ⇒ *bool*

sublocale *Rel-Sec* < *Relative-Security'-fin*
where *SV* = *Van.S* **and** *AV* = *Van.A* **and** *OV* = *Van.O*
and *SO* = *Opt.S* **and** *AO* = *Opt.A* **and** *OO* = *Opt.O*
by *standard*

context *Rel-Sec*
begin

abbreviation *getObsV* :: *'stateV* ⇒ *'obsV* **where** *getObsV* ≡ *Van.getObs*
abbreviation *getActV* :: *'stateV* ⇒ *'actV* **where** *getActV* ≡ *Van.getAct*
abbreviation *getObsO* :: *'stateO* ⇒ *'obsO* **where** *getObsO* ≡ *Opt.getObs*
abbreviation *getActO* :: *'stateO* ⇒ *'actO* **where** *getActO* ≡ *Opt.getAct*

abbreviation *reachV* **where** *reachV* ≡ *Van.reach*
abbreviation *reachO* **where** *reachO* ≡ *Opt.reach*

abbreviation *completedFromV* :: *'stateV* ⇒ *'stateV list* ⇒ *bool* **where** *completedFromV* ≡ *Van.completedFrom*
abbreviation *completedFromO* :: *'stateO* ⇒ *'stateO list* ⇒ *bool* **where** *completedFromO* ≡ *Opt.completedFrom*

lemmas *completedFromV-def* = *Van.completedFrom-def*
lemmas *completedFromO-def* = *Opt.completedFrom-def*

lemma *rsecure-def3*:
rsecure ⟷
(∀ *s1 tr1 s2 tr2*.
istateO s1 ∧ *Opt.validFromS s1 tr1* ∧ *completedFromO s1 tr1* ∧
istateO s2 ∧ *Opt.validFromS s2 tr2* ∧ *completedFromO s2 tr2* ∧
Opt.A tr1 = *Opt.A tr2* ∧ *Opt.O tr1* ≠ *Opt.O tr2* ∧

$(isIntO\ s1\ \wedge\ isIntO\ s2\ \longrightarrow\ getActO\ s1\ =\ getActO\ s2)$
$\longrightarrow$
$(\exists\, sv1\ trv1\ sv2\ trv2.$
$istateV\ sv1\ \wedge\ istateV\ sv2\ \wedge\ corrState\ sv1\ s1\ \wedge\ corrState\ sv2\ s2\ \wedge$
$Van.validFromS\ sv1\ trv1\ \wedge\ completedFromV\ sv1\ trv1\ \wedge$
$Van.validFromS\ sv2\ trv2\ \wedge\ completedFromV\ sv2\ trv2\ \wedge$
$Van.S\ trv1\ =\ Opt.S\ tr1\ \wedge\ Van.S\ trv2\ =\ Opt.S\ tr2\ \wedge$
$Van.A\ trv1\ =\ Van.A\ trv2\ \wedge\ Van.O\ trv1\ \neq\ Van.O\ trv2))$
unfolding *rsecure-def2* **apply** (*intro iff-allI iffI impI*)
subgoal by *auto*
subgoal
by *clarsimp* (*metis* (*full-types*) *Opt.A.Cons-unfold*
Opt.completed-Cons Opt.final-not-isInt
Simple-Transition-System.validFromS-Cons-iff
completedFromO-def list.sel(*1*) *neq-Nil-conv*) **.**

definition *eqSec trnO trnA* ≡
$(isSecV\ trnO = isSecO\ trnA) \wedge (isSecV\ trnO \longrightarrow getSecV\ trnO = getSecO\ trnA)$

lemma *eqSec-S-Cons′*:
$eqSec\ trnO\ trnA \Longrightarrow$
$(Van.S\ (trnO\ \#\ trO') = Opt.S\ (trnA\ \#\ trA')) \Longrightarrow Van.S\ trO' = Opt.S\ trA'$
apply(*cases* $trO' = []$)
subgoal apply(*cases* $trA' = []$)
subgoal by *auto*
subgoal unfolding *eqSec-def* **by** *auto* **.**
subgoal apply(*cases* $trA' = []$)
subgoal by *auto*
subgoal unfolding *eqSec-def* **by** *auto* **. .**

lemma *eqSec-S-Cons*[*simp*]:
$eqSec\ trnO\ trnA \Longrightarrow trO' = [] \longleftrightarrow trA' = [] \Longrightarrow$
$(Van.S\ (trnO\ \#\ trO') = Opt.S\ (trnA\ \#\ trA')) \longleftrightarrow (Van.S\ trO' = Opt.S\ trA')$
apply(*cases* $trO' = []$)
subgoal apply(*cases* $trA' = []$)
subgoal by *auto*
subgoal unfolding *eqSec-def* **by** *auto* **.**
subgoal apply(*cases* $trA' = []$)
subgoal by *auto*
subgoal unfolding *eqSec-def* **by** *auto* **. .**

end

locale *Relative-Security-Determ* =
Rel-Sec
validTransV istateV finalV isSecV getSecV isIntV getIntV
validTransO istateO finalO isSecO getSecO isIntO getIntO
corrState
+
System-Mod-Deterministic istateV validTransV finalV nextO
for *validTransV* :: *'stateV* × *'stateV* ⇒ *bool*
and *istateV* :: *'stateV* ⇒ *bool*
and *finalV* :: *'stateV* ⇒ *bool*
and *nextO* :: *'stateV* ⇒ *'stateV*
and *isSecV* :: *'stateV* ⇒ *bool* **and** *getSecV* :: *'stateV* ⇒ *'secret*
and *isIntV* :: *'stateV* ⇒ *bool* **and** *getIntV* :: *'stateV* ⇒ *'actV* × *'obsV*
and *validTransO* :: *'stateO* × *'stateO* ⇒ *bool*
and *istateO* :: *'stateO* ⇒ *bool*
and *finalO* :: *'stateO* ⇒ *bool*
and *isSecO* :: *'stateO* ⇒ *bool* **and** *getSecO* :: *'stateO* ⇒ *'secret*
and *isIntO* :: *'stateO* ⇒ *bool* **and** *getIntO* :: *'stateO* ⇒ *'actO* × *'obsO*
and *corrState* :: *'stateV* ⇒ *'stateO* ⇒ *bool*

end

2 Relative Security

This theory formalizes the general notion of relative security, applicable to possibly infinite traces.

theory *Relative-Security*
imports *Relative-Security-fin Preliminaries/Trivia*
begin

no-notation *relcomp* (**infixr** *O 75*)
no-notation *relcompp* (**infixr** *OO 75*)

2.1 Leakage models and attacker models

locale *Leakage-Mod* = *System-Mod istate validTrans final*
for *istate* :: *'state* ⇒ *bool* **and** *validTrans* :: *'state* × *'state* ⇒ *bool* **and** *final* :: *'state* ⇒ *bool*
+
fixes *S* :: *'state llist* ⇒ *'secret llist*
and *A* :: *'state ltrace* ⇒ *'act llist*
and *O* :: *'state ltrace* ⇒ *'obs llist*
and *lleakVia* :: *'state llist* ⇒ *'state llist* ⇒ *'leak* ⇒ *bool*

locale *Attacker-Mod* = *System-Mod istate validTrans final*

for *istate* :: *'state* ⇒ *bool* **and** *validTrans* :: *'state* × *'state* ⇒ *bool* **and** *final* :: *'state* ⇒ *bool*
+
fixes *S* :: *'state llist* ⇒ *'secret llist*
and *A* :: *'state ltrace* ⇒ *'act llist*
and *O* :: *'state ltrace* ⇒ *'obs llist*
begin

fun *lleakVia* :: *'state llist* ⇒ *'state llist* ⇒ *'secret llist* × *'secret llist* ⇒ *bool*
where
lleakVia tr tr' (*sl*,*sl'*) = (*S tr* = *sl* ∧ *S tr'* = *sl'* ∧ *A tr* = *A tr'* ∧ *O tr* ≠ *O tr'*)

lemmas *lleakVia-def* = *lleakVia.simps*

end

sublocale *Attacker-Mod* < *Leakage-Mod*
where *lleakVia* = *lleakVia*
by *standard*

2.2 Locales for increasingly concrete notions of relative security

locale *Relative-Security''* =
  Van: *Leakage-Mod istateV validTransV finalV SV AV OV lleakViaV*
+
  Opt: *Leakage-Mod istateO validTransO finalO SO AO OO lleakViaO*
  for *validTransV* :: *'stateV* × *'stateV* ⇒ *bool*
  and *istateV* :: *'stateV* ⇒ *bool* **and** *finalV* :: *'stateV* ⇒ *bool*
  and *SV* :: *'stateV llist* ⇒ *'secret llist*
  and *AV* :: *'stateV ltrace* ⇒ *'actV llist*
  and *OV* :: *'stateV ltrace* ⇒ *'obsV llist*
  and *lleakViaV* :: *'stateV llist* ⇒ *'stateV llist* ⇒ *'leak* ⇒ *bool*

  and *validTransO* :: *'stateO* × *'stateO* ⇒ *bool*
  and *istateO* :: *'stateO* ⇒ *bool* **and** *finalO* :: *'stateO* ⇒ *bool*
  and *SO* :: *'stateO llist* ⇒ *'secret llist*
  and *AO* :: *'stateO ltrace* ⇒ *'actO llist*
  and *OO* :: *'stateO ltrace* ⇒ *'obsO llist*
  and *lleakViaO* :: *'stateO llist* ⇒ *'stateO llist* ⇒ *'leak* ⇒ *bool*

  and *corrState* :: *'stateV* ⇒ *'stateO* ⇒ *bool*
begin

definition *lrsecure* :: *bool* **where**
lrsecure ≡ ∀ *l s1 tr1 s2 tr2*.
   istateO s1 ∧ *Opt.lvalidFromS s1 tr1* ∧ *Opt.lcompletedFrom s1 tr1* ∧
   istateO s2 ∧ *Opt.lvalidFromS s2 tr2* ∧ *Opt.lcompletedFrom s2 tr2* ∧

lleakViaO tr1 tr2 l
$\longrightarrow$
$(\exists$ *sv1 trv1 sv2 trv2*.
istateV sv1 $\wedge$ *istateV sv2* $\wedge$ *corrState sv1 s1* $\wedge$ *corrState sv2 s2* $\wedge$
Van.lvalidFromS sv1 trv1 $\wedge$ *Van.lcompletedFrom sv1 trv1* $\wedge$
Van.lvalidFromS sv2 trv2 $\wedge$ *Van.lcompletedFrom sv2 trv2* $\wedge$
lleakViaV trv1 trv2 l$)$

end

locale *Relative-Security'* =
Van: *Attacker-Mod istateV validTransV finalV SV AV OV*
+
Opt: *Attacker-Mod istateO validTransO finalO SO AO OO*
for *validTransV* :: *'stateV* $\times$ *'stateV* $\Rightarrow$ *bool*
and *istateV* :: *'stateV* $\Rightarrow$ *bool* **and** *finalV* :: *'stateV* $\Rightarrow$ *bool*
and *SV* :: *'stateV llist* $\Rightarrow$ *'secret llist*
and *AV* :: *'stateV ltrace* $\Rightarrow$ *'actV llist*
and *OV* :: *'stateV ltrace* $\Rightarrow$ *'obsV llist*

and *validTransO* :: *'stateO* $\times$ *'stateO* $\Rightarrow$ *bool*
and *istateO* :: *'stateO* $\Rightarrow$ *bool* **and** *finalO* :: *'stateO* $\Rightarrow$ *bool*
and *SO* :: *'stateO llist* $\Rightarrow$ *'secret llist*
and *AO* :: *'stateO ltrace* $\Rightarrow$ *'actO llist*
and *OO* :: *'stateO ltrace* $\Rightarrow$ *'obsO llist*
and *corrState* :: *'stateV* $\Rightarrow$ *'stateO* $\Rightarrow$ *bool*

sublocale *Relative-Security'* < *Relative-Security''*
where *lleakViaV* = *Van.lleakVia* **and** *lleakViaO* = *Opt.lleakVia*
by *standard*

context *Relative-Security'*
begin

lemma *lrsecure-def2*:
lrsecure $\longleftrightarrow$
$(\forall$ *s1 tr1 s2 tr2*.
istateO s1 $\wedge$ *Opt.lvalidFromS s1 tr1* $\wedge$ *Opt.lcompletedFrom s1 tr1* $\wedge$
istateO s2 $\wedge$ *Opt.lvalidFromS s2 tr2* $\wedge$ *Opt.lcompletedFrom s2 tr2* $\wedge$
AO tr1 = *AO tr2* $\wedge$ *OO tr1* $\neq$ *OO tr2*
$\longrightarrow$
$(\exists$ *sv1 trv1 sv2 trv2*.
istateV sv1 $\wedge$ *istateV sv2* $\wedge$ *corrState sv1 s1* $\wedge$ *corrState sv2 s2* $\wedge$

Van.lvalidFromS sv1 trv1 ∧ *Van.lcompletedFrom sv1 trv1* ∧
Van.lvalidFromS sv2 trv2 ∧ *Van.lcompletedFrom sv2 trv2* ∧
SV trv1 = *SO tr1* ∧ *SV trv2* = *SO tr2* ∧
AV trv1 = *AV trv2* ∧ *OV trv1* ≠ *OV trv2*))
unfolding *lrsecure-def*
unfolding *Van.lleakVia-def Opt.lleakVia-def*
by *auto metis*

end

context *Statewise-Attacker-Mod* **begin**

definition *lA* :: *'state ltrace* ⇒ *'act llist* **where**
lA tr ≡ *lfiltermap isInt getAct* (*lbutlast tr*)

sublocale *lA*: *LfiltermapBL isInt getAct lA*
apply *standard* **unfolding** *lA-def* **..**

lemma *lA*: *lcompletedFrom s tr* ⟹ *lA tr* = *lmap getAct* (*lfilter isInt tr*)
apply(*cases lfinite tr*)
subgoal unfolding *lA.lmap-lfilter lbutlast-def*
by *simp* (*metis final-not-isInt lbutlast-lfinite lcompletedFrom-def lfilter-llist-of lfiltermap-lmap-lfilter lfinite-lfiltermap-butlast llast-llist-of llist-of-list-of lmap-llist-of*)
subgoal unfolding *lA.lmap-lfilter lbutlast-def* **by** *auto* **.**

definition *lO* :: *'state ltrace* ⇒ *'obs llist* **where**
lO tr ≡ *lfiltermap isInt getObs* (*lbutlast tr*)

sublocale *lO*: *LfiltermapBL isInt getObs lO*
apply *standard* **unfolding** *lO-def* **..**

lemma *lO*: *lcompletedFrom s tr* ⟹ *lO tr* = *lmap getObs* (*lfilter isInt tr*)
apply(*cases lfinite tr*)
subgoal unfolding *lO.lmap-lfilter lbutlast-def*
by *simp* (*metis List-Filtermap.filtermap-def butlast.simps*(*1*) *filtermap-butlast final-not-isInt lcompletedFrom-def lfilter-llist-of llist-of-list-of lmap-llist-of*)
subgoal unfolding *lO.lmap-lfilter lbutlast-def* **by** *auto* **.**

definition *lS* :: *'state llist* ⇒ *'secret llist* **where**
lS tr ≡ *lfiltermap isSec getSec* (*lbutlast tr*)

sublocale *lS*: *LfiltermapBL isSec getSec lS*
  apply *standard* **unfolding** *lS-def* **..**

lemma *lS*: *lcompletedFrom s tr* $\Longrightarrow$ *lS tr* = *lmap getSec* (*lfilter isSec tr*)
apply(*cases lfinite tr*)
  subgoal unfolding *lS.lmap-lfilter lbutlast-def*
  by *simp* (*metis List-Filtermap.filtermap-def filtermap-butlast final-not-isSec lcompletedFrom-def lfilter-llist-of llist-of-eq-LNil-conv llist-of-list-of lmap-llist-of*)
  subgoal unfolding *lS.lmap-lfilter lbutlast-def* **by** *auto* **.**

end

sublocale *Statewise-Attacker-Mod* < *Attacker-Mod*
where *S* = *lS* **and** *A* = *lA* **and** *O* = *lO*
by *standard*

sublocale *Rel-Sec* < *Relative-Security'*
where *SV* = *Van.lS* **and** *AV* = *Van.lA* **and** *OV* = *Van.lO*
and *SO* = *Opt.lS* **and** *AO* = *Opt.lA* **and** *OO* = *Opt.lO*
by *standard*

context *Rel-Sec*
begin

abbreviation *lcompletedFromV* :: *'stateV* ⇒ *'stateV llist* ⇒ *bool* **where** *lcompletedFromV* ≡ *Van.lcompletedFrom*
abbreviation *lcompletedFromO* :: *'stateO* ⇒ *'stateO llist* ⇒ *bool* **where** *lcompletedFromO* ≡ *Opt.lcompletedFrom*

lemma *eqSec-lS-Cons'*:
eqSec trnO trnA $\Longrightarrow$
(*Van.lS* (*trnO* $ *trO'*) = *Opt.lS* (*trnA* $ *trA'*)) $\Longrightarrow$ *Van.lS trO'* = *Opt.lS trA'*
apply(*cases trO'* = [[]])
  subgoal apply(*cases trA'* = [[]])
    subgoal by *auto*
    subgoal unfolding *eqSec-def* **by** *auto* **.**
  subgoal apply(*cases trA'* = [[]])
    subgoal by *auto*
    subgoal unfolding *eqSec-def* **by** *auto* **. .**

lemma *eqSec-lS-Cons*[*simp*]:
eqSec trnO trnA $\Longrightarrow$ *trO'* = [[]] $\longleftrightarrow$ *trA'* = [[]] $\Longrightarrow$

$(Van.lS\ (trnO\ \$\ trO') = Opt.lS\ (trnA\ \$\ trA')) \longleftrightarrow (Van.lS\ trO' = Opt.lS\ trA')$
apply(*cases trO'* = [[]])
  subgoal apply(*cases trA'* = [[]])
    subgoal by *auto*
    subgoal unfolding *eqSec-def* **by** *auto* **.**
  subgoal apply(*cases trA'* = [[]])
    subgoal by *auto*
    subgoal unfolding *eqSec-def* **by** *auto* **. .**

end

end

3 Unwinding Proof Method for Finitary Relative Security

This theory formalizes the notion of unwinding for finitary relative security, and proves its soundness.

theory *Unwinding-fin*
imports *Relative-Security*
begin

3.1 The types and operators underlying unwinding: status, matching operators, etc.

context *Rel-Sec*
begin

datatype *status* = *Eq* | *Diff*

fun $updStat :: status \Rightarrow bool \times 'a \Rightarrow bool \times 'a \Rightarrow status$ **where**
$updStat\ Eq\ (True,a)\ (True,a') = (if\ a = a'\ then\ Eq\ else\ Diff)$
$|updStat\ stat\ \text{-}\ \text{-} = stat$

definition $sstatO'\ statO\ sv1\ sv2 = updStat\ statO\ (isIntV\ sv1,\ getObsV\ sv1)\ (isIntV\ sv2,\ getObsV\ sv2)$
definition $sstatA'\ statA\ s1\ s2 = updStat\ statA\ (isIntO\ s1,\ getObsO\ s1)\ (isIntO\ s2,\ getObsO\ s2)$

definition *initCond* ::
$(enat \Rightarrow 'stateO \Rightarrow 'stateO \Rightarrow status \Rightarrow 'stateV \Rightarrow 'stateV \Rightarrow status \Rightarrow bool) \Rightarrow bool$ **where**
$initCond\ \Delta \equiv \forall s1\ s2.$
  $istateO\ s1 \wedge istateO\ s2$
  $\longrightarrow$
  $(\exists sv1\ sv2.\ istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2$

$\wedge\ \Delta\ \infty\ s1\ s2\ Eq\ sv1\ sv2\ Eq)$

definition *match1-1* Δ *s1 s1' s2 statA sv1 sv2 statO* $\equiv$
$\exists$ *sv1'. validTransV* (*sv1*,*sv1'*) $\wedge$
$\Delta\ \infty$ *s1' s2 statA sv1' sv2 statO*

definition *match1-12* Δ *s1 s1' s2 statA sv1 sv2 statO* $\equiv$
$\exists$ *sv1' sv2'.*
let statO' = sstatO' statO sv1 sv2 in
validTransV (*sv1*,*sv1'*) $\wedge$
validTransV (*sv2*,*sv2'*) $\wedge$
$\Delta\ \infty$ *s1' s2 statA sv1' sv2' statO'*

definition *match1* Δ *s1 s2 statA sv1 sv2 statO* $\equiv$
$\neg$ *isIntO s1* $\longrightarrow$
($\forall$ *s1'. validTransO* (*s1*,*s1'*)
$\longrightarrow$
($\neg$ *isSecO s1* $\wedge$ $\Delta\ \infty$ *s1' s2 statA sv1 sv2 statO*) $\vee$
(*eqSec sv1 s1* $\wedge$ $\neg$ *isIntV sv1* $\wedge$ *match1-1* Δ *s1 s1' s2 statA sv1 sv2 statO*) $\vee$
(*eqSec sv1 s1* $\wedge$ $\neg$ *isSecV sv2* $\wedge$ *Van.eqAct sv1 sv2* $\wedge$ *match1-12* Δ *s1 s1' s2 statA sv1 sv2 statO*))

lemmas *match1-defs* = *match1-def match1-1-def match1-12-def*

lemma *match1-1-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match1-1* Δ *s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$ *match1-1* Δ' *s1 s1' s2 statA sv1 sv2 statO*
unfolding *le-fun-def match1-1-def* **by** *auto*

lemma *match1-12-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match1-12* Δ *s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$ *match1-12* Δ' *s1 s1' s2 statA sv1 sv2 statO*
unfolding *le-fun-def match1-12-def* **by** *fastforce*

lemma *match1-mono*:
assumes $\Delta \leq \Delta'$
shows *match1* Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match1* Δ' *s1 s2 statA sv1 sv2 statO*
unfolding *match1-def* **apply** *clarify* **subgoal for** *s1'* **apply**(*erule allE*[*of - s1'*])
using *match1-1-mono*[*OF assms*, *of s1 s1' s2 statA sv1 sv2 statO*]
match1-12-mono[*OF assms*, *of s1 s1' s2 statA sv1 sv2 statO*]
assms[*unfolded le-fun-def*, *rule-format*, *of - s1' s2 statA sv1 sv2 statO*]
by *auto* **.**

definition *match2-1* Δ *s1 s2 s2′ statA sv1 sv2 statO* ≡
∃ *sv2′*. *validTransV* (*sv2*,*sv2′*) ∧
Δ ∞ *s1 s2′ statA sv1 sv2′ statO*

definition *match2-12* Δ *s1 s2 s2′ statA sv1 sv2 statO* ≡
∃ *sv1′ sv2′*.
let statO′ = *sstatO′ statO sv1 sv2 in*
validTransV (*sv1*,*sv1′*) ∧
validTransV (*sv2*,*sv2′*) ∧
Δ ∞ *s1 s2′ statA sv1′ sv2′ statO′*

definition *match2* Δ *s1 s2 statA sv1 sv2 statO* ≡
¬ *isIntO s2* ⟶
(∀ *s2′*. *validTransO* (*s2*,*s2′*)
⟶
(¬ *isSecO s2* ∧ Δ ∞ *s1 s2′ statA sv1 sv2 statO*) ∨
(*eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧ *match2-1* Δ *s1 s2 s2′ statA sv1 sv2 statO*) ∨
(¬ *isSecV sv1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧ *match2-12* Δ *s1 s2 s2′ statA sv1 sv2 statO*))

lemmas *match2-defs* = *match2-def match2-1-def match2-12-def*

lemma *match2-1-mono*:
Δ ≤ Δ′ ⟹ *match2-1* Δ *s1 s1′ s2 statA sv1 sv2 statO* ⟹ *match2-1* Δ′ *s1 s1′ s2 statA sv1 sv2 statO*
unfolding *le-fun-def match2-1-def* **by** *auto*

lemma *match2-12-mono*:
Δ ≤ Δ′ ⟹ *match2-12* Δ *s1 s1′ s2 statA sv1 sv2 statO* ⟹ *match2-12* Δ′ *s1 s1′ s2 statA sv1 sv2 statO*
unfolding *le-fun-def match2-12-def* **by** *fastforce*

lemma *match2-mono*:
assumes Δ ≤ Δ′
shows *match2* Δ *s1 s2 statA sv1 sv2 statO* ⟹ *match2* Δ′ *s1 s2 statA sv1 sv2 statO*
unfolding *match2-def* **apply** *clarify* **subgoal for** *s2′* **apply**(*erule allE*[*of - s2′*])
using *match2-1-mono*[*OF assms*, *of s1 s2 s2′ statA sv1 sv2 statO*]
match2-12-mono[*OF assms*, *of s1 s2 s2′ statA sv1 sv2 statO*]
assms[*unfolded le-fun-def*, *rule-format*, *of - s1 s2′ statA sv1 sv2 statO*]
by *auto* **.**

definition *match12-1* Δ *s1′ s2′ statA′ sv1 sv2 statO* ≡
∃ *sv1′*. *validTransV* (*sv1*,*sv1′*) ∧
Δ ∞ *s1′ s2′ statA′ sv1′ sv2 statO*

definition *match12-2* Δ *s1′ s2′ statA′ sv1 sv2 statO* ≡

$\exists$ *sv2'*. *validTransV* (*sv2*,*sv2'*) $\wedge$
Δ ∞ *s1' s2' statA' sv1 sv2' statO*

definition *match12-12* Δ *s1' s2' statA' sv1 sv2 statO* $\equiv$
$\exists$ *sv1' sv2'*.
let statO' = sstatO' statO sv1 sv2 in
validTransV (*sv1*,*sv1'*) $\wedge$
validTransV (*sv2*,*sv2'*) $\wedge$
(*statA' = Diff* $\longrightarrow$ *statO' = Diff*) $\wedge$
Δ ∞ *s1' s2' statA' sv1' sv2' statO'*

definition *match12* Δ *s1 s2 statA sv1 sv2 statO* $\equiv$
$\forall$ *s1' s2'*.
let statA' = sstatA' statA s1 s2 in
validTransO (*s1*,*s1'*) $\wedge$
validTransO (*s2*,*s2'*) $\wedge$
Opt.eqAct s1 s2 $\wedge$
isIntO s1 $\wedge$ *isIntO s2*
$\longrightarrow$
($\neg$ *isSecO s1* $\wedge$ $\neg$ *isSecO s2* $\wedge$ (*statA = statA'* $\vee$ *statO = Diff*) $\wedge$ Δ ∞ *s1' s2' statA' sv1 sv2 statO*)
$\vee$
($\neg$ *isSecO s2* $\wedge$ *eqSec sv1 s1* $\wedge$ $\neg$ *isIntV sv1* $\wedge$
(*statA = statA'* $\vee$ *statO = Diff*) $\wedge$
match12-1 Δ *s1' s2' statA' sv1 sv2 statO*)
$\vee$
($\neg$ *isSecO s1* $\wedge$ *eqSec sv2 s2* $\wedge$ $\neg$ *isIntV sv2* $\wedge$
(*statA = statA'* $\vee$ *statO = Diff*) $\wedge$
match12-2 Δ *s1' s2' statA' sv1 sv2 statO*)
$\vee$
(*eqSec sv1 s1* $\wedge$ *eqSec sv2 s2* $\wedge$ *Van.eqAct sv1 sv2* $\wedge$
match12-12 Δ *s1' s2' statA' sv1 sv2 statO*)

lemmas *match12-defs = match12-def match12-1-def match12-2-def match12-12-def*

lemma *match12-simpleI*:
assumes $\bigwedge$*s1' s2' statA'*.
statA' = sstatA' statA s1 s2 $\Longrightarrow$
validTransO (*s1*,*s1'*) $\Longrightarrow$
validTransO (*s2*,*s2'*) $\Longrightarrow$
Opt.eqAct s1 s2 $\Longrightarrow$
($\neg$ *isSecO s1* $\wedge$ $\neg$ *isSecO s2* $\wedge$ (*statA = statA'* $\vee$ *statO = Diff*) $\wedge$ Δ ∞ *s1' s2' statA' sv1 sv2 statO*)
$\vee$
(*eqSec sv1 s1* $\wedge$ *eqSec sv2 s2* $\wedge$ *Van.eqAct sv1 sv2* $\wedge$
match12-12 Δ *s1' s2' statA' sv1 sv2 statO*)
shows *match12* Δ *s1 s2 statA sv1 sv2 statO*
using *assms* **unfolding** *match12-def Let-def* **by** *blast*

lemma *match12-1-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match12-1* Δ *s1′ s2′ statA′ sv1 sv2 statO* $\Longrightarrow$ *match12-1* Δ' *s1′ s2′ statA′ sv1 sv2 statO*
unfolding *le-fun-def match12-1-def* **by** *auto*

lemma *match12-2-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match12-2* Δ *s1 s2′ statA′ sv1 sv2 statO* $\Longrightarrow$ *match12-2* Δ' *s1 s2′ statA′ sv1 sv2 statO*
unfolding *le-fun-def match12-2-def* **by** *auto*

lemma *match12-12-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match12-12* Δ *s1′ s2′ statA′ sv1 sv2 statO* $\Longrightarrow$ *match12-12* Δ' *s1′ s2′ statA′ sv1 sv2 statO*
unfolding *le-fun-def match12-12-def* **by** *fastforce*

lemma *match12-mono*:
assumes $\Delta \leq \Delta'$
shows *match12* Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match12* Δ' *s1 s2 statA sv1 sv2 statO*
unfolding *match12-def* **apply** *clarify* **subgoal for** *s1′ s2′* **apply**(*erule allE*[*of - s1′*]) **apply**(*erule allE*[*of - s2′*])
using *match12-1-mono*[*OF assms*, *of s1′ s2′ - sv1 sv2 statO*]
match12-2-mono[*OF assms*, *of s1′ s2′ - sv1 sv2 statO*]
match12-12-mono[*OF assms*, *of s1′ s2′ - sv1 sv2 statO*]
assms[*unfolded le-fun-def*, *rule-format*, *of - s1′ s2′ sstatA′ statA s1 s2 sv1 sv2 statO*]
by *simp metis* **.**

definition *match* Δ *s1 s2 statA sv1 sv2 statO* $\equiv$
match1 Δ *s1 s2 statA sv1 sv2 statO*
$\wedge$
match2 Δ *s1 s2 statA sv1 sv2 statO*
$\wedge$
match12 Δ *s1 s2 statA sv1 sv2 statO*

lemmas *match-defs* = *match1-def match2-def match12-def*
lemmas *match-deep-defs* = *match1-defs match2-defs match12-defs*

lemma *match-mono*:
assumes $\Delta \leq \Delta'$
shows *match* Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match* Δ' *s1 s2 statA sv1 sv2 statO*
unfolding *match-def* **using** *match1-mono*[*OF assms*] *match2-mono*[*OF assms*] *match12-mono*[*OF assms*] **by** *auto*

definition *move-1* Δ *w s1 s2 statA sv1 sv2 statO* $\equiv$
$\exists$ *sv1'*. *validTransV* (*sv1*,*sv1'*) $\wedge$
Δ *w s1 s2 statA sv1' sv2 statO*

definition *move-2* Δ *w s1 s2 statA sv1 sv2 statO* $\equiv$
$\exists$ *sv2'*. *validTransV* (*sv2*,*sv2'*) $\wedge$
Δ *w s1 s2 statA sv1 sv2' statO*

definition *move-12* Δ *w s1 s2 statA sv1 sv2 statO* $\equiv$
$\exists$ *sv1' sv2'*.
let statO' = *sstatO' statO sv1 sv2 in*
validTransV (*sv1*,*sv1'*) $\wedge$ *validTransV* (*sv2*,*sv2'*) $\wedge$
Δ *w s1 s2 statA sv1' sv2' statO'*

definition *proact* Δ *w s1 s2 statA sv1 sv2 statO* $\equiv$
($\neg$ *isSecV sv1* $\wedge$ $\neg$ *isIntV sv1* $\wedge$ *move-1* Δ *w s1 s2 statA sv1 sv2 statO*)
$\vee$
($\neg$ *isSecV sv2* $\wedge$ $\neg$ *isIntV sv2* $\wedge$ *move-2* Δ *w s1 s2 statA sv1 sv2 statO*)
$\vee$
($\neg$ *isSecV sv1* $\wedge$ $\neg$ *isSecV sv2* $\wedge$ *Van.eqAct sv1 sv2* $\wedge$ *move-12* Δ *w s1 s2 statA sv1 sv2 statO*)

lemmas *proact-defs* = *proact-def move-1-def move-2-def move-12-def*

lemma *move-1-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-1* Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-1* Δ' *meas s1 s2 statA sv1 sv2 statO*
unfolding *le-fun-def move-1-def* **by** *auto*

lemma *move-2-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-2* Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-2* Δ' *meas s1 s2 statA sv1 sv2 statO*
unfolding *le-fun-def move-2-def* **by** *auto*

lemma *move-12-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-12* Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-12* Δ' *meas s1 s2 statA sv1 sv2 statO*
unfolding *le-fun-def move-12-def* **by** *fastforce*

lemma *proact-mono*:
assumes $\Delta \leq \Delta'$
shows *proact* Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *proact* Δ' *meas s1 s2 statA sv1 sv2 statO*
unfolding *proact-def* **using** *move-1-mono*[*OF assms*] *move-2-mono*[*OF assms*] *move-12-mono*[*OF assms*] **by** *auto*

3.2 The definition of unwinding

definition *unwindCond* ::

$(enat \Rightarrow 'stateO \Rightarrow 'stateO \Rightarrow status \Rightarrow 'stateV \Rightarrow 'stateV \Rightarrow status \Rightarrow bool) \Rightarrow bool$
where
unwindCond $\Delta \equiv \forall\, w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\longrightarrow$
$(finalO\ s1 \longleftrightarrow finalO\ s2) \wedge (finalV\ sv1 \longleftrightarrow finalO\ s1) \wedge (finalV\ sv2 \longleftrightarrow finalO\ s2)$
$\wedge$
$(statA = Eq \longrightarrow (isIntO\ s1 \longleftrightarrow isIntO\ s2))$
$\wedge$
$((\exists\, v < w.\ proact\ \Delta\ v\ s1\ s2\ statA\ sv1\ sv2\ statO)$
$\vee$
$match\ \Delta\ s1\ s2\ statA\ sv1\ sv2\ statO$
$)$

lemma *unwindCond-simpleI*:
assumes
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(finalO\ s1 \longleftrightarrow finalO\ s2) \wedge (finalV\ sv1 \longleftrightarrow finalO\ s1) \wedge (finalV\ sv2 \longleftrightarrow finalO\ s2)$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow statA = Eq$
$\Longrightarrow$
$isIntO\ s1 \longleftrightarrow isIntO\ s2$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$match\ \Delta\ s1\ s2\ statA\ sv1\ sv2\ statO$
shows *unwindCond* Δ
using *assms* **unfolding** *unwindCond-def* **by** *auto*

3.3 The soundness of unwinding

definition $\psi\ s1\ tr1\ s2\ tr2\ statO\ sv1\ trv1\ sv2\ trv2 \equiv$
$trv1 \neq [] \wedge trv2 \neq [] \wedge$
$Van.validFromS\ sv1\ trv1 \wedge$

Van.validFromS sv2 trv2 ∧
(*finalV* (*lastt sv1 trv1*) ⟷ *finalO* (*lastt s1 tr1*)) ∧ (*finalV* (*lastt sv2 trv2*) ⟷ *finalO* (*lastt s2 tr2*)) ∧
Van.S trv1 = *Opt.S tr1* ∧ *Van.S trv2* = *Opt.S tr2* ∧
Van.A trv1 = *Van.A trv2* ∧
(*statO* = *Eq* ∧ *Opt.O tr1* ≠ *Opt.O tr2* ⟶ *Van.O trv1* ≠ *Van.O trv2*)

lemma *ψ-completedFrom*: *completedFromO s1 tr1* ⟹ *completedFromO s2 tr2* ⟹

ψ s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2
⟹ *completedFromV sv1 trv1* ∧ *completedFromV sv2 trv2*
unfolding *ψ-def Opt.completedFrom-def Van.completedFrom-def lastt-def*
by *presburger*

lemma *completedFromO-lastt*: *completedFromO s1 tr1* ⟹ *finalO* (*lastt s1 tr1*)
unfolding *Opt.completedFrom-def lastt-def* **by** *auto*

lemma *rsecure-strong*:
assumes
⋀*s1 tr1 s2 tr2*.
istateO s1 ∧ *Opt.validFromS s1 tr1* ∧ *completedFromO s1 tr1* ∧
istateO s2 ∧ *Opt.validFromS s2 tr2* ∧ *completedFromO s2 tr2* ∧
Opt.A tr1 = *Opt.A tr2* ∧ (*isIntO s1* ∧ *isIntO s2* ⟶ *getActO s1* = *getActO s2*)
⟹
∃ *sv1 trv1 sv2 trv2*.
istateV sv1 ∧ *istateV sv2* ∧ *corrState sv1 s1* ∧ *corrState sv2 s2* ∧
ψ s1 tr1 s2 tr2 Eq sv1 trv1 sv2 trv2
shows *rsecure*
unfolding *rsecure-def3* **apply** *clarify*
subgoal for *s1 tr1 s2 tr2*
using *assms*[*of s1 tr1 s2 tr2*]
using *ψ-completedFrom ψ-def completedFromO-lastt* **by** (*metis* (*full-types*))
.

proposition *unwindCond-ex-ψ*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w s1 s2 statA sv1 sv2 statO* **and** *stat*: (*statA* = *Diff* ⟶ *statO* = *Diff*)

and *v*: *Opt.validFromS s1 tr1 Opt.completedFrom s1 tr1 Opt.validFromS s2 tr2 Opt.completedFrom s2 tr2*
and *tr14*: *Opt.A tr1* = *Opt.A tr2*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
shows ∃ *trv1 trv2*. *ψ s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2*
using *assms*(*2*−)
proof(*induction length tr1* + *length tr2 w*

arbitrary: *s1 s2 statA sv1 sv2 statO tr1 tr2 rule*: *less2-induct'*)
case (*less w tr1 tr2 s1 s2 statA sv1 sv2 statO*)
note *ok* = ‹*statA* = *Diff* ⟶ *statO* = *Diff*›
note Δ = ‹Δ *w s1 s2 statA sv1 sv2 statO*›
note *A34* = ‹*Opt.A tr1* = *Opt.A tr2*›
note *r34* = *less.prems*(*8*,*9*) **note** *r12* = *less.prems*(*10*,*11*)
note *r* = *r34 r12*
note *r3* = *r34*(*1*) **note** *r4* = *r34*(*2*) **note** *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)

have *i34*: *statA* = *Eq* ⟶ *isIntO s1* = *isIntO s2*
and *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v s1 s2 statA sv1 sv2 statO*
then obtain *sv1'*
where *0*: *validTransV* (*sv1*,*sv1'*)
and Δ: Δ *v s1 s2 statA sv1' sv2 statO*
unfolding *move-1-def* **by** *auto*
have *r1'*: *reachV sv1'* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2 tr2 statO sv1' trv1 sv2 trv2*
using *less*(*2*)[*OF v*, *of tr1 tr2 s1 s2 statA sv1' sv2 statO*, *simplified*, *OF* Δ *ok - - - - - r34 r1' r2*]
using *A34 less.prems*(*3*−*6*) **by** *blast*
show *?thesis* **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - trv2*])
using ψ *ok 0 sv1* **unfolding** ψ*-def Van.completedFrom-def* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2* Δ *v s1 s2 statA sv1 sv2 statO*
then obtain *sv2'*
where *0*: *validTransV* (*sv2*,*sv2'*)
and Δ: Δ *v s1 s2 statA sv1 sv2' statO*
unfolding *move-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2 tr2 statO sv1 trv1 sv2' trv2*
using *less*(*2*)[*OF v*, *of tr1 tr2 s1 s2 statA sv1 sv2' statO*, *simplified*, *OF* Δ *ok - - - - - r34 r1 r2'*]
using *A34 less.prems*(*3*−*6*) **by** *blast*
show *?thesis* **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using ψ *ok 0 sv2* **unfolding** ψ*-def Van.completedFrom-def* **by** *auto*
next

assume *sv12*: ¬ *isSecV sv1* ¬ *isSecV sv2 Van.eqAct sv1 sv2*
and *move-12* Δ *v s1 s2 statA sv1 sv2 statO*
then obtain *sv1′ sv2′ statO′*
where *0*: *statO′* = *sstatO′ statO sv1 sv2*
validTransV (*sv1*,*sv1′*) ¬ *isSecV sv1*
validTransV (*sv2*,*sv2′*) ¬ *isSecV sv2*
Van.eqAct sv1 sv2
and Δ: Δ *v s1 s2 statA sv1′ sv2′ statO′*
unfolding *move-12-def* **by** *auto*
have *r12′*: *reachV sv1′ reachV sv2′* **using** *r1 r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
have *ok′*: *statA* = *Diff* ⟶ *statO′* = *Diff* **using** *ok 0* **unfolding** *sstatO′-def* **by** (*cases statO*, *auto*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2 tr2 statO′ sv1′ trv1 sv2′ trv2*
using *less*(*2*)[*OF v*, *of tr1 tr2 s1 s2 statA sv1′ sv2′ statO′*, *simplified*, *OF* Δ *ok′* - - - - - *r34 r12′*]
using *A34 less.prems*(*3*−*6*) **by** *blast*
show *?thesis* **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using ψ *ok′ 0 sv12* **unfolding** ψ*-def sstatO′-def Van.completedFrom-def*
using *Van.A.Cons-unfold Van.eqAct-def completedFromO-lastt less.prems*(*4*)
less.prems(*6*) **by** *auto*
qed
next
assume *m*: *match* Δ *s1 s2 statA sv1 sv2 statO*
show *?thesis*
proof(*cases length tr1* ≤ *Suc 0*)
case *True* **note** *tr1* = *True*
hence *tr1* = [] ∨ *tr1* = [*s1*]
by (*metis Opt.validFromS-Cons-iff le-0-eq le-SucE length-0-conv length-Suc-conv less.prems*(*3*))
hence *finalO s1* **using** *less*(*3*−*6*)
using *Opt.completed-Cons Opt.completed-Nil* **by** *blast*
hence *f4*: *finalO s2* **using** *f34* **by** *blast*
hence *tr2*: *tr2* = [] ∨ *tr2* = [*s2*]
by (*metis Opt.final-def Simple-Transition-System.validFromS-Cons-iff less.prems*(*5*) *neq-Nil-conv*)
show *?thesis* **apply**(*rule exI*[*of* - [*sv1*]], *rule exI*[*of* - [*sv2*]]) **using** *tr1 tr2*
using *f4 f34*
using *completedFromO-lastt less.prems*(*4*)
by (*auto simp add*: *lastt-def* ψ*-def*)
next
case *False*
then obtain *s13 tr1′* **where** *tr1*: *tr1* = *s13* # *tr1′* **and** *tr1′NE*: *tr1′* ≠ []
by (*cases tr1*, *auto*)
have *s13*[*simp*]: *s13* = *s1* **using** ‹*Opt.validFromS s1 tr1*›
by (*simp add*: *Opt.validFromS-Cons-iff tr1*)
obtain *s1′* **where**
trn3: *validTransO* (*s1*,*s1′*) **and**

tr1': *Opt.validFromS s1' tr1'* **using** ‹*Opt.validFromS s1 tr1*›
unfolding *tr1 s13* **by** (*metis tr1'NE Simple-Transition-System.validFromS-Cons-iff*)
have *r3'*: *reachO s1'* **using** *r3 trn3* **by** (*metis Opt.reach.Step fst-conv snd-conv*)
have *f3*: ¬ *finalO s1* **using** *Opt.final-def trn3* **by** *blast*
hence *f4*: ¬ *finalO s2* **using** *f34* **by** *blast*
hence *tr2*: ¬ *length tr2* ≤ *Suc 0*
by (*metis* (*no-types*, *opaque-lifting*) *Opt.completed-Cons Opt.completed-Nil Simple-Transition-System.validFromS-Cons-iff Suc-n-not-le-n bot-nat-0.extremum le-Suc-eq length-Cons less.prems*(*5*) *less.prems*(*6*) *list.exhaust order-antisym-conv*)
then obtain *s24 tr2'* **where** *tr2*: *tr2* = *s24* # *tr2'* **and** *tr2'NE*: *tr2'* ≠ []
by (*cases tr2*, *auto*)
have *s24*[*simp*]: *s24* = *s2* **using** ‹*Opt.validFromS s2 tr2*›
by (*simp add*: *Opt.validFromS-Cons-iff tr2*)
obtain *s2'* **where**
trn4: *validTransO* (*s2*,*s2'*) ∨ (*s2* = *s2'* ∧ *tr2'* = []) **and**
tr2': *Opt.validFromS s2' tr2'* **using** ‹*Opt.validFromS s2 tr2*›
unfolding *tr2 s24* **using** *Opt.validFromS-Cons-iff* **by** *auto*
have *r34'*: *reachO s1' reachO s2'*
using *r3 trn3 r4 trn4* **by** (*metis Opt.reach.Step fst-conv snd-conv*)+
note *r3'* = *r34'*(*1*) **note** *r4'* = *r34'*(*2*)
define *statA'* **where** *statA'*: *statA'* = *sstatA' statA s1 s2*
have ¬ *isIntO s1* ∨ ¬ *isIntO s2* ∨ (*isIntO s1* ∧ *isIntO s2*)
by *auto*
thus *?thesis*
proof *safe*
assume *isAO3*: ¬ *isIntO s1*
have *O33'*: *Opt.O tr1* = *Opt.O tr1' Opt.A tr1* = *Opt.A tr1'*
using *isAO3* **unfolding** *tr1* **by** *auto*
have *A34'*: *Opt.A tr1'* = *Opt.A tr2*
using *A34 O33'*(*2*) **by** *auto*
have *m*: *match1* Δ *s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have (¬ *isSecO s1* ∧ Δ ∞ *s1' s2 statA sv1 sv2 statO*) ∨
(*eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧ *match1-1* Δ *s1 s1' s2 statA sv1 sv2 statO*) ∨
(*eqSec sv1 s1* ∧ ¬ *isSecV sv2* ∧ *Van.eqAct sv1 sv2* ∧ *match1-12* Δ *s1 s1' s2 statA sv1 sv2 statO*)
using *m isAO3 trn3 ok* **unfolding** *match1-def* **by** *auto*
thus *?thesis*
proof *safe*
assume ¬ *isSecO s1* **and** Δ: Δ ∞ *s1' s2 statA sv1 sv2 statO*
hence *S3*: *Opt.S tr1'* = *Opt.S tr1* **unfolding** *tr1* **by** *auto*
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1' s2 tr2 statO sv1 trv1 sv2 trv2*
using *less*(*1*)[*of tr1' tr2*, *OF* - Δ - - - - - - *r3' r4 r12*, *unfolded O33'*, *simplified*]
using *less.prems tr1' ok A34' f3 f4* **unfolding** *tr1 Opt.completedFrom-def*
by (*auto split*: *if-splits simp*: ψ*-def lastt-def*)
show *?thesis* **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *trv2*])

using ψ *O33′ S3* **unfolding** ψ*-def*
using *completedFromO-lastt less.prems(4)*
by (*auto simp add: tr1 tr1′NE*)
next
assume *trn13*: *eqSec sv1 s1* **and**
Atrn1: ¬ *isIntV sv1* **and** *match1-1* Δ *s1 s1′ s2 statA sv1 sv2 statO*
then obtain *sv1′* **where**
trn1: *validTransV* (*sv1*,*sv1′*) **and**
Δ: Δ ∞ *s1′ s2 statA sv1′ sv2 statO*
unfolding *match1-1-def* **by** *auto*
have *r1′*: *reachV sv1′* **using** *r1 trn1* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1′ s2 tr2 statO sv1′ trv1 sv2 trv2*
using *less(1)[of tr1′ tr2, OF - Δ - - - - - - r3′ r4 r1′ r2, unfolded O33′, simplified]*
using *less.prems tr1′ ok A34′ f3 f4* **unfolding** *tr1 tr2 Opt.completedFrom-def*

by (*auto simp*: ψ*-def lastt-def split*: *if-splits*)
show *?thesis* **apply**(*rule exI[of - sv1 # trv1]*) **apply**(*rule exI[of - trv2]*)
using ψ *O33′* **unfolding** *tr1 tr2 Van.completedFrom-def*
using *Van.validFromS-Cons trn1 tr1′NE tr2′NE*
using *isAO3 ok Atrn1 eqSec-S-Cons trn13*
unfolding ψ*-def* **using** *completedFromO-lastt less.prems(4) tr1* **by** *auto*

next
assume *sv2*: ¬ *isSecV sv2* **and** *trn13*: *eqSec sv1 s1* **and**
Atrn12: *Van.eqAct sv1 sv2* **and** *match1-12* Δ *s1 s1′ s2 statA sv1 sv2 statO*
then obtain *sv1′ sv2′ statO′* **where**
statO′: *statO′* = *sstatO′ statO sv1 sv2* **and**
trn1: *validTransV* (*sv1*,*sv1′*) **and**
trn2: *validTransV* (*sv2*,*sv2′*) **and**
Δ: Δ ∞ *s1′ s2 statA sv1′ sv2′ statO′*
unfolding *match1-12-def* **by** *auto*
have *r12′*: *reachV sv1′ reachV sv2′*
using *r1 trn1 r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *trv1 trv2* **where** ψ: ψ *s1′ tr1′ s2 tr2 statO′ sv1′ trv1 sv2′ trv2*
using *less(1)[of tr1′ tr2, OF - Δ - - - - - - r3′ r4 r12′, unfolded O33′, simplified]*
using *less.prems tr1′ ok A34′ f3 f4* **unfolding** *tr1 tr2 Opt.completedFrom-def statO′ sstatO′-def*
by *auto presburger+*
show *?thesis* **apply**(*rule exI[of - sv1 # trv1]*) **apply**(*rule exI[of - sv2 # trv2]*)
using ψ *O33′ tr1′NE tr2′NE sv2*
using *Van.validFromS-Cons trn1 trn2*
using *isAO3 ok Atrn12 eqSec-S-Cons trn13 f3 f34 s13*
unfolding ψ*-def tr1 Van.completedFrom-def Van.eqAct-def statO′ sstatO′-def*
using *Van.A.Cons-unfold tr1′ trn3* **by** *auto*
qed

next
assume *isAO4*: ¬ *isIntO s2*
have *O44'*: *Opt.O tr2* = *Opt.O tr2' Opt.A tr2* = *Opt.A tr2'*
using *isAO4* **unfolding** *tr2* **by** *auto*
have *A34'*: *Opt.A tr1* = *Opt.A tr2'*
using *A34 O44'*(*2*) **by** *auto*
have *m*: *match2* Δ *s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def*
by *auto*
have (¬ *isSecO s2* ∧ Δ ∞ *s1 s2' statA sv1 sv2 statO*) ∨
(*eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧ *match2-1* Δ *s1 s2 s2' statA sv1 sv2*
statO) ∨
(¬ *isSecV sv1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧ *match2-12* Δ *s1*
s2 s2' statA sv1 sv2 statO)
using *m isAO4 trn4 ok tr2'NE* **unfolding** *match2-def* **by** *auto*
thus *?thesis*
proof *safe*
assume ¬ *isSecO s2* **and** Δ: Δ ∞ *s1 s2' statA sv1 sv2 statO*
hence *S4*: *Opt.S tr2'* = *Opt.S tr2* **unfolding** *tr2* **by** *auto*
obtain *trv1 trv2* **where** *ψ*: *ψ s1 tr1 s2' tr2' statO sv1 trv1 sv2 trv2*
using *less*(*1*)[*of tr1 tr2'*, *OF* - Δ - - - - - - *r3 r4'*, *simplified*]
using *less.prems tr2' ok A34' tr1'NE tr2'NE* **unfolding** *tr1 tr2 Opt.completedFrom-def*
by (*cases isIntO s2*, *auto*)
show *?thesis* **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using *ψ O44' S4* **unfolding** *ψ-def*
using *completedFromO-lastt less.prems*(*6*)
unfolding *Opt.completedFrom-def* **using** *tr2 tr2'NE* **by** *auto*
next
assume *trn24*: *eqSec sv2 s2* **and**
Atrn2: ¬ *isIntV sv2* **and** *match2-1* Δ *s1 s2 s2' statA sv1 sv2 statO*
then obtain *sv2'* **where** *trn2*: *validTransV* (*sv2*,*sv2'*) **and**
Δ: Δ ∞ *s1 s2' statA sv1 sv2' statO*
unfolding *match2-1-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 trn2* **by** (*metis Van.reach.Step fst-conv*
snd-conv)
obtain *trv1 trv2* **where** *ψ*: *ψ s1 tr1 s2' tr2' statO sv1 trv1 sv2' trv2*
using *less*(*1*)[*of tr1 tr2'*, *OF* - Δ - - - - - - *r3 r4' r1 r2'*, *simplified*]
using *less.prems tr2' ok A34' tr1'NE tr2'NE* **unfolding** *tr1 tr2 Opt.completedFrom-def*
by (*cases isIntO s2*, *auto*)
show *?thesis* **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using *ψ tr1'NE tr2'NE*
using *Van.validFromS-Cons trn2*
using *isAO4 ok Atrn2 eqSec-S-Cons trn24*
unfolding *ψ-def tr1 tr2 s13 s24 Van.completedFrom-def lastt-def* **by** *auto*
next
assume *sv1*: ¬ *isSecV sv1* **and** *trn24*: *eqSec sv2 s2* **and**
Atrn12: *Van.eqAct sv1 sv2* **and** *match2-12* Δ *s1 s2 s2' statA sv1 sv2*
statO
then obtain *sv1' sv2' statO'* **where**
statO': *statO'* = *sstatO' statO sv1 sv2* **and**

trn1: *validTransV* (*sv1*,*sv1'*) **and**
trn2: *validTransV* (*sv2*,*sv2'*) **and**
Δ: Δ ∞ *s1 s2' statA sv1' sv2' statO'*
unfolding *match2-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'*
using *r1 trn1 r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2' tr2' statO' sv1' trv1 sv2' trv2*
using *less*(*1*)[*of tr1 tr2'*, *OF* - Δ - - - - - - *r3 r4' r12'*, *simplified*]
using *less.prems tr2' ok A34' tr1'NE tr2'NE* **unfolding** *tr1 tr2 Opt.completedFrom-def statO' sstatO'-def*
by (*cases isIntO s2*, *auto*)
show *?thesis* **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using ψ *O44' tr1'NE tr2'NE sv1*
using *Van.validFromS-Cons trn1 trn2*
using *isAO4 ok Atrn12 eqSec-S-Cons trn24*
unfolding ψ*-def tr2 tr1'NE Van.completedFrom-def Van.eqAct-def statO' sstatO'-def*
using *Van.A.Cons-unfold tr2' trn4* **by** *auto*
qed
next
assume *isAO34*: *isIntO s1 isIntO s2*
have *A34'*: *getActO s1* = *getActO s2 Opt.A tr1'* = *Opt.A tr2'*
using *A34 isAO34 tr1'NE tr2'NE* **unfolding** *tr1 tr2* **by** *auto*
have *O33'*: *Opt.O tr1* = *getObsO s1* # *Opt.O tr1'* **and**
O44': *Opt.O tr2* = *getObsO s2* # *Opt.O tr2'*
using *isAO34 tr1'NE tr2'NE* **unfolding** *tr1 s13 tr2 s24* **by** *auto*
have *m*: *match12* Δ *s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *statA' match-def* **by** *auto*
have *trn34*: *getObsO s1* = *getObsO s2* ∨ *statA'* = *Diff*
using *isAO34* **unfolding** *statA' sstatA'-def* **by** (*cases statA*,*auto*)
have (¬ *isSecO s1* ∧ ¬ *isSecO s2* ∧ (*statA* = *statA'* ∨ *statO* = *Diff*) ∧ Δ ∞ *s1' s2' statA' sv1 sv2 statO*)
∨
(¬ *isSecO s2* ∧ *eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧
(*statA* = *statA'* ∨ *statO* = *Diff*) ∧
match12-1 Δ *s1' s2' statA' sv1 sv2 statO*)
∨
(¬ *isSecO s1* ∧ *eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧
(*statA* = *statA'* ∨ *statO* = *Diff*) ∧
match12-2 Δ *s1' s2' statA' sv1 sv2 statO*)
∨
(*eqSec sv1 s1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧
match12-12 Δ *s1' s2' statA' sv1 sv2 statO*)
(**is** *?K1* ∨ *?K2* ∨ *?K3* ∨ *?K4*)
using *m*[*unfolded match12-def*, *rule-format*, *of s1' s2'*]
isAO34 A34' trn3 trn4 tr1'NE tr2'NE
unfolding *s13 s24 trn34 statA' Opt.eqAct-def sstatA'-def* **by** *auto*
thus *?thesis* **proof** (*elim disjE*)

assume *K1*: *?K1* **hence** Δ: Δ ∞ *s1′ s2′ statA′ sv1 sv2 statO* **by** *simp*
have *ok′*: (*statA′* = *Diff* ⟶ *statO* = *Diff*)
using *ok K1* **unfolding** *statA′* **using** *isAO34* **by** *auto*
obtain *trv1 trv2* **where** ψ: ψ *s1′ tr1′ s2′ tr2′ statO sv1 trv1 sv2 trv2*
using *less*(*1*)[*of tr1′ tr2′*, *OF* - Δ - - - - - - *r34′ r12*, *simplified*]
using *less.prems tr1′ tr2′ ok′ A34′ isAO34 tr1′NE tr2′NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using ψ *trn34 O33′ O44′ K1 ok* **unfolding** ψ*-def tr1 tr2*
using *completedFromO-lastt less.prems*(*4*,*6*)
unfolding *Opt.completedFrom-def* **using** *tr1 tr2 tr1′NE tr2′NE* **by** *auto*
next
assume *K2*: *?K2*
then obtain *sv1′* **where**
trn1: *validTransV* (*sv1*,*sv1′*) **and**
trn13: *eqSec sv1 s1* **and**
Atrn1: ¬ *isIntV sv1* **and** *ok′*: (*statA′* = *statA* ∨ *statO* = *Diff*) **and**
Δ: Δ ∞ *s1′ s2′ statA′ sv1′ sv2 statO*
unfolding *match12-1-def* **by** *auto*
have *r1′*: *reachV sv1′* **using** *r1 trn1* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1′ tr1′ s2′ tr2′ statO sv1′ trv1 sv2 trv2*
using *less*(*1*)[*of tr1′ tr2′*, *OF* - Δ - - - - - - *r34′ r1′ r2*, *simplified*]
using *less.prems tr1′ tr2′ ok′ A34′ tr1′NE tr2′NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using ψ *O33′ O44′ tr1′NE tr2′NE* **unfolding** *tr1 tr2*
using *Van.validFromS-Cons trn1 ok*
using *K2 ok′ Atrn1 eqSec-S-Cons trn13 trn34*
unfolding *statA′ Van.completedFrom-def eqSec-def*
using *s13 tr1 tr1′ tr2′ trn3 trn4*
by *simp* (*smt* (*verit*, *ccfv-SIG*) *Opt.S.simps*(*2*) *Simple-Transition-System.lastt-Cons*

Van.A.Cons-unfold Van.O.Cons-unfold ψ*-def list.simps*(*3*) *status.simps*(*1*))
next
assume *K3*: *?K3*
then obtain *sv2′* **where**
trn2: *validTransV* (*sv2*,*sv2′*) **and**
trn24: *eqSec sv2 s2* **and**
Atrn2: ¬ *isIntV sv2* **and** *ok′*: (*statA′* = *statA* ∨ *statO* = *Diff*) **and**
Δ: Δ ∞ *s1′ s2′ statA′ sv1 sv2′ statO*
unfolding *match12-2-def* **by** *auto*
have *r2′*: *reachV sv2′* **using** *r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1′ tr1′ s2′ tr2′ statO sv1 trv1 sv2′ trv2*
using *less*(*1*)[*of tr1′ tr2′*, *OF* - Δ - - - - - - *r34′ r1 r2′*, *simplified*]
using *less.prems tr1′ tr2′ ok′ A34′ tr1′NE tr2′NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])

using *ψ O33′ O44′ tr1′NE tr2′NE* **unfolding** *ψ-def tr1 tr2*
using *Van.validFromS-Cons trn2 ok*
using *K3 ok′ Atrn2 eqSec-S-Cons trn24 trn34*
unfolding *statA′ Van.completedFrom-def*
by *simp* (*metis last.simps lastt-def list.simps*(*3*) *status.distinct*(*2*))
next
assume *K4*: *?K4*
then obtain *sv1′ sv2′ statO′* **where** *0*:
statO′ = *sstatO′ statO sv1 sv2*
validTransV (*sv1*,*sv1′*)
eqSec sv1 s1
validTransV (*sv2*,*sv2′*)
eqSec sv2 s2
Van.eqAct sv1 sv2
and *ok′*: *statA′* = *Diff* ⟶ *statO′* = *Diff* **and** Δ: Δ ∞ *s1′ s2′ statA′ sv1′ sv2′ statO′*
unfolding *match12-12-def* **by** *auto*
have *r12′*: *reachV sv1′ reachV sv2′* **using** *r1 r2 0*
by (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *trv1 trv2* **where** *ψ*: *ψ s1′ tr1′ s2′ tr2′ statO′ sv1′ trv1 sv2′ trv2*
using *less*(*1*)[*of tr1′ tr2′*, *OF* - Δ - - - - - - *r34′ r12′*, *simplified*]
using *less.prems tr1′ tr2′ ok′ A34′ tr1′NE tr2′NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using *trn34*
using *ψ O33′ O44′ isAO34 tr1′NE tr2′NE* **unfolding** *ψ-def tr1 tr2*
using *Van.validFromS-Cons 0*
using *K4 eqSec-S-Cons*
unfolding *statA′ Van.eqAct-def Van.completedFrom-def match12-12-def sstatO′-def*
by (*smt* (*z3*) *Simple-Transition-System.lastt-Cons Van.A.Cons-unfold Van.O.Cons-unfold*
completedFromO-lastt f3 f34 lastt-Nil less.prems(*4*) *less.prems*(*6*) *list.inject s13*
s24 status.simps(*1*) *tr1 tr1′ tr2 tr2′ trn3 trn4 updStat.simps*(*1*) *updStat.simps*(*3*))
qed
qed
qed
qed
qed

theorem *unwind-rsecure*:
assumes *init*: *initCond* Δ
and *unwind*: *unwindCond* Δ
shows *rsecure*
apply (*rule rsecure-strong*)
apply (*elim conjE*)

subgoal for *s1 tr1 s2 tr2*
using *init* **unfolding** *initCond-def*
apply (*erule-tac allE*[*of - s1*])
apply (*elim allE*[*of - s2*] *conjE*)
apply (*elim impE*[*of* ‹*istateO s1* ∧ *istateO s2*›] *exE conjE*)
subgoal by *clarify*
subgoal for *sv1 sv2*
using *unwind* **apply** (*drule-tac unwindCond-ex-ψ*, *blast+*)
subgoal by (*rule Transition-System.reach.Istate*)
subgoal by (*rule Transition-System.reach.Istate*)
subgoal by (*rule Transition-System.reach.Istate*)
subgoal by (*rule Transition-System.reach.Istate*)
apply (*elim exE*)
subgoal for *trv1 trv2*
apply (*rule exI*[*of - sv1*], *rule exI*[*of - trv1*], *rule exI*[*of - sv2*], *rule exI*[*of - trv2*])
by *clarify*
.
.
.

3.4 Compositional unwinding

We allow networks of unwinding relations that unwind into each other, which offer a compositional alternative to monolithic unwinding.

definition *unwindIntoCond* ::
(*enat* ⇒ ′*stateO* ⇒ ′*stateO* ⇒ *status* ⇒ ′*stateV* ⇒ ′*stateV* ⇒ *status* ⇒ *bool*) ⇒
(*enat* ⇒ ′*stateO* ⇒ ′*stateO* ⇒ *status* ⇒ ′*stateV* ⇒ ′*stateV* ⇒ *status* ⇒ *bool*)
⇒ *bool*
where
unwindIntoCond Δ Δ′ ≡ ∀ *w s1 s2 statA sv1 sv2 statO*.
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Δ *w s1 s2 statA sv1 sv2 statO* ⟶
(*finalO s1* ⟷ *finalO s2*) ∧ (*finalV sv1* ⟷ *finalO s1*) ∧ (*finalV sv2* ⟷ *finalO s2*)
∧
(*statA* = *Eq* ⟶ (*isIntO s1* ⟷ *isIntO s2*))
∧
((∃ *v*<*w*. *proact* Δ′ *v s1 s2 statA sv1 sv2 statO*)
∨
match Δ′ *s1 s2 statA sv1 sv2 statO*)

lemma *unwindIntoCond-simpleI*:
assumes
⋀*w s1 s2 statA sv1 sv2 statO*.
reachO s1 ⟹ *reachO s2* ⟹ *reachV sv1* ⟹ *reachV sv2* ⟹
Δ *w s1 s2 statA sv1 sv2 statO*
⟹

($finalO\ s1 \longleftrightarrow finalO\ s2$) ∧ ($finalV\ sv1 \longleftrightarrow finalO\ s1$) ∧ ($finalV\ sv2 \longleftrightarrow finalO\ s2$)
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$
$statA = Eq$
$\Longrightarrow$
$isIntO\ s1 \longleftrightarrow isIntO\ s2$
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$match\ \Delta'\ s1\ s2\ statA\ sv1\ sv2\ statO$
shows *unwindIntoCond* $\Delta\ \Delta'$
using *assms* **unfolding** *unwindIntoCond-def* **by** *auto*

lemma *unwindIntoCond-simpleI2*:
assumes
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
($finalO\ s1 \longleftrightarrow finalO\ s2$) ∧ ($finalV\ sv1 \longleftrightarrow finalO\ s1$) ∧ ($finalV\ sv2 \longleftrightarrow finalO\ s2$)
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$
$statA = Eq$
$\Longrightarrow$
$isIntO\ s1 \longleftrightarrow isIntO\ s2$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(\exists\, v<w.\ proact\ \Delta'\ v\ s1\ s2\ statA\ sv1\ sv2\ statO)$
shows *unwindIntoCond* $\Delta\ \Delta'$
using *assms* **unfolding** *unwindIntoCond-def* **by** *auto*

lemma *unwindIntoCond-simpleIB*:
assumes
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
($finalO\ s1 \longleftrightarrow finalO\ s2$) ∧ ($finalV\ sv1 \longleftrightarrow finalO\ s1$) ∧ ($finalV\ sv2 \longleftrightarrow finalO\ s2$)

and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
statA = *Eq*
$\Longrightarrow$
isIntO s1 $\longleftrightarrow$ *isIntO s2*
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
($\exists\, v{<}w.$ *proact* Δ' *v s1 s2 statA sv1 sv2 statO*) $\vee$ *match* Δ' *s1 s2 statA sv1 sv2 statO*
shows *unwindIntoCond* Δ Δ'
using *assms* **unfolding** *unwindIntoCond-def* **by** *auto*

theorem *distrib-unwind-rsecure*:
assumes *m*: *0* < *m* **and** *nxt*: $\bigwedge i.\ i < (m{::}nat) \Longrightarrow nxt\ i \subseteq \{0..{<}m\}$
and *init*: *initCond* (Δs *0*)
and *step*: $\bigwedge i.\ i < m \Longrightarrow$
unwindIntoCond (Δs *i*) (λw *s1 s2 statA sv1 sv2 statO*.
$\exists\, j \in$ *nxt i*. Δs *j w s1 s2 statA sv1 sv2 statO*)
shows *rsecure*
proof−
define Δ **where** *D*: $\Delta \equiv \lambda w$ *s1 s2 statA sv1 sv2 statO*. $\exists\, i < m.$ Δs *i w s1 s2 statA sv1 sv2 statO*
have *i*: *initCond* Δ
using *init m* **unfolding** *initCond-def D* **by** *meson*
have *c*: *unwindCond* Δ **unfolding** *unwindCond-def* **apply**(*intro allI impI allI*)
apply(*subst* (*asm*) *D*) **apply** (*elim exE conjE*)
subgoal for *w s1 s2 statA sv1 sv2 statO i*
apply(*frule step*) **unfolding** *unwindIntoCond-def*
apply(*erule allE*[*of - w*])
apply(*erule allE*[*of - s1*]) **apply**(*erule allE*[*of - s2*]) **apply**(*erule allE*[*of - statA*])
apply(*erule allE*[*of - sv1*]) **apply**(*erule allE*[*of - sv2*]) **apply**(*erule allE*[*of - statO*])
apply *simp* **apply**(*elim conjE*)
apply(*erule disjE*)
subgoal apply(*rule disjI1*)
subgoal apply(*elim exE conjE*) **subgoal for** *v*
apply(*rule exI*[*of - v*], *simp*)
apply(*rule proact-mono*[*of* λw *s1 s2 statA sv1 sv2 statO*. $\exists j{\in}$*nxt i*. Δs *j w s1 s2 statA sv1 sv2 statO*])
subgoal unfolding *le-fun-def D* **by** *simp* (*meson atLeastLessThan-iff nxt subsetD*)
subgoal . . . .
subgoal apply(*rule disjI2*)

apply(*rule match-mono*[*of* λ*w s1 s2 statA sv1 sv2 statO*. $\exists$ *j*$\in$*nxt i*. Δ*s j w s1 s2 statA sv1 sv2 statO*])
subgoal unfolding *le-fun-def D* **by** *simp* (*meson atLeastLessThan-iff nxt subsetD*)
subgoal . . . .
show *?thesis* **using** *unwind-rsecure*[*OF i c*] **.**
qed

corollary *linear-unwind-rsecure*:
assumes *init*: *initCond* (Δ*s 0*)
and *step*: ($\bigwedge$*i*. *i* < *m* $\Longrightarrow$
unwindIntoCond (Δ*s i*) (λ*w s1 s2 statA sv1 sv2 statO*.
Δ*s i w s1 s2 statA sv1 sv2 statO* $\vee$
Δ*s* (*Suc i*) *w s1 s2 statA sv1 sv2 statO*))
and *finish*: *unwindIntoCond* (Δ*s m*) (Δ*s m*)
shows *rsecure*
apply(*rule distrib-unwind-rsecure*[*of Suc m* λ*i*. *if i*<*m then* {*i*,*Suc i*} *else* {*i*} Δ*s*, *OF - - init*])
using *step finish*
by (*auto simp*: *less-Suc-eq-le*)

definition *oor* **where**
oor Δ Δ_2 $\equiv$ λ*w s1 s2 statA sv1 sv2 statO*.
Δ *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w s1 s2 statA sv1 sv2 statO*

lemma *oorI1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor* Δ Δ_2 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor-def* **by** *simp*

lemma *oorI2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor* Δ Δ_2 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor-def* **by** *simp*

definition *oor3* **where**
oor3 Δ Δ_2 Δ_3 $\equiv$ λ*w s1 s2 statA sv1 sv2 statO*.
Δ *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w s1 s2 statA sv1 sv2 statO*

lemma *oor3I1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor3-def* **by** *simp*

lemma *oor3I2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor3-def* **by** *simp*

lemma *oor3I3*:

Δ_3 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor3-def* **by** *simp*

definition *oor4* **where**
oor4 Δ Δ_2 Δ_3 Δ_4 $\equiv$ λ*w s1 s2 statA sv1 sv2 statO.*
Δ *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_4 *w s1 s2 statA sv1 sv2 statO*

lemma *oor4I1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor4-def* **by** *simp*

lemma *oor4I2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor4-def* **by** *simp*

lemma *oor4I3*:
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor4-def* **by** *simp*

lemma *oor4I4*:
Δ_4 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor4-def* **by** *simp*

definition *oor5* **where**
oor5 Δ Δ_2 Δ_3 Δ_4 Δ_5 $\equiv$ λ*w s1 s2 statA sv1 sv2 statO.*
Δ *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_4 *w s1 s2 statA sv1 sv2 statO* $\vee$
Δ_5 *w s1 s2 statA sv1 sv2 statO*

lemma *oor5I1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *oor5I2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *oor5I3*:
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *oor5I4*:
Δ_4 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *oor5I5*:
Δ_5 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

definition *oor6* **where**
oor6 Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 $\equiv$ λ*w s1 s2 statA sv1 sv2 statO*.
Δ *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_4 *w s1 s2 statA sv1 sv2 statO* $\vee$
Δ_5 *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_6 *w s1 s2 statA sv1 sv2 statO*

lemma *oor6I1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor6-def* **by** *simp*

lemma *oor6I2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor6-def* **by** *simp*

lemma *oor6I3*:
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor6-def* **by** *simp*

lemma *oor6I4*:
Δ_4 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor6-def* **by** *simp*

lemma *oor6I5*:
Δ_5 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor6-def* **by** *simp*

lemma *oor6I6*:
Δ_6 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
unfolding *oor6-def* **by** *simp*

lemma *unwind-rsecure-foo*:
assumes *init*: *initCond* Δ_0
and *step0*: *unwindIntoCond* Δ_0 Δ*NN*
and *stepNN*: *unwindIntoCond* Δ*NN* (*oor5* Δ*NN* Δ*SN* Δ*NS* Δ*SS* Δ*nonspec*)

and *stepNS*: *unwindIntoCond* Δ*NS* (*oor4* Δ*NN* Δ*SN* Δ*NS* Δ*SS*)
and *stepSN*: *unwindIntoCond* Δ*SN* (*oor4* Δ*NN* Δ*SN* Δ*NS* Δ*SS*)
and *stepSS*: *unwindIntoCond* Δ*SS* (*oor4* Δ*NN* Δ*SN* Δ*NS* Δ*SS*)
and *stepNonspec*: *unwindIntoCond* Δ*nonspec* Δ*nonspec*
shows *rsecure*
proof−
define *m* **where** *m*: *m* ≡ (*6*::*nat*)
define Δ*s* **where** Δ*s*: Δ*s* ≡ λ*i*::*nat*.
if i = *0 then* Δ_0
else if i = *1 then* Δ*NN*
else if i = *2 then* Δ*SN*
else if i = *3 then* Δ*NS*
else if i = *4 then* Δ*SS*
else Δ*nonspec*
define *nxt* **where** *nxt*: *nxt* ≡ λ*i*::*nat*.
if i = *0 then* {*1*::*nat*}
else if i = *1 then* {*1*,*2*,*3*,*4*,*5*}
else if i = *2 then* {*1*,*2*,*3*,*4*}
else if i = *3 then* {*1*,*2*,*3*,*4*}
else if i = *4 then* {*1*,*2*,*3*,*4*}
else {*5*}
show *?thesis* **apply**(*rule distrib-unwind-rsecure*[*of m nxt* Δ*s*])
subgoal unfolding *m* **by** *auto*
subgoal unfolding *nxt m* **by** *auto*
subgoal using *init* **unfolding** Δ*s* **by** *auto*
subgoal
unfolding *m nxt* Δ*s*
using *step0 stepNN stepNS stepSN stepSS stepNonspec*
unfolding *oor4-def oor5-def* **by** *auto* **.**
qed

lemma *isIntO-match1*: *isIntO s1* ⟹ *match1* Δ *s1 s2 statA sv1 sv2 statO*
unfolding *match1-def* **by** *auto*

lemma *isIntO-match2*: *isIntO s2* ⟹ *match2* Δ *s1 s2 statA sv1 sv2 statO*
unfolding *match2-def* **by** *auto*

lemma *match1-1-oorI1*:
match1-1 Δ *s1 s1' s2 statA sv1 sv2 statO* ⟹
match1-1 (*oor* Δ Δ_2) *s1 s1' s2 statA sv1 sv2 statO*
apply(*rule match1-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match1-1-oorI2*:
match1-1 Δ_2 *s1 s1' s2 statA sv1 sv2 statO* ⟹
match1-1 (*oor* Δ Δ_2) *s1 s1' s2 statA sv1 sv2 statO*

apply(*rule match1-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match1-oorI1*:
match1 Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match1 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
apply(*rule match1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match1-oorI2*:
match1 Δ_2 *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match1 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
apply(*rule match1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match2-1-oorI1*:
match2-1 Δ *s1 s2 s2′ statA sv1 sv2 statO* $\Longrightarrow$
match2-1 (*oor* Δ Δ_2) *s1 s2 s2′ statA sv1 sv2 statO*
apply(*rule match2-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match2-1-oorI2*:
match2-1 Δ_2 *s1 s2 s2′ statA sv1 sv2 statO* $\Longrightarrow$
match2-1 (*oor* Δ Δ_2) *s1 s2 s2′ statA sv1 sv2 statO*
apply(*rule match2-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match2-oorI1*:
match2 Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match2 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
apply(*rule match2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match2-oorI2*:
match2 Δ_2 *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match2 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
apply(*rule match2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-oorI1*:
match12 Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match12 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
apply(*rule match12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-oorI2*:
match12 Δ_2 *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match12 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
apply(*rule match12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-1-oorI1*:
match12-1 Δ *s1′ s2′ statA′ sv1 sv2 statO* $\Longrightarrow$
match12-1 (*oor* Δ Δ_2) *s1′ s2′ statA′ sv1 sv2 statO*

apply(*rule match12-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-1-oorI2*:
match12-1 Δ_2 *s1′ s2′ statA′ sv1 sv2 statO* $\Longrightarrow$
match12-1 (*oor* Δ Δ_2) *s1′ s2′ statA′ sv1 sv2 statO*
apply(*rule match12-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-2-oorI1*:
match12-2 Δ *s2 s2′ statA′ sv1 sv2 statO* $\Longrightarrow$
match12-2 (*oor* Δ Δ_2) *s2 s2′ statA′ sv1 sv2 statO*
apply(*rule match12-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-2-oorI2*:
match12-2 Δ_2 *s2 s2′ statA′ sv1 sv2 statO* $\Longrightarrow$
match12-2 (*oor* Δ Δ_2) *s2 s2′ statA′ sv1 sv2 statO*
apply(*rule match12-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-12-oorI1*:
match12-12 Δ *s1′ s2′ statA′ sv1 sv2 statO* $\Longrightarrow$
match12-12 (*oor* Δ Δ_2) *s1′ s2′ statA′ sv1 sv2 statO*
apply(*rule match12-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-12-oorI2*:
match12-12 Δ_2 *s1′ s2′ statA′ sv1 sv2 statO* $\Longrightarrow$
match12-12 (*oor* Δ Δ_2) *s1′ s2′ statA′ sv1 sv2 statO*
apply(*rule match12-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match-oorI1*:
match Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
apply(*rule match-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match-oorI2*:
match Δ_2 *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
apply(*rule match-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *proact-oorI1*:
proact Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
proact (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
apply(*rule proact-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *proact-oorI2*:
proact Δ_2 *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
proact (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*

apply(*rule proact-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-1-oorI1*:
move-1 Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-1 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
apply(*rule move-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-1-oorI2*:
move-1 Δ_2 *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-1 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
apply(*rule move-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-2-oorI1*:
move-2 Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-2 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
apply(*rule move-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-2-oorI2*:
move-2 Δ_2 *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-2 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
apply(*rule move-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-12-oorI1*:
move-12 Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-12 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
apply(*rule move-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-12-oorI2*:
move-12 Δ_2 *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-12 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
apply(*rule move-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

end

context *Relative-Security-Determ*
begin

lemma *match1-1-defD*: *match1-1* Δ *s1 s1' s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ Δ ∞ *s1' s2 statA* (*nextO sv1*) *sv2 statO*
unfolding *match1-1-def validTrans-iff-next* **by** *simp*

lemma *match1-12-defD*: *match1-12* Δ *s1 s1' s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$
Δ ∞ *s1' s2 statA* (*nextO sv1*) (*nextO sv2*) (*sstatO' statO sv1 sv2*)
unfolding *match1-12-def validTrans-iff-next* **by** *simp*

lemmas *match1-defsD* = *match1-def match1-1-defD match1-12-defD*

lemma *match2-1-defD*: *match2-1* Δ *s1 s2 s2'* *statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ Δ ∞ *s1 s2'* *statA sv1* (*nextO sv2*) *statO*
unfolding *match2-1-def validTrans-iff-next* **by** *simp*

lemma *match2-12-defD*: *match2-12* Δ *s1 s2 s2'* *statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$ Δ ∞ *s1 s2'* *statA* (*nextO sv1*) (*nextO sv2*) (*sstatO'* *statO sv1 sv2*)
unfolding *match2-12-def validTrans-iff-next* **by** *simp*

lemmas *match2-defsD* = *match2-def match2-1-defD match2-12-defD*

lemma *match12-1-defD*: *match12-1* Δ *s1' s2'* *statA'* *sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ Δ ∞ *s1' s2'* *statA'* (*nextO sv1*) *sv2 statO*
unfolding *match12-1-def validTrans-iff-next* **by** *simp*

lemma *match12-2-defD*: *match12-2* Δ *s1' s2'* *statA'* *sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ Δ ∞ *s1' s2'* *statA'* *sv1* (*nextO sv2*) *statO*
unfolding *match12-2-def validTrans-iff-next* **by** *simp*

lemma *match12-12-defD*: *match12-12* Δ *s1' s2'* *statA'* *sv1 sv2 statO* $\longleftrightarrow$
(*let statO'* = *sstatO'* *statO sv1 sv2 in*
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$
(*statA'* = *Diff* $\longrightarrow$ *statO'* = *Diff*) $\wedge$
Δ ∞ *s1' s2'* *statA'* (*nextO sv1*) (*nextO sv2*) *statO'*)
unfolding *match12-12-def validTrans-iff-next* **by** *simp*

lemmas *match12-defsD* = *match12-def match12-1-defD match12-2-defD match12-12-defD*

lemmas *match-deep-defsD* = *match1-defsD match2-defsD match12-defsD*

lemma *move-1-defD*: *move-1* Δ *w s1 s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ Δ *w s1 s2 statA* (*nextO sv1*) *sv2 statO*
unfolding *move-1-def validTrans-iff-next* **by** *simp*

lemma *move-2-defD*: *move-2* Δ *w s1 s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ Δ *w s1 s2 statA sv1* (*nextO sv2*) *statO*
unfolding *move-2-def validTrans-iff-next* **by** *simp*

lemma *move-12-defD*: *move-12* Δ *w s1 s2 statA sv1 sv2 statO* $\longleftrightarrow$
(*let statO'* = *sstatO'* *statO sv1 sv2 in*
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$
Δ *w s1 s2 statA* (*nextO sv1*) (*nextO sv2*) *statO'*)
unfolding *move-12-def validTrans-iff-next* **by** *simp*

lemmas *proact-defsD* = *proact-def move-1-defD move-2-defD move-12-defD*

end

end

4 Unwinding Proof Method for Relative Security

This theory formalizes the notion of unwinding for relative security, and proves its soundness.

theory *Unwinding*
imports *Relative-Security*
begin

4.1 The types and operators underlying unwinding: status, matching operators, etc.

context *Rel-Sec*
begin

datatype *status* = *Eq* | *Diff*

fun *updStat* :: *status* ⇒ *bool* × *'a* ⇒ *bool* × *'a* ⇒ *status* **where**
updStat Eq (*True*,*a*) (*True*,*a'*) = (*if a* = *a' then Eq else Diff*)
|*updStat stat* - - = *stat*

definition *sstatO' statO sv1 sv2* = *updStat statO* (*isIntV sv1*, *getObsV sv1*) (*isIntV sv2*, *getObsV sv2*)
definition *sstatA' statA s1 s2* = *updStat statA* (*isIntO s1*, *getObsO s1*) (*isIntO s2*, *getObsO s2*)

lemma *updStat-EqI*:
assumes ‹*R* = *S*›
shows ‹*updStat Eq* (*P*, *R*) (*Q*, *S*) = *Eq*›
apply (*cases P*)
apply (*metis assms updStat.simps*(*1*) *updStat.simps*(*4*))
by (*cases Q*) *auto*

lemma *updStat-diff*:*updStat stat r r* = *Diff* ⟹ *stat* = *Diff*
by (*metis updStat.elims updStat.simps*(*1*))

definition *initCond* ::

(*enat* ⇒ *enat* ⇒ *enat* ⇒ *'stateO* ⇒ *'stateO* ⇒ *status* ⇒ *'stateV* ⇒ *'stateV* ⇒ *status* ⇒ *bool*) ⇒ *bool* **where**
initCond Δ ≡ ∀ *s1 s2*.
  istateO s1 ∧ *istateO s2*
  ⟶
  (∃ *sv1 sv2*. *istateV sv1* ∧ *istateV sv2* ∧ *corrState sv1 s1* ∧ *corrState sv2 s2*
    ∧ Δ ∞ ∞ ∞ *s1 s2 Eq sv1 sv2 Eq*)

definition *match1-1* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* ≡
 ∃ *sv1'*. *validTransV* (*sv1*,*sv1'*) ∧
  Δ ∞ *w1 w2 s1' s2 statA sv1' sv2 statO*

definition *match1-12* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* ≡
 (∃ *sv1' sv2'*.
  let statO' = sstatO' statO sv1 sv2 in
  validTransV (*sv1*,*sv1'*) ∧
  validTransV (*sv2*,*sv2'*) ∧
  Δ ∞ *w1 w2 s1' s2 statA sv1' sv2' statO'*)

definition *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* ≡
 ¬ *isIntO s1* ⟶
  (∀ *s1'*. *validTransO* (*s1*,*s1'*)
   ⟶
   (∃ *w1'*< *w1*. ∃ *w2'*< *w2*. ¬ *isSecO s1* ∧ Δ ∞ *w1' w2' s1' s2 statA sv1 sv2 statO*) ∨
   (∃ *w2'*< *w2*. *eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧ *match1-1* Δ ∞ *w2' s1 s1' s2 statA sv1 sv2 statO*) ∨
   (*eqSec sv1 s1* ∧ ¬ *isSecV sv2* ∧ *Van.eqAct sv1 sv2* ∧ *match1-12* Δ ∞ ∞ *s1 s1' s2 statA sv1 sv2 statO*))

lemmas *match1-defs* = *match1-def match1-1-def match1-12-def*

lemma *match1-1-mono*:
Δ ≤ Δ' ⟹ *match1-1* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* ⟹
 match1-1 Δ' *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
unfolding *le-fun-def match1-1-def* **by** *auto*

lemma *match1-12-mono*:
Δ ≤ Δ' ⟹ *match1-12* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* ⟹
 match1-12 Δ' *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
unfolding *le-fun-def match1-12-def* **by** *fastforce*

lemma *match1-mono*:
assumes Δ ≤ Δ'
shows *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* ⟹ *match1* Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*

unfolding *match1-def* **apply** *clarify* **subgoal for** *s1'* **apply**(*erule allE*[*of - s1'*])
using *match1-1-mono*[*OF assms*, *of - - s1 s1' s2 statA sv1 sv2 statO*]
match1-12-mono[*OF assms*, *of - - s1 s1' s2 statA sv1 sv2 statO*]
assms[*unfolded le-fun-def*, *rule-format*, *of - - - s1' s2 statA sv1 sv2 statO*]
by *fastforce* **.**

definition *match2-1* Δ *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\equiv$
$\exists$ *sv2'*. *validTransV* (*sv2*,*sv2'*) $\wedge$
Δ ∞ *w1 w2 s1 s2' statA sv1 sv2' statO*

definition *match2-12* Δ *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\equiv$
$\exists$ *sv1' sv2'*.
let statO' = *sstatO' statO sv1 sv2 in*
validTransV (*sv1*,*sv1'*) $\wedge$
validTransV (*sv2*,*sv2'*) $\wedge$
Δ ∞ *w1 w2 s1 s2' statA sv1' sv2' statO'*

definition *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\equiv$
$\neg$ *isIntO s2* $\longrightarrow$
($\forall$ *s2'*. *validTransO* (*s2*,*s2'*)
$\longrightarrow$
($\exists$ *w1'*< *w1*. $\exists$ *w2'*< *w2*. $\neg$ *isSecO s2* $\wedge$ Δ ∞ *w1' w2' s1 s2' statA sv1 sv2 statO*) $\vee$
($\exists$ *w1'*< *w1*. *eqSec sv2 s2* $\wedge$ $\neg$ *isIntV sv2* $\wedge$ *match2-1* Δ *w1'* ∞ *s1 s2 s2' statA sv1 sv2 statO*) $\vee$
($\neg$ *isSecV sv1* $\wedge$ *eqSec sv2 s2* $\wedge$ *Van.eqAct sv1 sv2* $\wedge$ *match2-12* Δ ∞ ∞ *s1 s2 s2' statA sv1 sv2 statO*))

lemmas *match2-defs* = *match2-def match2-1-def match2-12-def*

lemma *match2-1-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match2-1* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2-1* Δ' *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
unfolding *le-fun-def match2-1-def* **by** *auto*

lemma *match2-12-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match2-12* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2-12* Δ' *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
unfolding *le-fun-def match2-12-def* **by** *fastforce*

lemma *match2-mono*:
assumes $\Delta \leq \Delta'$
shows *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2* Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *match2-def* **apply** *clarify* **subgoal for** *s2'* **apply**(*erule allE*[*of - s2'*])
using *match2-1-mono*[*OF assms*, *of - - s1 s2 s2' statA sv1 sv2 statO*]
match2-12-mono[*OF assms*, *of - - s1 s2 s2' statA sv1 sv2 statO*]

assms[*unfolded le-fun-def*, *rule-format*, *of - - - s1 s2′ statA sv1 sv2 statO*]
by *fastforce* **.**

definition *match12-1* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* ≡
∃ *sv1′*. *validTransV* (*sv1*,*sv1′*) ∧
Δ ∞ *w1 w2 s1′ s2′ statA′ sv1′ sv2 statO*

definition *match12-2* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* ≡
∃ *sv2′*. *validTransV* (*sv2*,*sv2′*) ∧
Δ ∞ *w1 w2 s1′ s2′ statA′ sv1 sv2′ statO*

definition *match12-12* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* ≡
∃ *sv1′ sv2′*.
let statO′ = *sstatO′ statO sv1 sv2 in*
validTransV (*sv1*,*sv1′*) ∧
validTransV (*sv2*,*sv2′*) ∧
(*statA′* = *Diff* ⟶ *statO′* = *Diff*) ∧
Δ ∞ *w1 w2 s1′ s2′ statA′ sv1′ sv2′ statO′*

definition *match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* ≡
∀ *s1′ s2′*.
let statA′ = *sstatA′ statA s1 s2 in*
validTransO (*s1*,*s1′*) ∧
validTransO (*s2*,*s2′*) ∧
Opt.eqAct s1 s2 ∧
isIntO s1 ∧ *isIntO s2*
⟶
(∃ *w1′*< *w1*. ∃ *w2′*< *w2*. ¬ *isSecO s1* ∧ ¬ *isSecO s2* ∧ (*statA* = *statA′* ∨ *statO* = *Diff*) ∧
Δ ∞ *w1′ w2′ s1′ s2′ statA′ sv1 sv2 statO*)
∨
(∃ *w2′*< *w2*. ¬ *isSecO s2* ∧ *eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧
(*statA* = *statA′* ∨ *statO* = *Diff*) ∧
match12-1 Δ ∞ *w2′ s1′ s2′ statA′ sv1 sv2 statO*)
∨
(∃ *w1′*< *w1*. ¬ *isSecO s1* ∧ *eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧
(*statA* = *statA′* ∨ *statO* = *Diff*) ∧
match12-2 Δ *w1′* ∞ *s1′ s2′ statA′ sv1 sv2 statO*)
∨
(*eqSec sv1 s1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧
match12-12 Δ ∞ ∞ *s1′ s2′ statA′ sv1 sv2 statO*)

lemmas *match12-defs* = *match12-def match12-1-def match12-2-def match12-12-def*

lemma *match12-simpleI*:
assumes ⋀*s1′ s2′ statA′*.

statA′ = *sstatA′ statA s1 s2* ⟹
validTransO (*s1*,*s1′*) ⟹
validTransO (*s2*,*s2′*) ⟹
Opt.eqAct s1 s2 ⟹
(∃ *w1′*< *w1*. ∃ *w2′*< *w2*. ¬ *isSecO s1* ∧ ¬ *isSecO s2* ∧ (*statA* = *statA′* ∨ *statO* = *Diff*) ∧
Δ ∞ *w1′ w2′ s1′ s2′ statA′ sv1 sv2 statO*)
∨
(*eqSec sv1 s1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧
match12-12 Δ ∞ ∞ *s1′ s2′ statA′ sv1 sv2 statO*)
shows *match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using *assms* **unfolding** *match12-def Let-def* **by** *blast*

lemma *match12-1-mono*:
Δ ≤ Δ′ ⟹ *match12-1* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* ⟹ *match12-1* Δ′ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO*
unfolding *le-fun-def match12-1-def* **by** *auto*

lemma *match12-2-mono*:
Δ ≤ Δ′ ⟹ *match12-2* Δ *w1 w2 s1 s2′ statA′ sv1 sv2 statO* ⟹ *match12-2* Δ′ *w1 w2 s1 s2′ statA′ sv1 sv2 statO*
unfolding *le-fun-def match12-2-def* **by** *auto*

lemma *match12-12-mono*:
Δ ≤ Δ′ ⟹ *match12-12* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* ⟹ *match12-12* Δ′ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO*
unfolding *le-fun-def match12-12-def* **by** *fastforce*

lemma *match12-mono*:
assumes Δ ≤ Δ′
shows *match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* ⟹ *match12* Δ′ *w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *match12-def* **apply** *clarify* **subgoal for** *s1′ s2′* **apply**(*erule allE*[*of - s1′*]) **apply**(*erule allE*[*of - s2′*])
using *match12-1-mono*[*OF assms*, *of - - s1′ s2′ - sv1 sv2 statO*]
match12-2-mono[*OF assms*, *of - - s1′ s2′ - sv1 sv2 statO*]
match12-12-mono[*OF assms*, *of - - s1′ s2′ - sv1 sv2 statO*]
assms[*unfolded le-fun-def*, *rule-format*, *of - - - s1′ s2′ sstatA′ statA s1 s2 sv1 sv2 statO*]
apply *simp* **by** *blast* **.**

definition *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* ≡
match1 Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
∧
match2 Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
∧
match12 Δ *w1 w2 s1 s2 statA sv1 sv2 statO*

lemmas *match-defs* = *match1-def match2-def match12-def*
lemmas *match-deep-defs* = *match1-defs match2-defs match12-defs*

lemma *match-mono*:
assumes $\Delta \leq \Delta'$
shows *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match* Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *match-def* **using** *match1-mono*[*OF assms*] *match2-mono*[*OF assms*] *match12-mono*[*OF assms*] **by** *auto*

definition *move-1* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\equiv$
$\exists$ *sv1'*. *validTransV* (*sv1*,*sv1'*) $\wedge$
Δ *w w1 w2 s1 s2 statA sv1' sv2 statO*

definition *move-2* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\equiv$
$\exists$ *sv2'*. *validTransV* (*sv2*,*sv2'*) $\wedge$
Δ *w w1 w2 s1 s2 statA sv1 sv2' statO*

definition *move-12* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\equiv$
$\exists$ *sv1' sv2'*.
let statO' = *sstatO' statO sv1 sv2 in*
validTransV (*sv1*,*sv1'*) $\wedge$ *validTransV* (*sv2*,*sv2'*) $\wedge$
Δ *w w1 w2 s1 s2 statA sv1' sv2' statO'*

definition *proact* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\equiv$
($\neg$ *isSecV sv1* $\wedge$ $\neg$ *isIntV sv1* $\wedge$ *move-1* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*)
$\vee$
($\neg$ *isSecV sv2* $\wedge$ $\neg$ *isIntV sv2* $\wedge$ *move-2* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*)
$\vee$
($\neg$ *isSecV sv1* $\wedge$ $\neg$ *isSecV sv2* $\wedge$ *Van.eqAct sv1 sv2* $\wedge$ *move-12* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*)

lemmas *proact-defs* = *proact-def move-1-def move-2-def move-12-def*

lemma *move-1-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-1* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-1* Δ' *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *le-fun-def move-1-def* **by** *auto*

lemma *move-2-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-2* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-2* Δ' *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *le-fun-def move-2-def* **by** *auto*

lemma *move-12-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-12* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-12* Δ' *w w1*

w2 s1 s2 statA sv1 sv2 statO
unfolding *le-fun-def move-12-def* **by** *fastforce*

lemma *proact-mono*:
assumes $\Delta \leq \Delta'$
shows *proact* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *proact* Δ' *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *proact-def* **using** *move-1-mono*[*OF assms*] *move-2-mono*[*OF assms*] *move-12-mono*[*OF assms*] **by** *auto*

4.2 The definition of unwinding

definition *unwindCond* ::
(*enat* $\Rightarrow$ *enat* $\Rightarrow$ *enat* $\Rightarrow$ *'stateO* $\Rightarrow$ *'stateO* $\Rightarrow$ *status* $\Rightarrow$ *'stateV* $\Rightarrow$ *'stateV* $\Rightarrow$ *status* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*
where
unwindCond $\Delta \equiv \forall$ *w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 $\wedge$ *reachO s2* $\wedge$ *reachV sv1* $\wedge$ *reachV sv2* $\wedge$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\longrightarrow$
(*finalO s1* $\longleftrightarrow$ *finalO s2*) $\wedge$ (*finalV sv1* $\longleftrightarrow$ *finalO s1*) $\wedge$ (*finalV sv2* $\longleftrightarrow$ *finalO s2*)
$\wedge$
(*statA* = *Eq* $\longrightarrow$ (*isIntO s1* $\longleftrightarrow$ *isIntO s2*))
$\wedge$
(($\exists$ *v* < *w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*)
$\vee$
match Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
)

lemma *unwindCond-simpleI*:
assumes
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
(*finalO s1* $\longleftrightarrow$ *finalO s2*) $\wedge$ (*finalV sv1* $\longleftrightarrow$ *finalO s1*) $\wedge$ (*finalV sv2* $\longleftrightarrow$ *finalO s2*)
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *statA* = *Eq*
$\Longrightarrow$
isIntO s1 $\longleftrightarrow$ *isIntO s2*
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO*.

reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
match Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
shows *unwindCond* Δ
using *assms* **unfolding** *unwindCond-def* **by** *auto*

4.3 The soundness of unwinding

The proof of soundness for general unwinding is significantly more elaborate than that for the finitary case.

definition ψ *s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2* $\equiv$
trv1 $\neq$ [] $\wedge$ *trv2* $\neq$ [] $\wedge$
Van.validFromS sv1 trv1 $\wedge$
Van.validFromS sv2 trv2 $\wedge$
(*finalV* (*lastt sv1 trv1*) $\longleftrightarrow$ *finalO* (*lastt s1 tr1*)) $\wedge$ (*finalV* (*lastt sv2 trv2*) $\longleftrightarrow$ *finalO* (*lastt s2 tr2*)) $\wedge$
Van.S trv1 = *Opt.S tr1* $\wedge$ *Van.S trv2* = *Opt.S tr2* $\wedge$
Van.A trv1 = *Van.A trv2* $\wedge$
(*statO* = *Eq* $\wedge$ *Opt.O tr1* $\neq$ *Opt.O tr2* $\longrightarrow$ *Van.O trv1* $\neq$ *Van.O trv2*)

lemma ψ*-completedFrom*: *completedFromO s1 tr1* $\Longrightarrow$ *completedFromO s2 tr2* $\Longrightarrow$

ψ *s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2*
$\Longrightarrow$ *completedFromV sv1 trv1* $\wedge$ *completedFromV sv2 trv2*
unfolding ψ*-def Opt.completedFrom-def Van.completedFrom-def lastt-def*
by *presburger*

lemma *completedFromO-lastt*: *completedFromO s1 tr1* $\Longrightarrow$ *finalO* (*lastt s1 tr1*)
unfolding *Opt.completedFrom-def lastt-def* **by** *auto*

lemma *rsecure-strong*:
assumes
$\bigwedge$*s1 tr1 s2 tr2*.
istateO s1 $\wedge$ *Opt.validFromS s1 tr1* $\wedge$ *completedFromO s1 tr1* $\wedge$
istateO s2 $\wedge$ *Opt.validFromS s2 tr2* $\wedge$ *completedFromO s2 tr2* $\wedge$
Opt.A tr1 = *Opt.A tr2*
$\Longrightarrow$
$\exists$ *sv1 trv1 sv2 trv2*.
istateV sv1 $\wedge$ *istateV sv2* $\wedge$ *corrState sv1 s1* $\wedge$ *corrState sv2 s2* $\wedge$
ψ *s1 tr1 s2 tr2 Eq sv1 trv1 sv2 trv2*
shows *rsecure*
unfolding *rsecure-def2* **apply** *safe*
subgoal for *s1 tr1 s2 tr2*
using *assms*[*of s1 tr1 s2 tr2*]
using ψ*-completedFrom* ψ*-def completedFromO-lastt* **apply** *clarsimp* **by** *metis* **.**

proposition *unwindCond-ex-ψ*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* **and** *stat*: (*statA* = *Diff* ⟶ *statO* = *Diff*)
and *v*: *Opt.validFromS s1 tr1 Opt.completedFrom s1 tr1 Opt.validFromS s2 tr2 Opt.completedFrom s2 tr2*
and *tr14*: *Opt.A tr1* = *Opt.A tr2*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
shows ∃ *trv1 trv2*. ψ *s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2*
using *assms*(*2*−)
proof(*induction length tr1* + *length tr2 w*
arbitrary: *w1 w2 s1 s2 statA sv1 sv2 statO tr1 tr2 rule*: *less2-induct′*)
case (*less w tr1 tr2 w1 w2 s1 s2 statA sv1 sv2 statO*)
note *ok* = ‹*statA* = *Diff* ⟶ *statO* = *Diff*›
note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›
note *A34* = ‹*Opt.A tr1* = *Opt.A tr2*›
note *r34* = *less.prems*(*8*,*9*) **note** *r12* = *less.prems*(*10*,*11*)
note *r* = *r34 r12*
note *r3* = *r34*(*1*) **note** *r4* = *r34*(*2*) **note** *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)

have *i34*: *statA* = *Eq* ⟶ *isIntO s1* = *isIntO s2*
and *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1′*
where *0*: *validTransV* (*sv1*,*sv1′*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1′ sv2 statO*
unfolding *move-1-def* **by** *auto*
have *r1′*: *reachV sv1′* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2 tr2 statO sv1′ trv1 sv2 trv2*
using *less*(*2*)[*OF v*, *of tr1 tr2 w1 w2 s1 s2 statA sv1′ sv2 statO*, *simplified*, *OF* Δ *ok* - - - - - *r34 r1′ r2*]
using *A34 less.prems*(*3*−*6*) **by** *blast*
show *?thesis* **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using ψ *ok 0 sv1* **unfolding** *ψ-def Van.completedFrom-def* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*

then obtain *sv2'*
where *0*: *validTransV* (*sv2*,*sv2'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1 sv2' statO*
unfolding *move-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2 tr2 statO sv1 trv1 sv2' trv2*
using *less*(*2*)[*OF v*, *of tr1 tr2 w1 w2 s1 s2 statA sv1 sv2' statO*, *simplified*, *OF* Δ *ok - - - - - r34 r1 r2'*]
using *A34 less.prems*(*3*−*6*) **by** *blast*
show *?thesis* **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using ψ *ok 0 sv2* **unfolding** ψ*-def Van.completedFrom-def* **by** *auto*
next
assume *sv12*: ¬ *isSecV sv1* ¬ *isSecV sv2 Van.eqAct sv1 sv2*
and *move-12* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1' sv2' statO'*
where *0*: *statO'* = *sstatO' statO sv1 sv2*
validTransV (*sv1*,*sv1'*) ¬ *isSecV sv1*
validTransV (*sv2*,*sv2'*) ¬ *isSecV sv2*
Van.eqAct sv1 sv2
and Δ: Δ *v w1 w2 s1 s2 statA sv1' sv2' statO'*
unfolding *move-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'* **using** *r1 r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
have *ok'*: *statA* = *Diff* ⟶ *statO'* = *Diff* **using** *ok 0* **unfolding** *sstatO'-def* **by** (*cases statO*, *auto*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2 tr2 statO' sv1' trv1 sv2' trv2*
using *less*(*2*)[*OF v*, *of tr1 tr2 w1 w2 s1 s2 statA sv1' sv2' statO'*, *simplified*, *OF* Δ *ok' - - - - - r34 r12'*]
using *A34 less.prems*(*3*−*6*) **by** *blast*
show *?thesis* **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using ψ *ok' 0 sv12* **unfolding** ψ*-def sstatO'-def Van.completedFrom-def*
using *Van.A.Cons-unfold Van.eqAct-def completedFromO-lastt less.prems*(*4*)
less.prems(*6*) **by** *auto*
qed
next
assume *m*: *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
show *?thesis*
proof(*cases length tr1* ≤ *Suc 0*)
case *True* **note** *tr1* = *True*
hence *tr1* = [] ∨ *tr1* = [*s1*]
by (*metis Simple-Transition-System.validFromS-Cons-iff Suc-length-conv le-Suc-eq le-zero-eq length-0-conv less.prems*(*3*))
hence *finalO s1* **using** *less*(*3*−*6*)
using *Opt.completed-Cons Opt.completed-Nil* **by** *blast*
hence *f4*: *finalO s2* **using** *f34* **by** *blast*
hence *tr2*: *tr2* = [] ∨ *tr2* = [*s2*]
by (*metis Opt.final-def Simple-Transition-System.validFromS-Cons-iff less.prems*(*5*) *neq-Nil-conv*)

show *?thesis* **apply**(*rule exI*[*of* - [*sv1*]], *rule exI*[*of* - [*sv2*]]) **using** *tr1 tr2*
using *f4 f34*
using *completedFromO-lastt less.prems*(*4*)
by (*auto simp add*: *lastt-def ψ-def*)
next
case *False*
then obtain *s13 tr1′* **where** *tr1*: *tr1* = *s13* # *tr1′* **and** *tr1′NE*: *tr1′* ≠ []
by (*cases tr1*, *auto*)
have *s13*[*simp*]: *s13* = *s1* **using** ‹*Opt.validFromS s1 tr1*›
by (*simp add*: *Opt.validFromS-Cons-iff tr1*)
obtain *s1′* **where**
trn3: *validTransO* (*s1*,*s1′*) **and**
tr1′: *Opt.validFromS s1′ tr1′* **using** ‹*Opt.validFromS s1 tr1*›
unfolding *tr1 s13* **by** (*metis tr1′NE Simple-Transition-System.validFromS-Cons-iff*)
have *r3′*: *reachO s1′* **using** *r3 trn3* **by** (*metis Opt.reach.Step fst-conv snd-conv*)
have *f3*: ¬ *finalO s1* **using** *Opt.final-def trn3* **by** *blast*
hence *f4*: ¬ *finalO s2* **using** *f34* **by** *blast*
hence *tr2*: ¬ *length tr2* ≤ *Suc 0*
by (*metis Opt.completed-Cons Simple-Transition-System.validFromS-Cons-iff*

bot-nat-0.extremum completedFromO-def length-Cons less.prems(*5*) *less.prems*(*6*) *neq-Nil-conv not-less-eq-eq*)

then obtain *s24 tr2′* **where** *tr2*: *tr2* = *s24* # *tr2′* **and** *tr2′NE*: *tr2′* ≠ []
by (*cases tr2*, *auto*)
have *s24*[*simp*]: *s24* = *s2* **using** ‹*Opt.validFromS s2 tr2*›
by (*simp add*: *Opt.validFromS-Cons-iff tr2*)
obtain *s2′* **where**
trn4: *validTransO* (*s2*,*s2′*) ∨ (*s2* = *s2′* ∧ *tr2′* = []) **and**
tr2′: *Opt.validFromS s2′ tr2′* **using** ‹*Opt.validFromS s2 tr2*›
unfolding *tr2 s24* **using** *Opt.validFromS-Cons-iff* **by** *auto*
have *r34′*: *reachO s1′ reachO s2′*
using *r3 trn3 r4 trn4* **by** (*metis Opt.reach.Step fst-conv snd-conv*)+
note *r3′* = *r34′*(*1*) **note** *r4′* = *r34′*(*2*)
define *statA′* **where** *statA′*: *statA′* = *sstatA′ statA s1 s2*
have ¬ *isIntO s1* ∨ ¬ *isIntO s2* ∨ (*isIntO s1* ∧ *isIntO s2*)
by *auto*
thus *?thesis*
proof *safe*
assume *isAO3*: ¬ *isIntO s1*
have *O33′*: *Opt.O tr1* = *Opt.O tr1′ Opt.A tr1* = *Opt.A tr1′*
using *isAO3* **unfolding** *tr1* **by** *auto*
have *A34′*: *Opt.A tr1′* = *Opt.A tr2*
using *A34 O33′*(*2*) **by** *auto*
have *m*: *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have (∃ *w1′*<*w1*. ∃ *w2′*<*w2*. ¬ *isSecO s1* ∧ Δ ∞ *w1′ w2′ s1′ s2 statA sv1 sv2 statO*) ∨

(∃ *w2'<w2. eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧ *match1-1* Δ ∞ *w2' s1 s1' s2 statA sv1 sv2 statO*) ∨
(*eqSec sv1 s1* ∧ ¬ *isSecV sv2* ∧ *Van.eqAct sv1 sv2* ∧ *match1-12* Δ ∞ ∞ *s1 s1' s2 statA sv1 sv2 statO*)
using *m isAO3 trn3 ok* **unfolding** *match1-def* **by** *auto*
thus *?thesis*
proof *safe*
fix *w1' w2'*
assume ¬ *isSecO s1* **and** Δ: Δ ∞ *w1' w2' s1' s2 statA sv1 sv2 statO*
hence *S3*: *Opt.S tr1'* = *Opt.S tr1* **unfolding** *tr1* **by** *auto*
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1' s2 tr2 statO sv1 trv1 sv2 trv2*
using *less*(*1*)[*of tr1' tr2*, *OF* - Δ - - - - - - *r3' r4 r12*, *unfolded O33'*, *simplified*]
using *less.prems tr1' ok A34' f3 f4* **unfolding** *tr1 Opt.completedFrom-def*
by (*auto split*: *if-splits simp*: ψ*-def lastt-def*)
show *?thesis* **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using ψ *O33' S3* **unfolding** ψ*-def*
using *completedFromO-lastt less.prems*(*4*)
by (*auto simp add*: *tr1 tr1'NE*)
next
fix *w2'*
assume *trn13*: *eqSec sv1 s1* **and**
Atrn1: ¬ *isIntV sv1* **and** *match1-1* Δ ∞ *w2' s1 s1' s2 statA sv1 sv2 statO*
then obtain *sv1'* **where**
trn1: *validTransV* (*sv1*,*sv1'*) **and**
Δ: Δ ∞ ∞ *w2' s1' s2 statA sv1' sv2 statO*
unfolding *match1-1-def* **by** *auto*
have *r1'*: *reachV sv1'***using** *r1 trn1* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1' s2 tr2 statO sv1' trv1 sv2 trv2*
using *less*(*1*)[*of tr1' tr2*, *OF* - Δ - - - - - - *r3' r4 r1' r2*, *unfolded O33'*, *simplified*]
using *less.prems tr1' ok A34' f3 f4* **unfolding** *tr1 tr2 Opt.completedFrom-def*

by (*auto simp*: ψ*-def lastt-def split*: *if-splits*)
show *?thesis* **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using ψ *O33'* **unfolding** *tr1 tr2 Van.completedFrom-def*
using *Van.validFromS-Cons trn1 tr1'NE tr2'NE*
using *isAO3 ok Atrn1 eqSec-S-Cons trn13*
unfolding ψ*-def* **using** *completedFromO-lastt less.prems*(*4*) *tr1* **by** *auto*

next
assume *sv2*: ¬ *isSecV sv2* **and** *trn13*: *eqSec sv1 s1* **and**
Atrn12: *Van.eqAct sv1 sv2* **and** *match1-12* Δ ∞ ∞ *s1 s1' s2 statA sv1 sv2 statO*
then obtain *sv1' sv2' statO'* **where**
statO': *statO'* = *sstatO' statO sv1 sv2* **and**
trn1: *validTransV* (*sv1*,*sv1'*) **and**
trn2: *validTransV* (*sv2*,*sv2'*) **and**

Δ: Δ ∞ ∞ ∞ *s1' s2 statA sv1' sv2' statO'*
unfolding *match1-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'*
using *r1 trn1 r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *trv1 trv2* **where** ψ: ψ *s1' tr1' s2 tr2 statO' sv1' trv1 sv2' trv2*
using *less*(*1*)[*of tr1' tr2*, *OF* - Δ - - - - - - *r3' r4 r12'*, *unfolded O33'*, *simplified*]
using *less.prems tr1' ok A34' f3 f4* **unfolding** *tr1 tr2 Opt.completedFrom-def statO' sstatO'-def*
by *auto presburger*+
show *?thesis* **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using ψ *O33' tr1'NE tr2'NE sv2*
using *Van.validFromS-Cons trn1 trn2*
using *isAO3 ok Atrn12 eqSec-S-Cons trn13 f3 f34 s13*
unfolding *ψ-def tr1 Van.completedFrom-def Van.eqAct-def statO' sstatO'-def*
using *Van.A.Cons-unfold tr1' trn3* **by** *auto*
qed
next
assume *isAO4*: ¬ *isIntO s2*
have *O44'*: *Opt.O tr2* = *Opt.O tr2' Opt.A tr2* = *Opt.A tr2'*
using *isAO4* **unfolding** *tr2* **by** *auto*
have *A34'*: *Opt.A tr1* = *Opt.A tr2'*
using *A34 O44'*(*2*) **by** *auto*
have *m*: *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have (∃ *w1'*<*w1*. ∃ *w2'*<*w2*. ¬ *isSecO s2* ∧ Δ ∞ *w1' w2' s1 s2' statA sv1 sv2 statO*) ∨
(∃ *w1'*<*w1*. *eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧ *match2-1* Δ *w1'* ∞ *s1 s2 s2' statA sv1 sv2 statO*) ∨
(¬ *isSecV sv1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧ *match2-12* Δ ∞ ∞ *s1 s2 s2' statA sv1 sv2 statO*)
using *m isAO4 trn4 ok tr2'NE* **unfolding** *match2-def* **by** *auto*
thus *?thesis*
proof *safe*
fix *w1' w2'*
assume ¬ *isSecO s2* **and** Δ: Δ ∞ *w1' w2' s1 s2' statA sv1 sv2 statO*
hence *S4*: *Opt.S tr2'* = *Opt.S tr2* **unfolding** *tr2* **by** *auto*
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2' tr2' statO sv1 trv1 sv2 trv2*
using *less*(*1*)[*of tr1 tr2'*, *OF* - Δ - - - - - - *r3 r4'*, *simplified*]
using *less.prems tr2' ok A34' tr1'NE tr2'NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** (*cases isIntO s2*, *auto*)
show *?thesis* **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - trv2*])
using ψ *O44' S4* **unfolding** *ψ-def*
using *completedFromO-lastt less.prems*(*6*)
unfolding *Opt.completedFrom-def* **using** *tr2 tr2'NE* **by** *auto*
next
fix *w1'*
assume *trn24*: *eqSec sv2 s2* **and**

Atrn2: ¬ *isIntV sv2* **and** *match2-1* Δ *w1′* ∞ *s1 s2 s2′ statA sv1 sv2 statO*
then obtain *sv2′* **where** *trn2*: *validTransV* (*sv2*,*sv2′*) **and**
Δ: Δ ∞ *w1′* ∞ *s1 s2′ statA sv1 sv2′ statO*
unfolding *match2-1-def* **by** *auto*
have *r2′*: *reachV sv2′* **using** *r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2′ tr2′ statO sv1 trv1 sv2′ trv2*
using *less*(*1*)[*of tr1 tr2′*, *OF* - Δ - - - - - - *r3 r4′ r1 r2′*, *simplified*]
using *less.prems tr2′ ok A34′ tr1′NE tr2′NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** (*cases isIntO s2*, *auto*)
show *?thesis* **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using ψ *tr1′NE tr2′NE*
using *Van.validFromS-Cons trn2*
using *isAO4 ok Atrn2 eqSec-S-Cons trn24*
unfolding ψ*-def tr1 tr2 s13 s24 Van.completedFrom-def lastt-def* **by** *auto*
next
assume *sv1*: ¬ *isSecV sv1* **and** *trn24*: *eqSec sv2 s2* **and**
Atrn12: *Van.eqAct sv1 sv2* **and** *match2-12* Δ ∞ ∞ *s1 s2 s2′ statA sv1 sv2 statO*
then obtain *sv1′ sv2′ statO′* **where**
statO′: *statO′* = *sstatO′ statO sv1 sv2* **and**
trn1: *validTransV* (*sv1*,*sv1′*) **and**
trn2: *validTransV* (*sv2*,*sv2′*) **and**
Δ: Δ ∞ ∞ ∞ *s1 s2′ statA sv1′ sv2′ statO′*
unfolding *match2-12-def* **by** *auto*
have *r12′*: *reachV sv1′ reachV sv2′*
using *r1 trn1 r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *trv1 trv2* **where** ψ: ψ *s1 tr1 s2′ tr2′ statO′ sv1′ trv1 sv2′ trv2*
using *less*(*1*)[*of tr1 tr2′*, *OF* - Δ - - - - - - *r3 r4′ r12′*, *simplified*]
using *less.prems tr2′ ok A34′ tr1′NE tr2′NE* **unfolding** *tr1 tr2 Opt.completedFrom-def statO′ sstatO′-def*
by (*cases isIntO s2*, *auto*)
show *?thesis* **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using ψ *O44′ tr1′NE tr2′NE sv1*
using *Van.validFromS-Cons trn1 trn2*
using *isAO4 ok Atrn12 eqSec-S-Cons trn24*
unfolding ψ*-def tr2 tr1′NE Van.completedFrom-def Van.eqAct-def*
statO′ sstatO′-def
using *Van.A.Cons-unfold tr2′ trn4* **by** *auto*
qed
next
assume *isAO34*: *isIntO s1 isIntO s2*
have *A34′*: *getActO s1* = *getActO s2 Opt.A tr1′* = *Opt.A tr2′*
using *A34 isAO34 tr1′NE tr2′NE* **unfolding** *tr1 tr2* **by** *auto*
have *O33′*: *Opt.O tr1* = *getObsO s1* # *Opt.O tr1′* **and**
O44′: *Opt.O tr2* = *getObsO s2* # *Opt.O tr2′*
using *isAO34 tr1′NE tr2′NE* **unfolding** *tr1 s13 tr2 s24* **by** *auto*
have *m*: *match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding**

statA′ match-def **by** *auto*
have *trn34*: *getObsO s1* = *getObsO s2* ∨ *statA′* = *Diff*
using *isAO34* **unfolding** *statA′ sstatA′-def* **by** (*cases statA*,*auto*)
have (∃ *w1′*<*w1*. ∃ *w2′*<*w2*. ¬ *isSecO s1* ∧ ¬ *isSecO s2* ∧ (*statA* = *statA′* ∨ *statO* = *Diff*) ∧
Δ ∞ *w1′ w2′ s1′ s2′ statA′ sv1 sv2 statO*)
∨
(∃ *w2′*<*w2*. ¬ *isSecO s2* ∧ *eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧
(*statA* = *statA′* ∨ *statO* = *Diff*) ∧
match12-1 Δ ∞ *w2′ s1′ s2′ statA′ sv1 sv2 statO*)
∨
(∃ *w1′*<*w1*. ¬ *isSecO s1* ∧ *eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧
(*statA* = *statA′* ∨ *statO* = *Diff*) ∧
match12-2 Δ *w1′* ∞ *s1′ s2′ statA′ sv1 sv2 statO*)
∨
(*eqSec sv1 s1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧
match12-12 Δ ∞ ∞ *s1′ s2′ statA′ sv1 sv2 statO*)
(**is** *?K1* ∨ *?K2* ∨ *?K3* ∨ *?K4*)
using *m*[*unfolded match12-def*, *rule-format*, *of s1′ s2′*]
isAO34 A34′ trn3 trn4 tr1′NE tr2′NE
unfolding *s13 s24 trn34 statA′ Opt.eqAct-def sstatA′-def* **by** *auto*
thus *?thesis* **proof** (*elim disjE*)
assume *K1*: *?K1*
then obtain *w1′ w2′* **where** Δ: Δ ∞ *w1′ w2′ s1′ s2′ statA′ sv1 sv2 statO* **by** *auto*
have *ok′*: (*statA′* = *Diff* ⟶ *statO* = *Diff*)
using *ok K1* **unfolding** *statA′* **using** *isAO34* **by** *auto*
obtain *trv1 trv2* **where** ψ: ψ *s1′ tr1′ s2′ tr2′ statO sv1 trv1 sv2 trv2*
using *less*(*1*)[*of tr1′ tr2′*, *OF* - Δ - - - - - - *r34′ r12*, *simplified*]
using *less.prems tr1′ tr2′ ok′ A34′ isAO34 tr1′NE tr2′NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using ψ *trn34 O33′ O44′ K1 ok* **unfolding** *ψ-def tr1 tr2*
using *completedFromO-lastt less.prems*(*4*,*6*)
unfolding *Opt.completedFrom-def* **using** *tr1 tr2 tr1′NE tr2′NE* **by** *auto*
next
assume *K2*: *?K2*
then obtain *w2′ sv1′* **where**
trn1: *validTransV* (*sv1*,*sv1′*) **and**
trn13: *eqSec sv1 s1* **and**
Atrn1: ¬ *isIntV sv1* **and** *ok′*: (*statA′* = *statA* ∨ *statO* = *Diff*) **and**
Δ: Δ ∞ ∞ *w2′ s1′ s2′ statA′ sv1′ sv2 statO*
unfolding *match12-1-def* **by** *auto*
have *r1′*: *reachV sv1′* **using** *r1 trn1* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** ψ: ψ *s1′ tr1′ s2′ tr2′ statO sv1′ trv1 sv2 trv2*
using *less*(*1*)[*of tr1′ tr2′*, *OF* - Δ - - - - - - *r34′ r1′ r2*, *simplified*]
using *less.prems tr1′ tr2′ ok′ A34′ tr1′NE tr2′NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** *auto*

show *?thesis* **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - trv2*])
using *ψ O33' O44' tr1'NE tr2'NE* **unfolding** *tr1 tr2*
using *Van.validFromS-Cons trn1 ok*
using *K2 ok' Atrn1 eqSec-S-Cons trn13 trn34*
unfolding *statA' Van.completedFrom-def eqSec-def*
using *s13 tr1 tr1' tr2' trn3 trn4*
by *simp* (*smt* (*verit*, *best*) *Opt.S.Cons-unfold Simple-Transition-System.lastt-Cons*

Van.A.Cons-unfold Van.O.Cons-unfold ψ-def completedFromO-lastt f3 f34 lastt-Nil
less.prems(*4*) *status.simps*(*1*))
next
assume *K3*: *?K3*
then obtain *w1' sv2'* **where**
trn2: *validTransV* (*sv2*,*sv2'*) **and**
trn24: *eqSec sv2 s2* **and**
Atrn2: ¬ *isIntV sv2* **and** *ok'*: (*statA'* = *statA* ∨ *statO* = *Diff*) **and**
Δ: Δ ∞ *w1'* ∞ *s1' s2' statA' sv1 sv2' statO*
unfolding *match12-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *trv1 trv2* **where** *ψ*: *ψ s1' tr1' s2' tr2' statO sv1 trv1 sv2' trv2*
using *less*(*1*)[*of tr1' tr2'*, *OF* - Δ - - - - - - *r34' r1 r2'*, *simplified*]
using *less.prems tr1' tr2' ok' A34' tr1'NE tr2'NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using *ψ O33' O44' tr1'NE tr2'NE* **unfolding** *ψ-def tr1 tr2*
using *Van.validFromS-Cons trn2 ok*
using *K3 ok' Atrn2 eqSec-S-Cons trn24 trn34*
unfolding *statA' Van.completedFrom-def*
using *tr1' tr2' trn3 trn4* **by** *force*
next
assume *K4*: *?K4*
then obtain *sv1' sv2' statO'* **where** *0*:
statO' = *sstatO' statO sv1 sv2*
validTransV (*sv1*,*sv1'*)
eqSec sv1 s1
validTransV (*sv2*,*sv2'*)
eqSec sv2 s2
Van.eqAct sv1 sv2
and *ok'*: *statA'* = *Diff* ⟶ *statO'* = *Diff* **and** Δ: Δ ∞ ∞ ∞ *s1' s2' statA' sv1' sv2' statO'*
unfolding *match12-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'* **using** *r1 r2 0*
by (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *trv1 trv2* **where** *ψ*: *ψ s1' tr1' s2' tr2' statO' sv1' trv1 sv2' trv2*
using *less*(*1*)[*of tr1' tr2'*, *OF* - Δ - - - - - - *r34' r12'*, *simplified*]
using *less.prems tr1' tr2' ok' A34' tr1'NE tr2'NE* **unfolding** *tr1 tr2 Opt.completedFrom-def* **by** *auto*

show *?thesis* **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using *trn34*
using ψ *O33′ O44′ isAO34 tr1′NE tr2′NE* **unfolding** ψ*-def tr1 tr2*
using *Van.validFromS-Cons 0*
using *K4 eqSec-S-Cons*
unfolding *statA′ Van.eqAct-def Van.completedFrom-def match12-12-def sstatO′-def*
by *simp* (*smt* (*z3*) *Simple-Transition-System.lastt-Cons Van.A.Cons-unfold Van.O.Cons-unfold list.inject status.exhaust status.simps*(*1*) *tr1′ tr2′ trn3 trn4 updStat.simps*(*4*) *updStat-diff*)
qed
qed
qed
qed
qed

lemma *unwindCond-final*:
unwindCond $\Delta \Longrightarrow$ *reachO s1* $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
(*finalV sv1* $\longleftrightarrow$ *finalO s1*) $\wedge$ (*finalV sv2* $\longleftrightarrow$ *finalO s2*)
unfolding *unwindCond-def*
unfolding *proact-def match-def match1-def match1-1-def*
by *auto*

definition φ Δ *w w1 w2 w1′ w2′ statA s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO* $\equiv$
trv1 $\neq$ [] $\wedge$ *trv2* $\neq$ [] $\wedge$
(*length trv1* $>$ *Suc 0* $\vee$ *w1′* $\leq$ *w1*) $\wedge$ (*length trv2* $>$ *Suc 0* $\vee$ *w2′* $\leq$ *w2*) $\wedge$
Van.validFromS sv1 trv1 $\wedge$
Van.validFromS sv2 trv2 $\wedge$
Van.S trv1 = *Opt.S tr1* $\wedge$ *Van.S trv2* = *Opt.S tr2* $\wedge$
Van.A trv1 = *Van.A trv2* $\wedge$
(*statO* = *Eq* $\longrightarrow$ (*statOO* = *Diff* $\longleftrightarrow$ *Van.O trv1* $\neq$ *Van.O trv2*)) $\wedge$
(*statA* = *Eq* $\longrightarrow$ (*statAA* = *Diff* $\longleftrightarrow$ *Opt.O tr1* $\neq$ *Opt.O tr2*)) $\wedge$
—
(*statO* = *Diff* $\longrightarrow$ *statOO* = *Diff*) $\wedge$
(*statAA* = *Diff* $\longrightarrow$ *statOO* = *Diff*) $\wedge$
Δ *w w1′ w2′* (*lastt s1 tr1*) (*lastt s2 tr2*) *statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma φ*-final*:
assumes *unw*: *unwindCond* Δ
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *vtr14*: *Opt.validFromS s1 tr1 Opt.validFromS s2 tr2*

and φ: φ Δ *w w1 w2 w1′ w2′ statA s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO*
shows (*finalV* (*lastt sv1 trv1*) $\longleftrightarrow$ *finalO* (*lastt s1 tr1*)) $\wedge$ (*finalV* (*lastt sv2 trv2*) $\longleftrightarrow$ *finalO* (*lastt s2 tr2*))
proof−
  have *rsv12*: *Van.validFromS sv1 trv1* $\longrightarrow$ *reachV* (*lastt sv1 trv1*)
    Van.validFromS sv2 trv2 $\longrightarrow$ *reachV* (*lastt sv2 trv2*) **using** *r*
    by (*simp add*: *Van.reach-validFromS-reach lastt-def*)+
  have *rs14*: *Opt.validFromS s1 tr1* $\longrightarrow$ *reachO* (*lastt s1 tr1*)
    Opt.validFromS s2 tr2 $\longrightarrow$ *reachO* (*lastt s2 tr2*) **using** *r*
    by (*simp add*: *Opt.reach-validFromS-reach lastt-def*)+
  show *?thesis* **using** φ[*unfolded* φ*-def*] *rsv12 rs14* **using** *unw*[*unfolded unwind-Cond-def*, *rule-format*,
    of lastt s1 tr1 lastt s2 tr2 lastt sv1 trv1 lastt sv2 trv2 w w1′ w2′ statAA statOO]
  using *vtr14*(*1*) *vtr14*(*2*) **by** *auto*
qed

lemma φ*-completedFrom*: *unwindCond* Δ $\Longrightarrow$
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Opt.validFromS s1 tr1 $\Longrightarrow$ *completedFromO s1 tr1* $\Longrightarrow$
Opt.validFromS s2 tr2 $\Longrightarrow$ *completedFromO s2 tr2* $\Longrightarrow$
φ Δ *statA w w1 w2 w1′ w2′ s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO*
$\Longrightarrow$ *completedFromV sv1 trv1* $\wedge$ *completedFromV sv2 trv2*
using φ*-final*
by (*metis Van.completedFrom-def completedFromO-lastt lastt-def*)

lemma *unwindCond-ex-*φ:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: (*statA* = *Diff* $\longrightarrow$ *statO* = *Diff*)
and *v*: *Opt.validFromS s1 tr1 Opt.validFromS s2 tr2*
and *i*: *isIntO* (*lastt s1 tr1*) *isIntO* (*lastt s2 tr2*)
and *nev*: *never isIntO* (*butlast tr1*) *never isIntO* (*butlast tr2*)
shows $\exists$ *w′ w1′ w2′ trv1 trv2 statAA statOO*. φ Δ *w′ w1 w2 w1′ w2′ statA s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO*
using *assms*(*2*−)
proof(*induction length tr1* + *length tr2 w*
  arbitrary: *w1 w2 s1 s2 statA sv1 sv2 statO tr1 tr2 rule*: *less2-induct′*)
  case (*less w tr1 tr2 w1 w2 s1 s2 statA sv1 sv2 statO*)
  note *ok* = ‹*statA* = *Diff* $\longrightarrow$ *statO* = *Diff*›
  note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›
  note *r34* = *less*(*4*,*5*) **note** *r12* = *less*(*6*,*7*)
  note *r* = *r34 r12*
  note *r3* = *r34*(*1*) **note** *r4* = *r34*(*2*) **note** *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)
  note *nev34* = *less*(*13*,*14*)
  note *nev3* = *nev34*(*1*) **note** *nev4* = *nev34*(*2*)

  have *i34*: *statA* = *Eq* $\longrightarrow$ *isIntO s1* = *isIntO s2*

and *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

note *is1* = ‹*isIntO* (*lastt s1 tr1*)›
note *is2* = ‹*isIntO* (*lastt s2 tr2*)›
note *vtr1* = ‹*Opt.validFromS s1 tr1*›
note *vtr2* = ‹*Opt.validFromS s2 tr2*›

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1'*
where *0*: *validTransV* (*sv1*,*sv1'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1' sv2 statO*
unfolding *move-1-def* **by** *auto*
have *r1'*: *reachV sv1'* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w' w1' w2' trv1 trv2 statAA statOO* **where** *φ*: *φ* Δ *w' w1 w2 w1' w2' statA s1 tr1 s2 tr2 statAA statO sv1' trv1 sv2 trv2 statOO*
using *less*(*2*)[*OF v*, *of tr1 tr2 w1 w2 s1 s2 statA sv1' sv2 statO*, *simplified*, *OF* Δ *r34 r1' r2 ok*]
using *is1 is2 nev3 nev4 vtr1 vtr2* **by** *blast*
show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*])
apply(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - trv2*])
using *φ ok 0 sv1* **unfolding** *φ-def* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv2'*
where *0*: *validTransV* (*sv2*,*sv2'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1 sv2' statO*
unfolding *move-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w' w1' w2' trv1 trv2 statAA statOO* **where** *φ*: *φ* Δ *w' w1 w2 w1' w2' statA s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2' trv2 statOO*
using *less*(*2*)[*OF v*, *of tr1 tr2 w1 w2 s1 s2 statA sv1 sv2' statO*, *simplified*, *OF* Δ *r34 r1 r2' ok*]
using *is1 is2 nev3 nev4 vtr1 vtr2* **by** *blast*
show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using *φ ok 0 sv2* **unfolding** *φ-def* **by** *auto*

next
assume *sv12*: ¬ *isSecV sv1* ¬ *isSecV sv2 Van.eqAct sv1 sv2*
and *move-12* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1′ sv2′ statO′*
where *0*: *statO′* = *sstatO′ statO sv1 sv2*
validTransV (*sv1*,*sv1′*) ¬ *isSecV sv1*
validTransV (*sv2*,*sv2′*) ¬ *isSecV sv2*
Van.eqAct sv1 sv2
and Δ: Δ *v w1 w2 s1 s2 statA sv1′ sv2′ statO′*
unfolding *move-12-def* **by** *auto*
have *r12′*: *reachV sv1′ reachV sv2′* **using** *r1 r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
have *ok′*: *statA* = *Diff* ⟶ *statO′* = *Diff*
using *ok 0* **unfolding** *sstatO′-def* **by** (*cases statO*, *auto*)
obtain *w′ w1′ w2′ trv1 trv2 statAA statOO* **where** *φ*: *φ* Δ *w′ w1 w2 w1′ w2′ statA s1 tr1 s2 tr2 statAA statO′ sv1′ trv1 sv2′ trv2 statOO*
using *less*(*2*)[*OF v*, *of tr1 tr2 w1 w2 s1 s2 statA sv1′ sv2′ statO′*, *simplified*, *OF* Δ *r34 r12′ ok′*]
using *is1 is2 nev3 nev4 vtr1 vtr2* **by** *blast*
show *?thesis* **apply**(*rule exI*[*of - w′*]) **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
apply(*rule exI*[*of - statAA*]) **apply**(*rule exI*[*of - statOO*])
using *φ ok′ 0 sv12 nev* **unfolding** *φ-def sstatO′-def*
by *simp* (*smt* (*verit*, *ccfv-SIG*) *Statewise-Attacker-Mod.eqAct-def*
Van.A.Cons-unfold Van.O.Cons-unfold Van.Statewise-Attacker-Mod-axioms
Van.validFromS-Cons list.inject updStat.simps(*1*) *updStat.simps*(*4*))
qed
next
assume *m*: *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
define *statA′* **where** *statA′*: *statA′* = *sstatA′ statA s1 s2*
show *?thesis*
proof(*cases length tr1* ≤ *Suc 0*)
case *True*
hence *tr1e*: *tr1* = [] ∨ *tr1* = [*s1*]
by (*metis Opt.validFromS-singl-iff Suc-length-conv le-Suc-eq le-zero-eq length-0-conv vtr1*)
hence *Opt.A tr1* = [] **by** (*simp add*: *True*)
hence *Opt.A tr2* = [] **using** *Opt.A.eq-Nil-iff nev4* **by** *blast*
show *?thesis*
proof(*cases length tr2* ≤ *Suc 0*)
case *True*
hence *tr2e*: *tr2* = [] ∨ *tr2* = [*s2*]
by (*metis Opt.validFromS-def Suc-length-conv le-Suc-eq le-zero-eq length-0-conv list.sel*(*1*) *vtr2*)
show *?thesis* **apply**(*rule exI*[*of - w*]) **apply**(*rule exI*[*of - w1*]) **apply**(*rule exI*[*of - w2*])
apply(*rule exI*[*of - *[*sv1*]], *rule exI*[*of - *[*sv2*]], *rule exI*[*of - statA*], *rule exI*[*of - statO*])

using *tr1e tr2e*
using *f34* Δ **apply** (*clarsimp simp*: *φ-def lastt-def*)
apply(*cases statA*, *simp-all*)
apply (*metis Opt.O.simps*(*4*) *Opt.S.simps*(*4*) *last-ConsL*)
by (*metis Opt.S.simps*(*4*) *last.simps ok*)
next
case *False*
then obtain *s24 tr2′* **where** *tr2*: *tr2* = *s24* # *tr2′* **and** *tr2′NE*: *tr2′* ≠ []
by (*cases tr2*, *auto*)
have *s24*[*simp*]: *s24* = *s2* **using** ‹*Opt.validFromS s2 tr2*›
by (*simp add*: *Opt.validFromS-Cons-iff tr2*)
obtain *s2′* **where**
trn4: *validTransO* (*s2*,*s2′*) ∨ (*s2* = *s2′* ∧ *tr2′* = []) **and**
tr2′: *Opt.validFromS s2′ tr2′* **using** ‹*Opt.validFromS s2 tr2*›
unfolding *tr2 s24* **using** *Opt.validFromS-Cons-iff* **by** *auto*
have *r4′*: *reachO s2′*
using *r4 trn4* **by** (*metis Opt.reach.Step fst-conv snd-conv*)+
have *nev4′*: *never isIntO* (*butlast tr2′*)
by (*metis Opt.O.Nil-iff Opt.O.eq-Nil-iff nev4 tr2*)
have *isAO4*: ¬ *isIntO s2*
using ‹*Opt.A tr2* = []› *tr2 tr2′NE* **by** *auto*
have *O44′*: *Opt.O tr2* = *Opt.O tr2′ Opt.A tr2* = *Opt.A tr2′*
using *isAO4* ‹*Opt.A tr2* = []› *tr2* **by** *auto*
have *m*: *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have (∃ *w1′*<*w1*. ∃ *w2′*<*w2*. ¬ *isSecO s2* ∧ Δ ∞ *w1′ w2′ s1 s2′ statA sv1 sv2 statO*) ∨
(∃ *w1′*<*w1*. *eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧ *match2-1* Δ *w1′* ∞ *s1 s2 s2′ statA sv1 sv2 statO*) ∨
(¬ *isSecV sv1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧ *match2-12* Δ ∞ ∞ *s1 s2 s2′ statA sv1 sv2 statO*)
using *isAO4 trn4 ok tr2′NE*
using *m*[*unfolded match2-def*, *rule-format*, *of s2′*] **by** *auto*
thus *?thesis*
proof *safe*
fix *w1″ w2″* **assume** *w12′*: *w1″* < *w1 w2″* < *w2*
assume ¬ *isSecO s2* **and** Δ: Δ ∞ *w1″ w2″ s1 s2′ statA sv1 sv2 statO*
hence *S4*: *Opt.S tr2′* = *Opt.S tr2* **unfolding** *tr2* **by** *auto*
obtain *w′ w1′ w2′ trv1 trv2 statAA statOO* **where** *φ*: *φ* Δ *w′ w1″ w2″ w1′ w2′ statA s1 tr1 s2′ tr2′ statAA statO sv1 trv1 sv2 trv2 statOO*
using *less*(*1*)[*of tr1 tr2′*, *OF* - Δ *r3 r4′* - - - - - - - *nev3 nev4′*, *unfolded tr2*, *simplified*]
using *is1 is2 vtr1 vtr2 tr2′ ok tr2′NE trn4 r1 r2 tr2* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *w′*]) **apply**(*rule exI*[*of* - *w1′*]) **apply**(*rule exI*[*of* - *w2′*]) **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using *φ O44′ S4 tr2 tr2′NE trn4 tr2′ w12′* **unfolding** *φ-def* **by** *auto*
next
fix *w1″* **assume** *w1′*: *w1″* < *w1*
assume *trn24*: *eqSec sv2 s2* **and**

Atrn2: ¬ *isIntV sv2* **and** *match2-1* Δ *w1''* ∞ *s1 s2 s2' statA sv1 sv2 statO*
then obtain *sv2'* **where** *trn2*: *validTransV* (*sv2*,*sv2'*) **and**
Δ: Δ ∞ *w1''* ∞ *s1 s2' statA sv1 sv2' statO*
unfolding *match2-1-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w' w1' w2' trv1 trv2 statAA statOO* **where** φ: φ Δ *w' w1''* ∞ *w1' w2' statA s1 tr1 s2' tr2' statAA statO sv1 trv1 sv2' trv2 statOO*
using *less*(*1*)[*of tr1 tr2'*, *OF* - Δ *r3 r4' r1 r2'* - - - - - *nev3 nev4'*, *unfolded tr2*, *simplified*]
using *is1 is2 tr2' tr2 vtr1 ok tr2'NE trn4* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *w'*]) **apply**(*rule exI*[*of* - *w1'*]) **apply**(*rule exI*[*of* - *w2'*]) **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using φ *tr2'NE*
using *Van.validFromS-Cons trn2*
using *isAO4 ok Atrn2 eqSec-S-Cons trn24 tr2' trn4 w1'*
unfolding *φ-def tr2 s24*
by *auto*
next
assume *sv1*: ¬ *isSecV sv1* **and** *trn24*: *eqSec sv2 s2* **and**
Atrn12: *Van.eqAct sv1 sv2* **and** *match2-12* Δ ∞ ∞ *s1 s2 s2' statA sv1 sv2 statO*
then obtain *sv1' sv2' statO'* **where**
statO': *statO'* = *sstatO' statO sv1 sv2* **and**
trn1: *validTransV* (*sv1*,*sv1'*) **and**
trn2: *validTransV* (*sv2*,*sv2'*) **and**
Δ: Δ ∞ ∞ ∞ *s1 s2' statA sv1' sv2' statO'*
unfolding *match2-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'*
using *r1 trn1 r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *w' w1' w2' trv1 trv2 statAA statOO* **where** φ: φ Δ *w'* ∞ ∞ *w1' w2' statA s1 tr1 s2' tr2' statAA statO' sv1' trv1 sv2' trv2 statOO*
using *less*(*1*)[*of tr1 tr2'*, *OF* - Δ *r3 r4' r12'* - - - - - *nev3 nev4'*, *simplified*]
using *is1 is2 vtr1 tr2 tr2' ok tr2'NE trn4* **unfolding** *tr2 statO' sstatO'-def* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *w'*]) **apply**(*rule exI*[*of* - *w1'*]) **apply**(*rule exI*[*of* - *w2'*]) **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using φ *O44' tr2'NE sv1*
using *Van.validFromS-Cons trn1 trn2*
using *isAO4 ok Atrn12 eqSec-S-Cons trn24 tr2' trn4*
unfolding *φ-def tr2 Van.completedFrom-def Van.eqAct-def statO' sstatO'-def*

by *simp* (*smt* (*verit*, *ccfv-threshold*) *Van.A.Cons-unfold i34 is1 last-ConsL lastt-def status.exhaust tr1e updStat.simps*(*2*))
qed
qed
next
case *False*
then obtain *s13 tr1'* **where** *tr1*: *tr1* = *s13* # *tr1'* **and** *tr1'NE*: *tr1'* ≠ []

by (*cases tr1*, *auto*)
have *s13*[*simp*]: *s13* = *s1* **using** ‹*Opt.validFromS s1 tr1*›
by (*simp add*: *Opt.validFromS-Cons-iff tr1*)
obtain *s1'* **where**
trn3: *validTransO* (*s1*,*s1'*) **and**
tr1': *Opt.validFromS s1' tr1'* **using** ‹*Opt.validFromS s1 tr1*›
unfolding *tr1 s13* **by** (*metis tr1'NE Simple-Transition-System.validFromS-Cons-iff*)
have *r3'*: *reachO s1'* **using** *r3 trn3* **by** (*metis Opt.reach.Step fst-conv snd-conv*)
have *f3*: ¬ *finalO s1* **using** *Opt.final-def trn3* **by** *blast*
hence *f4*: ¬ *finalO s2* **using** *f34* **by** *blast*
have *nev3'*: *never isIntO* (*butlast tr1'*)
using *nev3 tr1 tr1'NE* **by** *auto*
have *isAO3*: ¬ *isIntO s1* **using** *less.prems*(*11*) *tr1 tr1'NE* **by** *auto*
have *O33'*: *Opt.O tr1* = *Opt.O tr1' Opt.A tr1* = *Opt.A tr1'*
using *isAO3* **unfolding** *tr1* **by** *auto*
have *m*: *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have (∃ *w1'*<*w1*. ∃ *w2'*<*w2*. ¬ *isSecO s1* ∧ Δ ∞ *w1' w2' s1' s2 statA sv1 sv2 statO*) ∨
(∃ *w2'*<*w2*. *eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧ *match1-1* Δ ∞ *w2' s1 s1' s2 statA sv1 sv2 statO*) ∨
(*eqSec sv1 s1* ∧ ¬ *isSecV sv2* ∧ *Van.eqAct sv1 sv2* ∧ *match1-12* Δ ∞ ∞ *s1 s1' s2 statA sv1 sv2 statO*)
using *m isAO3 trn3 ok* **unfolding** *match1-def* **by** *auto*
thus *?thesis*
proof *safe*
fix *w1'' w2''* **assume** *w12'*: *w1''* < *w1 w2''* < *w2*
assume ¬ *isSecO s1* **and** Δ: Δ ∞ *w1'' w2'' s1' s2 statA sv1 sv2 statO*
hence *S3*: *Opt.S tr1'* = *Opt.S tr1* **unfolding** *tr1* **by** *auto*
obtain *w' w1' w2' trv1 trv2 statAA statOO* **where** *φ*: *φ* Δ *w' w1'' w2'' w1' w2' statA s1' tr1' s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO*
using *less*(*1*)[*of tr1' tr2*, *OF* - Δ *r3' r4 r12*, *unfolded O33'*, *simplified*]
using *is1 is2 tr1' ok f3 f4 tr1'NE trn3 O33'*(*1*) *nev3' nev4 vtr2* **unfolding** *tr1* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *w'*]) **apply**(*rule exI*[*of* - *w1'*]) **apply**(*rule exI*[*of* - *w2'*]) **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using *φ O33' S3 tr1 tr1'NE tr1' trn3 w12'* **unfolding** *φ-def* **by** *auto*
next
fix *w2''* **assume** *w2'*: *w2''* < *w2*
assume *trn13*: *eqSec sv1 s1* **and**
Atrn1: ¬ *isIntV sv1* **and** *match1-1* Δ ∞ *w2'' s1 s1' s2 statA sv1 sv2 statO*
then obtain *sv1'* **where**
trn1: *validTransV* (*sv1*,*sv1'*) **and**
Δ: Δ ∞ ∞ *w2'' s1' s2 statA sv1' sv2 statO*
unfolding *match1-1-def* **by** *auto*
have *r1'*: *reachV sv1'* **using** *r1 trn1* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w' w1' w2' trv1 trv2 statAA statOO* **where** *φ*: *φ* Δ *w'* ∞ *w2'' w1' w2' statA s1' tr1' s2 tr2 statAA statO sv1' trv1 sv2 trv2 statOO*

using *less*(*1*)[*of tr1′ tr2*, *OF* - Δ *r3′ r4 r1′ r2*, *unfolded O33′*, *simplified*]
using *is1 is2 tr1 nev3′ nev4 vtr1 vtr2 tr1′ ok f3 f4 tr1′NE trn3 O33′*(*1*)
unfolding *tr1* **by** *auto*
show *?thesis* **apply**(*rule exI*[*of* - *w′*]) **apply**(*rule exI*[*of* - *w1′*]) **apply**(*rule exI*[*of* - *w2′*]) **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using φ *O33′* **unfolding** *φ-def tr1 Van.completedFrom-def*
using *Van.validFromS-Cons trn1 tr1′NE tr1′ trn3*
using *isAO3 ok Atrn1 eqSec-S-Cons trn13 w2′*
by *auto*
next
assume *sv2*: ¬ *isSecV sv2* **and** *trn13*: *eqSec sv1 s1* **and**
Atrn12: *Van.eqAct sv1 sv2* **and** *match1-12* Δ ∞ ∞ *s1 s1′ s2 statA sv1 sv2 statO*
then obtain *sv1′ sv2′ statO′* **where**
statO′: *statO′* = *sstatO′ statO sv1 sv2* **and**
trn1: *validTransV* (*sv1*,*sv1′*) **and**
trn2: *validTransV* (*sv2*,*sv2′*) **and**
Δ: Δ ∞ ∞ ∞ *s1′ s2 statA sv1′ sv2′ statO′*
unfolding *match1-12-def* **by** *auto*
have *r12′*: *reachV sv1′ reachV sv2′*
using *r1 trn1 r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *w′ w1′ w2′ trv1 trv2 statAA statOO* **where** φ: φ Δ *w′* ∞ ∞ *w1′ w2′ statA s1′ tr1′ s2 tr2 statAA statO′ sv1′ trv1 sv2′ trv2 statOO*
using *less*(*1*)[*of tr1′ tr2*, *OF* - Δ *r3′ r4 r12′*, *unfolded O33′*, *simplified*]
using *less.prems tr1′ ok f3 f4 tr1′NE trn3 O33′*(*1*) **unfolding** *tr1 statO′ sstatO′-def* **by** *auto*

have *trv1NE*: *trv1* ≠ [] **and** *trv2NE*: *trv2* ≠ [] **using** φ **unfolding** *φ-def* **by** *auto*
have [*simp*]: *Van.O* (*sv1* # *trv1*) = *Van.O* (*sv2* # *trv2*) ⟷ (*isIntV sv1* ⟶ *getObsV sv1* = *getObsV sv2*) ∧ *Van.O trv1* = *Van.O trv2*
using *Atrn12 trv1NE trv2NE* **unfolding** *Van.O.map-filter Van.eqAct-def* **by** *simp*
show *?thesis* **apply**(*rule exI*[*of* - *w′*]) **apply**(*rule exI*[*of* - *w1′*]) **apply**(*rule exI*[*of* - *w2′*]) **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using φ *O33′ tr1′NE sv2*
using *Van.validFromS-Cons trn1 trn2*
using *isAO3 ok Atrn12 eqSec-S-Cons trn13 f3 f34 s13 tr1′ trn3*
unfolding *φ-def tr1 Van.completedFrom-def Van.eqAct-def statO′ sstatO′-def*
apply *clarsimp*
by (*smt* (*verit*, *ccfv-SIG*) *Van.A.Cons-unfold updStat.simps*(*1*) *updStat.simps*(*2*) *updStat.simps*(*4*))
qed
qed
qed
qed

definition *φa* Δ *w w1 w2 w1′ w2′ statA s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO* ≡

trv1 ≠ [] ∧ *trv2* ≠ [] ∧
(*length trv1* > *Suc 0* ∨ *w1'* < *w1*) ∧ (*length trv2* > *Suc 0* ∨ *w2'* < *w2*) ∧
Van.validFromS sv1 trv1 ∧
Van.validFromS sv2 trv2 ∧
Van.S trv1 = *Opt.S tr1* ∧ *Van.S trv2* = *Opt.S tr2* ∧
Van.A trv1 = *Van.A trv2* ∧
(*statO* = *Eq* ⟶ (*statOO* = *Diff* ⟷ *Van.O trv1* ≠ *Van.O trv2*)) ∧
(*statA* = *Eq* ⟶ (*statAA* = *Diff* ⟷ *Opt.O tr1* ≠ *Opt.O tr2*)) ∧
—
(*statO* = *Diff* ⟶ *statOO* = *Diff*) ∧
(*statAA* = *Diff* ⟶ *statOO* = *Diff*) ∧
Δ *w w1' w2'* (*lastt s1 tr1*) (*lastt s2 tr2*) *statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *unwindCond-ex-φa-getActO*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r34*: *reachO s1 reachO s2* **and** *r12*: *reachV sv1 reachV sv2*
and *stat*: (*statA* = *Diff* ⟶ *statO* = *Diff*)
and *v*: *validTransO* (*s1*, *s1'*) *validTransO* (*s2*, *s2'*)
and *i34*: *isIntO s1 isIntO s2 getActO s1* = *getActO s2*
shows ∃ *w1' w2' trv1 trv2 statOO*.
φa Δ ∞ *w1 w2 w1' w2' statA s1* [*s1*, *s1'*] *s2* [*s2*, *s2'*] (*sstatA' statA s1 s2*) *statO sv1 trv1 sv2 trv2 statOO*
using Δ *r12 stat*
proof(*induction w arbitrary*: *w1 w2 sv1 sv2 statO rule*: *less-induct*)
case (*less w w1 w2 sv1 sv2 statO*)
note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›
note *r12* = *less.prems*(*2*,*3*)
note *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)
note *r* = *r34 r12*
note *stat* = ‹*statA* = *Diff* ⟶ *statO* = *Diff*›

have *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1'*
where *0*: *validTransV* (*sv1*,*sv1'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1' sv2 statO*

unfolding *move-1-def* **by** *auto*
have *r1'*: *reachV sv1'* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1' w2' trv1 trv2 statOO* **where**
φ: *φa Δ ∞ w1 w2 w1' w2' statA s1 [s1, s1'] s2 [s2, s2'] (sstatA' statA s1 s2) statO sv1' trv1 sv2 trv2 statOO*
using *less(1)[OF v Δ r1' r2 stat]* **by** *auto*
show *?thesis* **apply**(*rule exI[of - w1']*) **apply**(*rule exI[of - w2']*) **apply**(*rule exI[of - sv1 # trv1]*) **apply**(*rule exI[of - trv2]*)
using *φ 0 sv1* **unfolding** *φa-def* **apply** *simp*
by (*metis Van.validFromS-Cons*)
next
assume *sv2*: *¬ isSecV sv2 ¬ isIntV sv2* **and** *move-2 Δ v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv2'*
where *0*: *validTransV (sv2,sv2')*
and Δ: *Δ v w1 w2 s1 s2 statA sv1 sv2' statO*
unfolding *move-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1' w2' trv1 trv2 statOO* **where**
φ: *φa Δ ∞ w1 w2 w1' w2' statA s1 [s1, s1'] s2 [s2, s2'] (sstatA' statA s1 s2) statO sv1 trv1 sv2' trv2 statOO*
using *less(1)[OF v Δ r1 r2' stat]* **by** *auto*
show *?thesis* **apply**(*rule exI[of - w1']*) **apply**(*rule exI[of - w2']*) **apply**(*rule exI[of - trv1]*) **apply**(*rule exI[of - sv2 # trv2]*)
using *φ 0 sv2* **unfolding** *φa-def* **apply** *simp* **by** (*metis Van.validFromS-Cons*)
next
assume *sv12*: *¬ isSecV sv1 ¬ isSecV sv2 Van.eqAct sv1 sv2*
and *move-12 Δ v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1' sv2' statO'*
where *0*: *statO' = sstatO' statO sv1 sv2*
validTransV (sv1,sv1') ¬ isSecV sv1
validTransV (sv2,sv2') ¬ isSecV sv2
Van.eqAct sv1 sv2
and Δ: *Δ v w1 w2 s1 s2 statA sv1' sv2' statO'*
unfolding *move-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'* **using** *r1 r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
have *stat'*: *statA = Diff ⟶ statO' = Diff*
using *stat 0* **unfolding** *sstatO'-def* **by** (*cases statO*, *auto*)
obtain *w1' w2' trv1 trv2 statOO* **where**
φ: *φa Δ ∞ w1 w2 w1' w2' statA s1 [s1, s1'] s2 [s2, s2'] (sstatA' statA s1 s2) statO' sv1' trv1 sv2' trv2 statOO*
using *less(1)[OF v Δ r12' stat']* **unfolding** *φa-def* **apply** *simp* **by** *metis*
show *?thesis* **apply**(*rule exI[of - w1']*) **apply**(*rule exI[of - w2']*) **apply**(*rule exI[of - sv1 # trv1]*) **apply**(*rule exI[of - sv2 # trv2]*)
using *φ 0* **unfolding** *φa-def sstatO'-def* **apply** *clarsimp* **apply**(*intro conjI*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*metis Van.A.Cons-unfold Van.eqAct-def*)

subgoal apply(*rule exI*[*of - statOO*]) **apply** *simp*
by (*smt* (*verit*, *ccfv-threshold*) *Van.O.Cons-unfold Van.eqAct-def*
list.inject updStat.simps(*1*) *updStat.simps*(*3*)) **.**
qed
next
assume *m*: *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
define *statA'* **where** *statA'*: *statA'* = *sstatA' statA s1 s2*
have *m*: *match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have ($\exists$ *w1' w2'*. *w1'* < *w1* $\wedge$ *w2'*< *w2* $\wedge$ $\neg$ *isSecO s1* $\wedge$ $\neg$ *isSecO s2* $\wedge$ (*statA* = *statA'* $\vee$ *statO* = *Diff*) $\wedge$
Δ ∞ *w1' w2' s1' s2' statA' sv1 sv2 statO*)
$\vee$
($\exists$ *w2'*< *w2*. $\neg$ *isSecO s2* $\wedge$
eqSec sv1 s1 $\wedge$ $\neg$ *isIntV sv1* $\wedge$ (*statA* = *statA'* $\vee$ *statO* = *Diff*) $\wedge$
match12-1 Δ ∞ *w2' s1' s2' statA' sv1 sv2 statO*)
$\vee$
($\exists$ *w1'*<*w1*. $\neg$ *isSecO s1* $\wedge$
eqSec sv2 s2 $\wedge$ $\neg$ *isIntV sv2* $\wedge$ (*statA* = *statA'* $\vee$ *statO* = *Diff*) $\wedge$
match12-2 Δ *w1'* ∞ *s1' s2' statA' sv1 sv2 statO*)
$\vee$
(*eqSec sv1 s1* $\wedge$ *eqSec sv2 s2* $\wedge$ *Van.eqAct sv1 sv2* $\wedge$
match12-12 Δ ∞ ∞ *s1' s2' statA' sv1 sv2 statO*)
using *m* **unfolding** *match12-def*
by (*simp add*: *Opt.eqAct-def i34*(*1*) *i34*(*2*) *i34*(*3*) *statA' v*(*1*) *v*(*2*))
thus *?thesis*
apply(*elim disjE exE*)
subgoal for *w1' w2'* **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of -* [*sv1*]]) **apply**(*rule exI*[*of -* [*sv2*]])
apply(*rule exI*[*of - statO*])
using *stat* **unfolding** *φa-def statA'*
by (*auto simp add*: *i34*(*1*) *i34*(*2*) *sstatA'-def lastt-def*)
subgoal for *w2'* **apply**(*rule exI*[*of -* ∞]) **apply**(*rule exI*[*of - w2'*])
unfolding *match12-1-def* **apply**(*elim conjE exE*) **subgoal for** *sv1'*
apply(*rule exI*[*of -* [*sv1*,*sv1'*]]) **apply**(*rule exI*[*of -* [*sv2*]])
apply(*rule exI*[*of - statO*])
using *stat* **unfolding** *φa-def statA'*
by (*auto simp add*: *i34*(*1*) *i34*(*2*) *sstatA'-def lastt-def*) **.**
subgoal for *w1'* **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of -* ∞])
unfolding *match12-2-def* **apply**(*elim conjE exE*) **subgoal for** *sv2'*
apply(*rule exI*[*of -* [*sv1*]]) **apply**(*rule exI*[*of -* [*sv2*,*sv2'*]])
apply(*rule exI*[*of - statO*])
using *stat* **unfolding** *φa-def statA'*
by (*auto simp add*: *i34*(*1*) *i34*(*2*) *sstatA'-def lastt-def*) **.**
subgoal unfolding *match12-12-def* **apply**(*elim conjE exE*) **subgoal for** *sv1' sv2'*
apply(*rule exI*[*of -* ∞]) **apply**(*rule exI*[*of -* ∞])
apply(*rule exI*[*of -* [*sv1*,*sv1'*]]) **apply**(*rule exI*[*of -* [*sv2*,*sv2'*]])
apply(*rule exI*[*of - sstatO' statO sv1 sv2*])

using *stat* **unfolding** *φa-def statA′*
by (*auto simp add*: *i34 i34 sstatA′-def sstatO′-def lastt-def Van.eqAct-def*) **. .**
qed
qed

lemma *unwindCond-ex-φa′-aux*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: (*statA* = *Diff* ⟶ *statO* = *Diff*)
and *tr14NE*: *tr1* ≠ [] *tr2* ≠ []
and *v3′*: *Opt.validFromS s1* (*tr1* ## *s1′*) **and** *v4′*: *Opt.validFromS s2* (*tr2* ## *s2′*)
and *i*: *isIntO* (*lastt s1 tr1*) *isIntO* (*lastt s2 tr2*)
and *A34*: *getActO* (*lastt s1 tr1*) = *getActO* (*lastt s2 tr2*)
and *nev*: *never isIntO* (*butlast tr1*) *never isIntO* (*butlast tr2*)
shows ∃ *w1′ w2′ trv1′ trv2′ statAA′ statOO′*.
φa Δ ∞ *w1 w2 w1′ w2′ statA s1* (*tr1* ## *s1′*) *s2* (*tr2* ## *s2′*) *statAA′ statO sv1 trv1′ sv2 trv2′ statOO′*
proof−
have *v3*: *Opt.validFromS s1 tr1* **and** *s13′*: *validTransO* (*lastt s1 tr1*,*s1′*)
apply (*metis v3′ Opt.validFromS-def Opt.validS-append1 Nil-is-append-conv hd-append2*)
by (*metis Opt.validFromS-def Opt.validS-validTrans append-is-Nil-conv lastt-def list.distinct*(*1*) *list.sel*(*1*) *tr14NE*(*1*) *v3′*)
have *v4*: *Opt.validFromS s2 tr2* **and** *s24′*: *validTransO* (*lastt s2 tr2*,*s2′*)
apply (*metis v4′ Opt.validFromS-def Opt.validS-append1 Nil-is-append-conv hd-append2*)
by (*metis Opt.validFromS-def Opt.validS-validTrans append-is-Nil-conv lastt-def list.sel*(*1*) *list.simps*(*3*) *tr14NE*(*2*) *v4′*)

obtain *ww ww1 ww2 trv1 trv2 statAA statOO* **where** *φ*: *φ* Δ *ww w1 w2 ww1 ww2 statA s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO*
using *unwindCond-ex-φ*[*OF unwind* Δ *r stat v3 v4 i nev*] **by** *auto*

have *trv12NE*: *trv1* ≠ [] *trv2* ≠ [] **using** *φ* **unfolding** *φ-def* **by** *auto*

define *ss1 ss2 ssv1 ssv2* **where** *ss1*: *ss1* ≡ *lastt s1 tr1* **and** *ss2*: *ss2* ≡ *lastt s2 tr2*
and *ssv1*: *ssv1* ≡ *lastt sv1 trv1* **and** *ssv2*: *ssv2* ≡ *lastt sv2 trv2*

have *ss1l*: *ss1* = *last tr1* **by** (*simp add*: *lastt-def ss1 tr14NE*(*1*))
have *tr1l*: *tr1* = *butlast tr1* @ [*ss1*] **by** (*simp add*: *ss1l tr14NE*(*1*))
have *ss2l*: *ss2* = *last tr2* **by** (*simp add*: *lastt-def ss2 tr14NE*(*2*))
have *tr2l*: *tr2* = *butlast tr2* @ [*ss2*] **by** (*simp add*: *ss2l tr14NE*(*2*))
have *ssv1l*: *ssv1* = *last trv1* **using** *φ* **unfolding** *φ-def* **by** (*metis lastt-def ssv1*)
have *trv1l*: *trv1* = *butlast trv1* @ [*ssv1*] **by** (*simp add*: *ssv1l trv12NE*(*1*))
have *ssv2l*: *ssv2* = *last trv2* **using** *φ* **unfolding** *φ-def* **by** (*metis lastt-def ssv2*)
have *trv2l*: *trv2* = *butlast trv2* @ [*ssv2*] **by** (*simp add*: *ssv2l trv12NE*(*2*))

have *iss14*[*simp*]: *isIntO ss1 isIntO ss2* **using** *i* **unfolding** *ss1 ss2* **by** *auto*

```
have giss14[simp]: getActO ss1 = getActO ss2
  using A34 ss1 ss2 by fastforce

have [simp]: Opt.O (tr1 ## s1') = Opt.O tr1 ## getObsO ss1
by (metis Opt.O-def ‹isIntO ss1› holds-filtermap-RCons snoc-eq-iff-butlast tr1l)
have [simp]: Opt.O (tr2 ## s2') = Opt.O tr2 ## getObsO ss2
by (metis Opt.O-def ‹isIntO ss2› holds-filtermap-RCons snoc-eq-iff-butlast tr2l)

have [simp]: Opt.A (tr1 ## s1') = Opt.A tr1 ## getActO ss1
by (metis Opt.A-def ‹isIntO ss1› holds-filtermap-RCons snoc-eq-iff-butlast tr1l)
have [simp]: Opt.A (tr2 ## s2') = Opt.A tr2 ## getActO ss2
by (metis Opt.A-def ‹isIntO ss2› holds-filtermap-RCons snoc-eq-iff-butlast tr2l)
have [simp]: Opt.A (tr1 ## s1') = Opt.A (tr2 ## s2') ⟷ Opt.A tr1 = Opt.A
tr2 by simp

have rss: reachO ss1 reachO ss2 reachV ssv1 reachV ssv2
using Opt.reach-validFromS-reach r ss1l tr14NE(1) v3 apply blast
using Opt.reach-validFromS-reach r(2) ss2l tr14NE(2) v4 apply blast
using Van.reach-validFromS-reach φ-def φ r(3) ssv1l
apply (smt (verit, del-insts))
using Van.reach-validFromS-reach φ-def φ r(4) ssv2l
apply (smt (verit, del-insts)) .

have stat: statAA = Diff ⟶ statOO = Diff
and Δ: Δ ww ww1 ww2 ss1 ss2 statAA ssv1 ssv2 statOO
using φ unfolding φ-def ss1[symmetric] ss2[symmetric] ssv1[symmetric] ssv2[symmetric]
by auto

note vs13 = s13'[unfolded ss1[symmetric]] note vs24 = s24'[unfolded ss2[symmetric]]
have ∃ w1' w2' trv1' trv2' statA' statO'.
φa Δ ∞ ww1 ww2 w1' w2' statAA ss1 [ss1,s1'] ss2 [ss2,s2'] (sstatA' statAA ss1
ss2) statOO ssv1 trv1' ssv2 trv2' statO'
using unwindCond-ex-φa-getActO[OF unwind Δ rss stat vs13 vs24 iss14 giss14]
by blast

then obtain w1' w2' trv1' trv2' statA' statO' where
φ1: φa Δ ∞ ww1 ww2 w1' w2' statAA ss1 [ss1,s1'] ss2 [ss2,s2'] statA' statOO
ssv1 trv1' ssv2 trv2' statO' by auto

have trv12'NE: trv1' ≠ [] trv2' ≠ [] using φ1 unfolding φa-def by auto

have [simp]: Van.O (butlast trv1 @ trv1') = Van.O trv1 @ Van.O trv1'
using trv12'NE unfolding φ-def Van.O.map-filter Opt.O.map-filter apply(subst
butlast-append) by simp

have [simp]: Van.O (butlast trv2 @ trv2') = Van.O trv2 @ Van.O trv2'
using trv12'NE unfolding φ-def Van.O.map-filter Opt.O.map-filter apply(subst
butlast-append) by simp
```

have *Van.A trv1′* = *Van.A trv2′* **using** *φ1* **unfolding** *φa-def* **by** *auto*
moreover have *length* (*Van.O trv1′*) = *length* (*Van.A trv1′*) ∧ *length* (*Van.O trv2′*) = *length* (*Van.A trv2′*)
unfolding *Van.A.map-filter Van.O.map-filter* **by** *auto*
ultimately have *length* (*Van.O trv1′*) = *length* (*Van.O trv2′*) **by** *auto*
hence [*simp*]: *Van.O trv1* @ *Van.O trv1′* = *Van.O trv2* @ *Van.O trv2′* ⟷
Van.O trv1 = *Van.O trv2* ∧ *Van.O trv1′* = *Van.O trv2′* **by** *auto*

have *len*: *trv1* ≠ [] ∧ *trv2* ≠ [] ∧ *trv1′* ≠ [] ∧ *trv2′* ≠ [] ∧
(*Suc 0* < *length trv1* ∨ *ww1* ≤ *w1*) ∧
(*Suc 0* < *length trv1′* ∨ *w1′* < *ww1*) ∧
(*Suc 0* < *length trv2* ∨ *ww2* ≤ *w2*) ∧
(*Suc 0* < *length trv2′* ∨ *w2′* < *ww2*)
using *φ φ1* **unfolding** *φ-def φa-def* **by** *auto*

show *?thesis*
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - butlast trv1* @ *trv1′*]) **apply**(*rule exI*[*of - butlast trv2* @ *trv2′*])
apply(*rule exI*[*of - statA′*]) **apply**(*rule exI*[*of - statO′*])
unfolding *φa-def* **apply**(*intro conjI*)
subgoal using *φ φ1* **unfolding** *φ-def φa-def* **by** *auto*
subgoal using *φ φ1* **unfolding** *φ-def φa-def* **by** *auto*
subgoal using *len*
by *simp* (*metis Suc-lessI add-is-1 diff-is-0-eq length-greater-0-conv linorder-not-less order-trans trans-less-add2*)
subgoal using *len*
by *simp* (*metis Suc-leI le-add-diff-inverse2 length-greater-0-conv nless-le order-le-less-trans trans-less-add2*)
subgoal using *φ φ1* **unfolding** *φ-def φa-def ssv1*
using *Van.validFromS-append* **by** *auto*
subgoal using *φ φ1* **unfolding** *φ-def φa-def ssv2*
using *Van.validFromS-append* **by** *auto*
subgoal using *φ φ1* **unfolding** *φ-def φa-def Van.S.map-filter Opt.S.map-filter*
apply(*subst tr1l*) **apply**(*subst butlast-append*) **by** *simp*
subgoal using *φ φ1* **unfolding** *φ-def φa-def Van.S.map-filter Opt.S.map-filter*
apply(*subst tr2l*) **apply**(*subst butlast-append*) **by** *simp*
subgoal using *φ φ1* **unfolding** *φ-def φa-def Van.A.map-filter Opt.A.map-filter*
apply(*subst trv1l*) **apply**(*subst trv2l*)
apply(*subst butlast-append*) **apply** *simp* **apply**(*subst butlast-append*) **by** *simp*
subgoal using *φ φ1* **unfolding** *φ-def φa-def* **apply** *simp*
apply(*cases Opt.O tr1* = *Opt.O tr2*, *simp-all*) **apply** *clarify*
using *status.exhaust* **by** (*metis* (*full-types*))+
subgoal using *φ φ1* **unfolding** *φ-def φa-def* **apply** *simp*
apply(*cases Opt.O tr1* = *Opt.O tr2*, *simp-all*) **apply** *clarify*

apply (*smt* (*verit*, *del-insts*) *status.exhaust*)
by (*metis Opt.O.eq-Nil-iff nev*(*1*) *nev*(*2*))
subgoal using φ φ*1* **unfolding** φ*-def* φ*a-def* **by** *simp*
subgoal using φ φ*1* **unfolding** φ*-def* φ*a-def* **by** *simp*
subgoal using φ*1 trv12'NE tr14NE* **unfolding** φ*-def* φ*a-def lastt-def* **by** *simp*
.
qed

lemma *unwindCond-ex-*φ*a-aux2*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: (*statA* = *Diff* $\longrightarrow$ *statO* = *Diff*)
and *v3'*: *Opt.validFromS s1* (*tr1* @ [*s1'*,*s1''*]) **and** *v4'*: *Opt.validFromS s2* (*tr2* @ [*s2'*,*s2''*])
and *i*: *isIntO s1' isIntO s2'*
and *A34*: *getActO s1'* = *getActO s2'*
and *nev*: *never isIntO tr1 never isIntO tr2*
shows $\exists$ *w1' w2' trv1 trv2 statAA statOO*.
φ*a* Δ ∞ *w1 w2 w1' w2' statA s1* (*tr1* @ [*s1'*,*s1''*]) *s2* (*tr2* @ [*s2'*,*s2''*]) *statAA statO sv1 trv1 sv2 trv2 statOO*
proof−
have *0*: *lastt s1* (*tr1* ## *s1'*) = *s1' lastt s2* (*tr2* ## *s2'*) = *s2'*
unfolding *lastt-def* **by** *auto*
show *?thesis*
apply(*rule unwindCond-ex-*φ*a'-aux*[*OF unwind* Δ *r stat*, *of tr1* ## *s1' tr2* ## *s2'*, *unfolded 0*, *simplified*])
using *assms* **by** *auto*
qed

lemma *lastt-snoc*[*simp*]: *lastt s1* (*tr1* @ [*s1''*]) = *s1''*
unfolding *lastt-def* **by** *auto*

lemma *lastt-snoc2*[*simp*]: *lastt s1* (*tr1* @ [*s1'*, *s1''*]) = *s1''*
unfolding *lastt-def* **by** *auto*

lemma *append-snoc2*: *tr1* @ [*s1'*, *s1''*] = (*tr1* ## *s1'*) ## *s1''*
by *auto*

definition φ*'* Δ *w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO sv1 trv1 sv1'' sv2 trv2 sv2'' statOO* ≡
(*trv1* ≠ [] ∨ *w1'* < *w1*) ∧ (*trv2* ≠ [] ∨ *w2'* < *w2*) ∧
Van.validFromS sv1 (*trv1* ## *sv1''*) ∧ *Van.validFromS sv2* (*trv2* ## *sv2''*) ∧
Van.S (*trv1* ## *sv1''*) = *Opt.S* ((*tr1* ## *s1'*) ## *s1''*) ∧ *Van.S* (*trv2* ## *sv2''*) = *Opt.S* ((*tr2* ## *s2'*) ## *s2''*) ∧
Van.A (*trv1* ## *sv1''*) = *Van.A* (*trv2* ## *sv2''*) ∧
(*statO* = *Eq* ⟶ (*statOO* = *Diff*) = (*Van.O* (*trv1* ## *sv1''*) ≠ *Van.O* (*trv2* ## *sv2''*))) ∧

(*statA* = *Eq* ⟶ (*statAA* = *Diff*) = (*Opt.O* ((*tr1* ## *s1*′) ## *s1*′′) ≠ *Opt.O* ((*tr2* ## *s2*′) ## *s2*′′))) ∧
(*statO* = *Diff* ⟶ *statOO* = *Diff*) ∧ (*statAA* = *Diff* ⟶ *statOO* = *Diff*) ∧
Δ ∞ *w1*′ *w2*′ *s1*′′ *s2*′′ *statAA* *sv1*′′ *sv2*′′ *statOO*

proposition *unwindCond-ex-φ*′:
assumes *unwind*: *unwindCond* Δ **and** Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *v3*′: *Opt.validFromS s1* ((*tr1* ## *s1*′) ## *s1*′′) **and** *v4*′: *Opt.validFromS s2* ((*tr2* ## *s2*′) ## *s2*′′)
and *i*: *isIntO s1*′ *isIntO s2*′
and *A34*: *getActO s1*′ = *getActO s2*′
and *nev*: *never isIntO tr1 never isIntO tr2*
shows ∃ *w1*′ *w2*′ *trv1 sv1*′′ *trv2 sv2*′′ *statAA statOO*.
φ′ Δ *w1 w2 w1*′ *w2*′ *statA s1 tr1 s1*′ *s1*′′ *s2 tr2 s2*′ *s2*′′ *statAA statO sv1 trv1 sv1*′′ *sv2 trv2 sv2*′′ *statOO*
using *unwindCond-ex-φa-aux2*[*unfolded φ-def*, *unfolded lastt-snoc lastt-snoc2 append-snoc2*, *OF assms*]
unfolding *φa-def* **apply**(*elim exE*) **subgoal for** *w1*′ *w2*′ *trv1 trv2 statAA statOO*
apply(*cases trv1 rule*: *rev-cases*)
  subgoal by *auto*
  apply(*cases trv2 rule*: *rev-cases*)
    subgoal by *auto*
    subgoal unfolding *φ*′*-def* **apply** *simp* **by** *blast* **. .**

definition *χ3* Δ *w* (*w1*::*enat*) *w2 w1*′ *w2*′ *s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO* ≡
trv1 ≠ [] ∧ *trv2* ≠ [] ∧ (*length trv2* > *Suc 0* ∨ *w2*′ ≤ *w2*) ∧
Van.validFromS sv1 trv1 ∧ *Van.validFromS sv2 trv2* ∧
never isSecV (*butlast trv1*) ∧
isSecV (*lastt sv1 trv1*) ∧ *getSecV* (*lastt sv1 trv1*) = *getSecO* (*lastt s1 tr1*) ∧
never isSecV (*butlast trv2*) ∧
Van.A trv1 = *Van.A trv2* ∧
Δ *w w1*′ *w2*′ (*lastt s1 tr1*) *s2 statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *χ3-final*:
assumes *unw*: *unwindCond* Δ
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *vtr1*: *Opt.validFromS s1 tr1*
and *χ3*: *χ3* Δ *w w1 w2 w1*′ *w2*′ *s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO*
shows (*finalV* (*lastt sv1 trv1*) ⟷ *finalO* (*lastt s1 tr1*)) ∧ (*finalV* (*lastt sv2 trv2*) ⟷ *finalO s2*)
proof−
  have *rsv12*: *Van.validFromS sv1 trv1* ⟶ *reachV* (*lastt sv1 trv1*)
      Van.validFromS sv2 trv2 ⟶ *reachV* (*lastt sv2 trv2*) **using** *r*

by (*simp add*: *Van.reach-validFromS-reach lastt-def*)+
have *rs1*: *Opt.validFromS s1 tr1* ⟶ *reachO* (*lastt s1 tr1*)
using *r*
by (*simp add*: *Opt.reach-validFromS-reach lastt-def*)+
show *?thesis* **using** *χ3*[*unfolded χ3-def*] *rsv12 rs1* **using** *unw*[*unfolded unwind-Cond-def*, *rule-format*,
of lastt s1 tr1 s2 lastt sv1 trv1 lastt sv2 trv2 w w1′ w2′ statAA statOO]
using *vtr1* ‹*reachO s2*› **by** *auto*
qed

lemma *χ3-completedFrom*: *unwindCond* Δ ⟹
reachO s1 ⟹ *reachO s2* ⟹ *reachV sv1* ⟹ *reachV sv2* ⟹
Opt.validFromS s1 tr1 ⟹ *completedFromO s1 tr1* ⟹
χ3 Δ *w w1 w2 w1′ w2′ s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO*
⟹ *completedFromV sv1 trv1* ∧ *completedFromV sv2 trv2*
by (*metis Van.final-not-isSec χ3-def χ3-final completedFromO-lastt*)

lemma *unwindCond-ex-χ3*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *vtr1*: *Opt.validFromS s1 tr1*
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *inter3*: *never isIntO tr1*
and *sec*: *never isSecO* (*butlast tr1*) *isSecO* (*lastt s1 tr1*)
shows ∃ *w′ w1′ w2′ trv1 trv2 statOO*. *χ3* Δ *w′ w1 w2 w1′ w2′ s1 tr1 s2 statA sv1 trv1 sv2 trv2 statOO*
using *assms*(*2*−)
proof(*induction length tr1 w*
arbitrary: *w1 w2 s1 s2 statA sv1 sv2 statO tr1 rule*: *less2-induct′*)
case (*less w tr1 w1 w2 s1 s2 statA sv1 sv2 statO*)
note *vtr1* = *less*(*8*)

note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›
note *nis1* = *less*(*9*) **note** *nis2* = *less*(*10*)
note *inter3* = *less*(*11*)
note *sec3* = *less*(*12*,*13*)
note *r34* = *less.prems*(*2*,*3*) **note** *r12* = *less.prems*(*4*,*5*)
note *r* = *r34 r12*
note *r3* = *r34*(*1*) **note** *r4* = *r34*(*2*) **note** *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)

have *i34*: *statA* = *Eq* ⟶ *isIntO s1* = *isIntO s2*
and *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: $v < w$
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1'*
where *0*:*validTransV* (*sv1*,*sv1'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1' sv2 statO*
unfolding *move-1-def* **by** *auto*
have *r1'*: *reachV sv1'* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w' w1' w2' trv1 trv2 statOO* **where** *χ3*: *χ3* Δ *w' w1 w2 w1' w2' s1 tr1 s2 statA sv1' trv1 sv2 trv2 statOO*
using *less*(*2*)[*OF v*, *of tr1 w1 w2 s1 s2 statA sv1' sv2 statO*,
simplified, *OF* Δ *r34 r1' r2 vtr1 nis1 nis2 inter3 sec3*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - trv2*])
using *χ3 0 sv1* **unfolding** *χ3-def* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv2'*
where *0*: *validTransV* (*sv2*,*sv2'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1 sv2' statO*
unfolding *move-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w' w1' w2' trv1 trv2 statOO* **where** *χ3*: *χ3* Δ *w' w1 w2 w1' w2' s1 tr1 s2 statA sv1 trv1 sv2' trv2 statOO*
using *less*(*2*)[*OF v*, *of tr1 w1 w2 s1 s2 statA sv1 sv2' statO*,
simplified, *OF* Δ *r34 r1 r2' vtr1 nis1 nis2 inter3 sec3*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using *χ3 0 sv2* **unfolding** *χ3-def* **by** *auto*
next
assume *sv12*: ¬ *isSecV sv1* ¬ *isSecV sv2 Van.eqAct sv1 sv2*
and *move-12* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1' sv2' statO'*
where *0*: *statO'* = *sstatO' statO sv1 sv2*
validTransV (*sv1*,*sv1'*) ¬ *isSecV sv1*
validTransV (*sv2*,*sv2'*) ¬ *isSecV sv2*
Van.eqAct sv1 sv2
and Δ: Δ *v w1 w2 s1 s2 statA sv1' sv2' statO'*
unfolding *move-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'* **using** *r1 r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)+

obtain *w' w1' w2' trv1 trv2 statOO* **where** *χ3*: *χ3* Δ *w' w1 w2 w1' w2' s1 tr1 s2 statA sv1' trv1 sv2' trv2 statOO*
using *less*(*2*)[*OF v*, *of tr1 w1 w2 s1 s2 statA sv1' sv2' statO'*,

simplified, *OF* Δ *r34 r12′ vtr1 nis1 nis2 inter3 sec3*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w′*]) **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
apply(*rule exI*[*of - statOO*])
using *χ3 0 sv12* **unfolding** *χ3-def sstatO′-def*
by (*auto simp*: *Van.eqAct-def*)
qed
next
assume *m*: *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
define *statA′* **where** *statA′*: *statA′* = *sstatA′ statA s1 s2*
show *?thesis*
proof(*cases length tr1* ≤ *Suc 0*)
case *True*
hence *tr1e*: *tr1* = [] ∨ *tr1* = [*s1*]
by (*metis Opt.validFromS-singl-iff Suc-length-conv le-Suc-eq le-zero-eq length-0-conv vtr1*)
hence *Opt.A tr1* = [] **by** (*simp add*: *True*)
have *is1*: *isSecO s1*
by (*metis last.simps lastt-def sec3*(*2*) *tr1e*)
hence ¬ *finalO s1* **using** *Opt.final-not-isSec* **by** *blast*
then obtain *s1′* **where** *s13′*: *validTransO* (*s1*, *s1′*) **unfolding** *Opt.final-def* **by** *auto*
hence *isv1*: *isSecV sv1* ∧ *getSecV sv1* = *getSecO s1* **using** *m is1 nis1*
unfolding *match-def match1-def eqSec-def* **by** *auto*
show *?thesis* **using** *tr1e isv1* **apply**−
apply(*rule exI*[*of - w*]) **apply**(*rule exI*[*of - w1*]) **apply**(*rule exI*[*of - w2*])
apply(*rule exI*[*of -* [*sv1*]], *rule exI*[*of -* [*sv2*]], *rule exI*[*of - statO*])
using *tr1e*
using *f34* Δ **by** (*clarsimp simp*: *χ3-def lastt-def*)
next
case *False*
then obtain *s13 tr1′* **where** *tr1*: *tr1* = *s13* # *tr1′* **and** *tr1′NE*: *tr1′* ≠ []
by (*cases tr1*, *auto*)
have *s13*[*simp*]: *s13* = *s1* **using** ‹*Opt.validFromS s1 tr1*›
by (*simp add*: *Opt.validFromS-Cons-iff tr1*)
obtain *s1′* **where**
trn3: *validTransO* (*s1*,*s1′*) **and**
tr1′: *Opt.validFromS s1′ tr1′* **using** ‹*Opt.validFromS s1 tr1*›
unfolding *tr1 s13* **by** (*metis tr1′NE Simple-Transition-System.validFromS-Cons-iff*)
have *r3′*: *reachO s1′* **using** *r3 trn3* **by** (*metis Opt.reach.Step fst-conv snd-conv*)
have *f3*: ¬ *finalO s1* **using** *Opt.final-def trn3* **by** *blast*
hence *f4*: ¬ *finalO s2* **using** *f34* **by** *blast*
have *nev3′*: *never isIntO tr1′*
using *inter3 tr1 tr1′NE* **by** *auto*
have *isAO3*: ¬ *isIntO s1* **by** (*simp add*: *nis1*)
have *O33′*: *Opt.O tr1* = *Opt.O tr1′ Opt.A tr1* = *Opt.A tr1′*
using *isAO3* **unfolding** *tr1* **by** *auto*
have *m*: *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*

have (∃ w1′<w1. ∃ w2′<w2. ¬ isSecO s1 ∧ Δ ∞ w1′ w2′ s1′ s2 statA sv1 sv2 statO) ∨
(∃ w2′<w2. eqSec sv1 s1 ∧ ¬ isIntV sv1 ∧ match1-1 Δ ∞ w2′ s1 s1′ s2 statA sv1 sv2 statO) ∨
(eqSec sv1 s1 ∧ ¬ isSecV sv2 ∧ Van.eqAct sv1 sv2 ∧ match1-12 Δ ∞ ∞ s1 s1′ s2 statA sv1 sv2 statO)
using *m isAO3 trn3* **unfolding** *match1-def* **by** *auto*
thus *?thesis*
proof *safe*
fix *w1′′ w2′′* **assume** *w12′*: *w1′′ < w1 w2′′ < w2*
assume ¬ *isSecO s1* **and** Δ: Δ ∞ *w1′′ w2′′ s1′ s2 statA sv1 sv2 statO*
hence *S3*: *Opt.S tr1′ = Opt.S tr1* **unfolding** *tr1* **by** *auto*
obtain *w′ w1′ w2′ trv1 trv2 statOO* **where** *χ3*: *χ3* Δ *w′ w1′′ w2′′ w1′ w2′ s1′ tr1′ s2 statA sv1 trv1 sv2 trv2 statOO*
using *less*(*1*)[*of tr1′*, *OF* - Δ *r3′ r4 r12* -] **unfolding** *tr1*
by *simp* (*metis Opt.S.eq-Nil-iff*(*2*) *S3 Opt.validFromS-def* ‹¬ *isSecO s1*› *last.simps*
lastt-def list-all-hd nev3′ nis2 s13 sec3(*1*) *sec3*(*2*) *tr1 tr1′*)
show *?thesis* **apply**(*rule exI*[*of* - *w′*]) **apply**(*rule exI*[*of* - *w1′*]) **apply**(*rule exI*[*of* - *w2′*]) **apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using *χ3 O33′* **unfolding** *χ3-def tr1 Van.completedFrom-def*
using *Van.validFromS-Cons tr1′NE tr1′ trn3 isAO3 w12′* **by** *auto*
next
fix *w2′′* **assume** *w2′*: *w2′′ < w2*
assume *trn13*: *eqSec sv1 s1* **and**
Atrn1: ¬ *isIntV sv1* **and** *match1-1* Δ ∞ *w2′′ s1 s1′ s2 statA sv1 sv2 statO*
then obtain *sv1′* **where**
trn1: *validTransV* (*sv1*,*sv1′*) **and**
Δ: Δ ∞ ∞ *w2′′ s1′ s2 statA sv1′ sv2 statO*
unfolding *match1-1-def* **by** *auto*
have *r1′*: *reachV sv1′***using** *r1 trn1* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w′ w1′ w2′ trv1 trv2 statOO* **where** *χ3*: *χ3* Δ *w′* ∞ *w2′′ w1′ w2′ s1′ tr1′ s2 statA sv1′ trv1 sv2 trv2 statOO*

using *less*(*1*)[*of tr1′*, *OF* - Δ *r3′ r4 r1′ r2*, *unfolded O33′*, *simplified*]
using *less.prems tr1′ f3 f4 tr1′NE trn3 O33′*(*1*)
unfolding *tr1*
by *simp* (*metis Opt.validFromS-def list-all-hd*)
show *?thesis* **apply**(*rule exI*[*of* - *w′*]) **apply**(*rule exI*[*of* - *w1′*]) **apply**(*rule exI*[*of* - *w2′*]) **apply**(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *trv2*])
using *χ3 O33′* **unfolding** *χ3-def tr1 Van.completedFrom-def*
using *Van.validFromS-Cons trn1 tr1′NE tr1′ trn3*
using *isAO3 Atrn1 eqSec-S-Cons trn13 w2′*
by *simp* (*metis Opt.S.Nil-iff Opt.S.eq-Nil-iff*(*1*) *eqSec-def nless-le order-le-less-trans s13 sec3*(*1*) *tr1*)
next
assume *sv2*: ¬ *isSecV sv2* **and** *trn13*: *eqSec sv1 s1* **and**
Atrn12: *Van.eqAct sv1 sv2* **and** *match1-12* Δ ∞ ∞ *s1 s1′ s2 statA sv1 sv2*

statO
then obtain *sv1' sv2' statO'* **where**
statO': *statO' = sstatO' statO sv1 sv2* **and**
trn1: *validTransV (sv1,sv1')* **and**
trn2: *validTransV (sv2,sv2')* **and**
Δ: Δ ∞ ∞ ∞ *s1' s2 statA sv1' sv2' statO'*
unfolding *match1-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'*
using *r1 trn1 r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *w' w1' w2' trv1 trv2 statOO* **where** *χ3*: *χ3* Δ *w'* ∞ ∞ *w1' w2' s1' tr1' s2 statA sv1' trv1 sv2' trv2 statOO*
using *less*(*1*)[*of tr1', OF* - Δ *r3' r4 r12', unfolded O33', simplified*]
using *less.prems tr1' f3 f4 tr1'NE trn3 O33'*(*1*) **unfolding** *tr1 statO' sstatO'-def*
by *simp* (*metis Simple-Transition-System.validFromS-def list-all-hd*)+
show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using *χ3 O33' tr1'NE sv2*
using *Van.validFromS-Cons trn1 trn2*
using *isAO3 Atrn12 eqSec-S-Cons trn13 f3 f34 s13 tr1' trn3*
unfolding *χ3-def tr1 Van.completedFrom-def Van.eqAct-def*
using *Van.A.Cons-unfold eqSec-def sec3*(*1*) *tr1* **by** *auto*
qed
qed
qed
qed

definition *χ3a* **where** *χ3a* Δ *w* (*w1::enat*) *w2 w1' w2' s1 s1' s2 statAA sv1 trv1 sv2 trv2 statOO* ≡
trv1 ≠ [] ∧ *trv2* ≠ [] ∧ (*length trv2* > *Suc 0* ∨ *w2'* < *w2*) ∧
Van.validFromS sv1 trv1 ∧ *Van.validFromS sv2 trv2* ∧
Van.S trv1 = [*getSecO s1*] ∧
never isSecV (*butlast trv2*) ∧
Van.A trv1 = *Van.A trv2* ∧
Δ *w w1' w2' s1' s2 statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *unwindCond-ex-χ3a-getSec*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r34*: *reachO s1 reachO s2* **and** *r12*: *reachV sv1 reachV sv2*
and *v*: *validTransO* (*s1, s1'*)
and *ii3*: ¬ *isIntO s1*
and *is1*: *isSecO s1* **and** *isv13*: *isSecV sv1 getSecO s1* = *getSecV sv1*
shows ∃ *w1' w2' trv1 trv2 statOO*.
χ3a Δ ∞ *w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv2 trv2 statOO*
using Δ *r12 isv13*
proof(*induction w arbitrary*: *w1 w2 sv1 sv2 statO rule*: *less-induct*)
case (*less w w1 w2 sv1 sv2 statO*)
note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›

note *r12* = *less.prems*(*2*,*3*)
note *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)
note *r* = *r34 r12*
note *isv13* = ‹*isSecV sv1*› ‹*getSecO s1* = *getSecV sv1*›

have *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
hence *False* **using** *isv13* **by** *blast*
thus *?thesis* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv2'*
where *0*: *validTransV* (*sv2*,*sv2'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1 sv2' statO*
unfolding *move-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1' w2' trv1 trv2 statOO* **where**
χ3a: *χ3a* Δ ∞ *w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv2' trv2 statOO*
using *less*(*1*)[*OF v* Δ *r1 r2' isv13*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using *χ3a 0 sv2* **unfolding** *χ3a-def* **by** *auto*
next
assume *sv12*: ¬ *isSecV sv1* ¬ *isSecV sv2 Van.eqAct sv1 sv2*
and *move-12* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
hence *False* **using** *isv13* **by** *blast*
thus *?thesis* **by** *auto*
qed
next
assume *m*: *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
have *m*: *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have (∃ *w1' w2'*. *w1'*<*w1* ∧ *w2'*<*w2* ∧ ¬ *isSecO s1* ∧ Δ ∞ *w1' w2' s1' s2 statA sv1 sv2 statO*) ∨
(∃ *w2'*< *w2*. *eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧ *match1-1* Δ ∞ *w2' s1 s1' s2 statA sv1 sv2 statO*) ∨
(*eqSec sv1 s1* ∧ ¬ *isSecV sv2* ∧ *Van.eqAct sv1 sv2* ∧ *match1-12* Δ ∞ ∞

s1 s1' s2 statA sv1 sv2 statO)
  using *m v ii3* **unfolding** *match1-def* **by** *auto*

  thus *?thesis*
  apply(*elim disjE exE*)
   subgoal for *w1' w2'* **using** *is1* **by** *auto*
   subgoal for *w2'* **apply**(*rule exI*[*of* - ∞]) **apply**(*rule exI*[*of* - *w2'*])
   unfolding *match1-1-def* **apply**(*elim conjE exE*) **subgoal for** *sv1'*
   apply(*rule exI*[*of* - [*sv1*,*sv1'*]]) **apply**(*rule exI*[*of* - [*sv2*]])
   apply(*rule exI*[*of* - *statO*])
   using *is1 isv13* **unfolding** *χ3a-def*
   by (*auto simp* : *sstatA'-def lastt-def*) **.**
   subgoal apply(*rule exI*[*of* - ∞]) **apply**(*rule exI*[*of* - ∞])
   unfolding *match1-12-def* **apply**(*elim conjE exE*) **subgoal for** *sv1' sv2'*
   apply(*rule exI*[*of* - [*sv1*,*sv1'*]]) **apply**(*rule exI*[*of* - [*sv2*,*sv2'*]])
   apply(*rule exI*[*of* - *sstatO' statO sv1 sv2*])
   using *is1 isv13* **unfolding** *χ3a-def*
   by (*auto simp* : *sstatA'-def sstatO'-def lastt-def Van.eqAct-def*) **. .**
 qed
qed

definition *χ3b* Δ *w* (*w1*::*enat*) *w2 w1' w2' s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO* ≡
trv1 ≠ [] ∧
trv2 ≠ [] ∧ (*length trv2* > *Suc 0* ∨ *w2'* < *w2*) ∧
Van.validFromS sv1 trv1 ∧
Van.validFromS sv2 trv2 ∧
Van.S trv1 = *Opt.S tr1* ∧
never isSecV (*butlast trv2*) ∧ *Van.A trv1* = *Van.A trv2* ∧
Δ *w w1' w2'* (*lastt s1 tr1*) *s2 statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *unwindCond-ex-χ3b-aux*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* **and**
r: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *tr1NE*: *tr1* ≠ []
and *v3'*: *Opt.validFromS s1* (*tr1* ## *s1'*)
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter3'*: *never isIntO* (*tr1* ## *s1'*)
and *sec*: *never isSecO* (*butlast tr1*) *isSecO* (*lastt s1 tr1*)
shows ∃ *w1' w2' trv1 trv2 statOO*. *χ3b* Δ ∞ *w1 w2 w1' w2' s1* (*tr1* ## *s1'*) *s2 statA sv1 trv1 sv2 trv2 statOO*
proof−
 have *v3*: *Opt.validFromS s1 tr1* **and** *s13'*: *validTransO* (*lastt s1 tr1*,*s1'*)
 apply (*metis v3' Opt.validFromS-def Opt.validS-append1 Nil-is-append-conv hd-append2*)
 by (*metis Opt.validFromS-def Opt.validS-validTrans lastt-def list.sel*(*1*) *not-Cons-self2 snoc-eq-iff-butlast tr1NE v3'*)

have *ninter3*: *never isIntO tr1* **and** *nis1'*: ¬ *isIntO s1'*
using *ninter3'* **by** *auto*
obtain *ww ww1 ww2 trv1 trv2 statOO* **where** *χ3*: *χ3* Δ *ww w1 w2 ww1 ww2 s1 tr1 s2 statA sv1 trv1 sv2 trv2 statOO*
using *unwindCond-ex-χ3*[*OF unwind* Δ *r v3 nis1 nis2 ninter3 sec*] **by** *auto*

have *trv12NE*: *trv1* ≠ [] *trv2* ≠ [] **using** *χ3* **unfolding** *χ3-def* **by** *auto*

define *ss1 ssv1 ssv2* **where** *ss1*: *ss1* ≡ *lastt s1 tr1*
and *ssv1*: *ssv1* ≡ *lastt sv1 trv1* **and** *ssv2*: *ssv2* ≡ *lastt sv2 trv2*

have *ss1l*: *ss1* = *last tr1* **by** (*simp add*: *lastt-def ss1 tr1NE*)
have *tr1l*: *tr1* = *butlast tr1* @ [*ss1*] **by** (*simp add*: *ss1l tr1NE*)
have *ssv1l*: *ssv1* = *last trv1* **using** *χ3* **unfolding** *χ3-def* **by** (*metis lastt-def ssv1*)
have *trv1l*: *trv1* = *butlast trv1* @ [*ssv1*] **by** (*simp add*: *ssv1l trv12NE*(*1*))
have *ssv2l*: *ssv2* = *last trv2* **using** *χ3* **unfolding** *χ3-def* **by** (*metis lastt-def ssv2*)
have *trv2l*: *trv2* = *butlast trv2* @ [*ssv2*] **by** (*simp add*: *ssv2l trv12NE*(*2*))

have *iss1*[*simp*]: *isSecO ss1* **using** *sec*(*2*) **unfolding** *ss1* **by** *auto*
have *issv1*[*simp*]: *isSecV ssv1* **and** *gissv13*[*simp*]: *getSecO ss1* = *getSecV ssv1*
using *χ3* **unfolding** *χ3-def ssv1 ss1* **by** *auto*

have *niss1*: ¬ *isIntO ss1*
by (*metis list-all-append list-all-simps*(*1*) *ninter3 tr1l*)

have *rss1*: *reachO ss1* **and** *rssv12*: *reachV ssv1 reachV ssv2*
using *Opt.reach-validFromS-reach r ss1l tr1NE v3* **apply** *blast*
apply (*metis Van.reach-validFromS-reach χ3-def χ3 r*(*3*) *ssv1l*)
by (*metis Van.reach-validFromS-reach χ3-def χ3 r*(*4*) *ssv2l*)

have Δ: Δ *ww ww1 ww2 ss1 s2 statA ssv1 ssv2 statOO*
using *χ3* **unfolding** *χ3-def ss1*[*symmetric*] *ssv1*[*symmetric*] *ssv2*[*symmetric*] **by** *auto*

have *s13'*: *validTransO* (*ss1*, *s1'*)
by (*simp add*: *s13' ss1*)

note *vs13* = *s13'*[*unfolded ss1*[*symmetric*]]
obtain *w1' w2' trv1' trv2' statO'* **where**
χ3a: *χ3a* Δ ∞ *ww1 ww2 w1' w2' ss1 s1' s2 statA ssv1 trv1' ssv2 trv2' statO'*
using *unwindCond-ex-χ3a-getSec*[*OF unwind* Δ *rss1 r*(*2*) *rssv12 s13' niss1 iss1 issv1 gissv13*]
by *blast*

have *trv12'NE*: *trv1'* ≠ [] *trv2'* ≠ [] **using** *χ3a* **unfolding** *χ3a-def* **by** *auto*

have [*simp*]: *Van.O* (*butlast trv1* @ *trv1'*) = *Van.O trv1* @ *Van.O trv1'*
using *trv12'NE* **unfolding** *χ3-def Van.O.map-filter Opt.O.map-filter* **apply**(*subst butlast-append*) **by** *simp*

have [*simp*]: *Van.O* (*butlast trv2* @ *trv2'*) = *Van.O trv2* @ *Van.O trv2'*
using *trv12'NE* **unfolding** *χ3-def Van.O.map-filter Opt.O.map-filter* **apply**(*subst butlast-append*) **by** *simp*

have *Van.A trv1'* = *Van.A trv2'* **using** *χ3a* **unfolding** *χ3a-def* **by** *auto*
moreover have *length* (*Van.O trv1'*) = *length* (*Van.A trv1'*) ∧ *length* (*Van.O trv2'*) = *length* (*Van.A trv2'*)
unfolding *Van.A.map-filter Van.O.map-filter* **by** *auto*
ultimately have *length* (*Van.O trv1'*) = *length* (*Van.O trv2'*) **by** *auto*
hence [*simp*]: *Van.O trv1* @ *Van.O trv1'* = *Van.O trv2* @ *Van.O trv2'* ⟷
Van.O trv1 = *Van.O trv2* ∧ *Van.O trv1'* = *Van.O trv2'* **by** *auto*

show *?thesis*
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - butlast trv1* @ *trv1'*]) **apply**(*rule exI*[*of - butlast trv2* @ *trv2'*])
apply(*rule exI*[*of - statO'*])
unfolding *χ3b-def* **apply**(*intro conjI*)
subgoal using *χ3 χ3a* **unfolding** *χ3-def χ3a-def* **by** *auto*
subgoal using *χ3 χ3a* **unfolding** *χ3-def χ3a-def* **by** *auto*
subgoal using *χ3 χ3a* **unfolding** *χ3-def χ3a-def*
by *simp* (*metis Simple-Transition-System.fromS-eq-Nil Simple-Transition-System.toS-fromS-nonSingl Van.toS-Nil diff-add-inverse2 linorder-not-less order-le-less-trans trans-less-add2 zero-less-diff*)

subgoal using *χ3 χ3a* **unfolding** *χ3-def χ3a-def ssv1*
using *Van.validFromS-append* **by** *auto*
subgoal using *χ3 χ3a* **unfolding** *χ3-def χ3a-def ssv2*
using *Van.validFromS-append* **by** *auto*
subgoal using *χ3 χ3a* **unfolding** *χ3-def χ3a-def* **unfolding** *Van.S.map-filter Opt.S.map-filter*
apply(*subst tr1l*) **apply**(*subst butlast-append*)
by *simp* (*metis Opt.S.map-filter Opt.S.eq-Nil-iff*(*2*) *Van.S.map-filter Van.S.eq-Nil-iff*(*2*) *sec*(*1*))
subgoal using *χ3 χ3a* **unfolding** *χ3-def χ3a-def*
by (*simp add*: *butlast-append*)
subgoal using *χ3 χ3a* **unfolding** *χ3-def χ3a-def Van.A.map-filter Opt.A.map-filter*

apply(*subst trv1l*) **apply**(*subst trv2l*) **by** (*simp add*: *butlast-append*)
subgoal using *χ3a trv12'NE tr1NE* **unfolding** *χ3a-def lastt-def* **by** *simp* **.**
qed

lemma *unwindCond-ex-χ3b-aux2*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*

and *v3'*: *Opt.validFromS s1* (*tr1* @ [*s1'*,*s1''*])
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter3'*: *never isIntO* (*tr1* @ [*s1'*,*s1''*])
and *sec*: *never isSecO tr1 isSecO s1'*
shows ∃ *w1' w2' trv1 trv2 statOO*. *χ3b* Δ ∞ *w1 w2 w1' w2' s1* (*tr1* @ [*s1'*,*s1''*]) *s2 statA sv1 trv1 sv2 trv2 statOO*
proof−
  have *0*: *lastt s1* (*tr1* ## *s1'*) = *s1'*
  unfolding *lastt-def* **by** *auto*
  show *?thesis*
  using *unwindCond-ex-χ3b-aux*[*OF unwind* Δ *r*, *of tr1* ## *s1'*, *unfolded 0*, *simplified*]
  using *assms* **by** *auto*
qed

definition *χ3'* Δ *w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statAA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO* ≡
  Van.validFromS sv1 (*trv1* ## *sv1''*) ∧ *Van.validFromS sv2* (*trv2* ## *sv2''*) ∧
  Van.S (*trv1* ## *sv1''*) = *Opt.S* ((*tr1* ## *s1'*) ## *s1''*) ∧ *never isSecV trv2* ∧
  Van.A (*trv1* ## *sv1''*) = *Van.A* (*trv2* ## *sv2''*) ∧
  trv1 ≠ [] ∧ (*trv2* ≠ [] ∨ *w2'* < *w2*) ∧
  Δ ∞ *w1' w2' s1'' s2 statAA sv1'' sv2'' statOO*

proposition *unwindCond-ex-χ3'*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* **and**
r: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *v3'*: *Opt.validFromS s1* ((*tr1* ## *s1'*) ## *s1''*)
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter3'*: *never isIntO* ((*tr1* ## *s1'*) ## *s1''*)
and *sec*: *never isSecO tr1 isSecO s1'*
shows ∃ *w1' w2' trv1 sv1'' trv2 sv2'' statOO*. *χ3'* Δ *w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
using *unwindCond-ex-χ3b-aux2*[*unfolded φ-def*, *unfolded lastt-snoc lastt-snoc2 append-snoc2*, *OF assms*]
unfolding *χ3b-def* **apply**(*elim exE*) **subgoal for** *w1' w2' trv1 trv2 statOO*
apply(*cases trv1 rule*: *rev-cases*)
  subgoal by *auto*
  subgoal for *trv1' sv1''* **apply**(*cases trv2 rule*: *rev-cases*)
    subgoal by *auto*
    subgoal for *trv2' sv2''* **unfolding** *χ3'-def*
    apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
    apply(*rule exI*[*of - trv1'*]) **apply**(*rule exI*[*of - sv1''*])
    apply(*rule exI*[*of - trv2'*]) **apply**(*rule exI*[*of - sv2''*])
    apply(*rule exI*[*of - statOO*])
    by *simp* (*metis Opt.S.Nil-iff Opt.S.eq-Nil-iff*(*1*) *Van.S.simps*(*4*) *append-snoc2*

list-all-append sec(*2*)
self-append-conv2 snoc-eq-iff-butlast) . . .

definition *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statAA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO* ≡
Van.validFromS sv1 (*trv1* ## *sv1′*) ∧ *Van.validFromS sv2* (*trv2* ## *sv2′*) ∧
never isSecV trv1 ∧ *never isSecV trv2* ∧
Van.A (*trv1* ## *sv1′*) = *Van.A* (*trv2* ## *sv2′*) ∧
(*trv1* ≠ [] ∨ *w1′* < *w1*) ∧ (*trv2* ≠ [] ∨ *w2′* < *w2*) ∧
Δ ∞ *w1′ w2′ s1′ s2 statAA sv1′ sv2′ statOO*

proposition *unwindCond-ex-ω3*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r34*: *reachO s1 reachO s2* **and** *r12*: *reachV sv1 reachV sv2*
and *v3*: *validTransO* (*s1*,*s1′*)
and *nis1*: ¬ *isIntO s1* ¬ *isIntO s1′* ¬ *isSecO s1*
and *nis2*: ¬ *isIntO s2*
shows ∃ *w1′ w2′ trv1 sv1′ trv2 sv2′ statOO*. *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
using Δ *r12*
proof(*induction w arbitrary*: *w1 w2 sv1 sv2 statO rule*: *less-induct*)
case (*less w w1 w2 sv1 sv2 statO*)
note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›
note *r12* = *less.prems*(*2*,*3*)
note *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)
note *r* = *r34 r12*

have *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1′* **where** *0*: *validTransV* (*sv1*, *sv1′*) **and** Δ: Δ *v w1 w2 s1 s2 statA sv1′ sv2 statO*
unfolding *move-1-def* **by** *auto*
have *r1′*: *reachV sv1′* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1′ w2′ trv1 sv1″ trv2 sv2′ statOO* **where**
ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1′ trv1 sv1″ sv2 trv2 sv2′ statOO*

using *less*(*1*)[*OF v* Δ *r1′ r2*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv1′′*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2′*])
using *ω3 0 sv1* **unfolding** *ω3-def* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv2′*
where *0*: *validTransV* (*sv2*,*sv2′*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1 sv2′ statO*
unfolding *move-2-def* **by** *auto*
have *r2′*: *reachV sv2′* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1′ w2′ trv1 sv1′ trv2 sv2′′ statOO* **where**
ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2′ trv2 sv2′′ statOO*

using *less*(*1*)[*OF v* Δ *r1 r2′*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1′*])
apply(*rule exI*[*of - sv2 # trv2*]) **apply**(*rule exI*[*of - sv2′′*])
using *ω3 0 sv2* **unfolding** *ω3-def* **by** *auto*
next
assume *sv1*: ¬ *isSecV sv1* **and** *sv2*: ¬ *isSecV sv2* **and**
move-12 Δ *v w1 w2 s1 s2 statA sv1 sv2 statO* **and**
sv12: *Van.eqAct sv1 sv2*
then obtain *sv1′ sv2′ statO′*
where *statO′*: *statO′* = *sstatO′ statO sv1 sv2*
and *0*: *validTransV* (*sv1*,*sv1′*) *validTransV* (*sv2*,*sv2′*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1′ sv2′ statO′*
unfolding *move-12-def* **by** *auto*
have *r1′*: *reachV sv1′* **and** *r2′*: *reachV sv2′* **using** *r1 r2 0*
by (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *w1′ w2′ trv1 sv1′′ trv2 sv2′′ statOO* **where**
ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1′ trv1 sv1′′ sv2′ trv2 sv2′′ statOO*

using *less*(*1*)[*OF v* Δ *r1′ r2′*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv1′′*])
apply(*rule exI*[*of - sv2 # trv2*]) **apply**(*rule exI*[*of - sv2′′*])
using *ω3 0 sv1 sv2 sv12* **unfolding** *ω3-def statO′* **by** (*auto simp*: *Van.eqAct-def*)
qed
next
assume *m*: *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
have *m*: *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have (∃ *w1′ w2′*. *w1′* < *w1* ∧ *w2′*<*w2* ∧ ¬ *isSecO s1* ∧ Δ ∞ *w1′ w2′ s1′ s2 statA sv1 sv2 statO*) ∨
(∃ *w2′*<*w2*. *eqSec sv1 s1* ∧ ¬ *isIntV sv1* ∧ *match1-1* Δ ∞ *w2′ s1 s1′ s2 statA sv1 sv2 statO*) ∨

(*eqSec sv1 s1* ∧ ¬ *isSecV sv2* ∧ *Van.eqAct sv1 sv2* ∧ *match1-12* Δ ∞ ∞ *s1 s1′ s2 statA sv1 sv2 statO*)
using *m v3 nis1* **unfolding** *match1-def* **by** *auto*

thus *?thesis*
apply(*elim disjE exE*)
subgoal for *w1′ w2′*
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of -* []]) **apply**(*rule exI*[*of - sv1*])
apply(*rule exI*[*of -* []]) **apply**(*rule exI*[*of - sv2*])
apply(*rule exI*[*of - statO*]) **unfolding** *ω3-def*
by *auto*
subgoal for *w2′*
apply(*rule exI*[*of -* ∞]) **apply**(*rule exI*[*of - w2′*])
unfolding *match1-1-def* **apply**(*elim conjE exE*) **subgoal for** *sv1′*
apply(*rule exI*[*of -* [*sv1*]]) **apply**(*rule exI*[*of - sv1′*])
apply(*rule exI*[*of -* []]) **apply**(*rule exI*[*of - sv2*])
apply(*rule exI*[*of - statO*])
unfolding *ω3-def* **using** *nis1*(*3*) **by** (*auto simp*: *eqSec-def*) **.**
subgoal
apply(*rule exI*[*of -* ∞]) **apply**(*rule exI*[*of -* ∞])
unfolding *match1-12-def* **apply**(*elim conjE exE*) **subgoal for** *sv1′ sv2′*
apply(*rule exI*[*of -* [*sv1*]]) **apply**(*rule exI*[*of - sv1′*])
apply(*rule exI*[*of -* [*sv2*]]) **apply**(*rule exI*[*of - sv2′*])
apply(*rule exI*[*of - sstatO′ statO sv1 sv2*])
unfolding *ω3-def* **using** *nis1*(*3*) **apply** (*auto simp*: *eqSec-def sstatA′-def sstatO′-def lastt-def Van.eqAct-def*) **. . .**
qed
qed

definition *χ4* Δ *w w1* (*w2*::*enat*) *w1′ w2′ s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO* ≡
trv1 ≠ [] ∧ *trv2* ≠ [] ∧ (*length trv1* > *Suc 0* ∨ *w1′* ≤ *w1*) ∧
Van.validFromS sv1 trv1 ∧ *Van.validFromS sv2 trv2* ∧
never isSecV (*butlast trv1*) ∧
never isSecV (*butlast trv2*) ∧
isSecV (*lastt sv2 trv2*) ∧ *getSecV* (*lastt sv2 trv2*) = *getSecO* (*lastt s2 tr2*) ∧
Van.A trv1 = *Van.A trv2* ∧
Δ *w w1′ w2′ s1* (*lastt s2 tr2*) *statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *χ4-final*:
assumes *unw*: *unwindCond* Δ
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *vtr2*: *Opt.validFromS s2 tr2*
and *χ4*: *χ4* Δ *w w1 w2 w1′ w2′ s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO*
shows (*finalV* (*lastt sv1 trv1*) ⟷ *finalO s1*) ∧ (*finalV* (*lastt sv2 trv2*) ⟷ *finalO*

(*lastt s2 tr2*))
proof−
  have *rsv12*: *Van.validFromS sv1 trv1* ⟶ *reachV* (*lastt sv1 trv1*)
    Van.validFromS sv2 trv2 ⟶ *reachV* (*lastt sv2 trv2*) **using** *r*
   by (*simp add*: *Van.reach-validFromS-reach lastt-def*)+
  have *rs2*: *Opt.validFromS s2 tr2* ⟶ *reachO* (*lastt s2 tr2*)
  using *r*
   by (*simp add*: *Opt.reach-validFromS-reach lastt-def*)+
  show *?thesis* **using** *χ4*[*unfolded χ4-def*] *rsv12 rs2* **using** *unw*[*unfolded unwind-Cond-def*, *rule-format*,
   of s1 lastt s2 tr2 lastt sv1 trv1 lastt sv2 trv2 w w1′ w2′ statAA statOO]
  using *vtr2* ‹*reachO s1*› **by** *auto*
qed

lemma *χ4-completedFrom*: *unwindCond* Δ ⟹
reachO s1 ⟹ *reachO s2* ⟹ *reachV sv1* ⟹ *reachV sv2* ⟹
Opt.validFromS s2 tr2 ⟹ *completedFromO s2 tr2* ⟹
χ4 Δ *w w1 w2 w1′ w2′ s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO*
⟹ *completedFromV sv1 trv1* ∧ *completedFromV sv2 trv2*
by (*metis Van.final-not-isSec χ4-def χ4-final completedFromO-lastt*)

proposition *unwindCond-ex-χ4*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *vtr2*: *Opt.validFromS s2 tr2*
and *nis2*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *inter4*: *never isIntO tr2*
and *sec*: *never isSecO* (*butlast tr2*) *isSecO* (*lastt s2 tr2*)
shows ∃ *w′ w1′ w2′ trv1 trv2 statOO*. *χ4* Δ *w′ w1 w2 w1′ w2′ s1 s2 tr2 statA sv1 trv1 sv2 trv2 statOO*
using *assms*(*2*−)
proof(*induction length tr2 w*
  arbitrary: *w1 w2 s1 s2 statA sv1 sv2 statO tr2 rule*: *less2-induct′*)
  case (*less w tr2 w1 w2 s1 s2 statA sv1 sv2 statO*)
  note *vtr2* = *less*(*8*)
  note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›
  note *nis1* = *less*(*9*) **note** *nis2* = *less*(*10*)
  note *inter4* = *less*(*11*)
  note *sec4* = *less*(*12*,*13*)
  note *r34* = *less.prems*(*2*,*3*) **note** *r12* = *less.prems*(*4*,*5*)
  note *r* = *r34 r12*
  note *r3* = *r34*(*1*) **note** *r4* = *r34*(*2*) **note** *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)

  have *i34*: *statA* = *Eq* ⟶ *isIntO s1* = *isIntO s2*
  and *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
   using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: ($\exists$ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) $\vee$ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1'*
where *0*:*validTransV* (*sv1*,*sv1'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1' sv2 statO*
unfolding *move-1-def* **by** *auto*
have *r1'*: *reachV sv1'* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w' w1' w2' trv1 trv2 statOO* **where** χ*4*: χ*4* Δ *w' w1 w2 w1' w2' s1 s2 tr2 statA sv1' trv1 sv2 trv2 statOO*
using *less*(*2*)[*OF v*, *of tr2 w1 w2 s1 s2 statA sv1' sv2 statO*,
simplified, *OF* Δ *r34 r1' r2 vtr2 nis1 nis2 inter4 sec4*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - trv2*])
using χ*4 0 sv1* **unfolding** χ*4-def* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv2'*
where *0*: *validTransV* (*sv2*,*sv2'*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1 sv2' statO*
unfolding *move-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1' w2' w' trv1 trv2 statOO* **where** χ*4*: χ*4* Δ *w' w1 w2 w1' w2' s1 s2 tr2 statA sv1 trv1 sv2' trv2 statOO*
using *less*(*2*)[*OF v*, *of tr2 w1 w2 s1 s2 statA sv1 sv2' statO*,
simplified, *OF* Δ *r34 r1 r2' vtr2 nis1 nis2 inter4 sec4*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
using χ*4 0 sv2* **unfolding** χ*4-def* **by** *auto*
next
assume *sv12*: ¬ *isSecV sv1* ¬ *isSecV sv2 Van.eqAct sv1 sv2*
and *move-12* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1' sv2' statO'*
where *0*: *statO'* = *sstatO' statO sv1 sv2*
validTransV (*sv1*,*sv1'*) ¬ *isSecV sv1*
validTransV (*sv2*,*sv2'*) ¬ *isSecV sv2*
Van.eqAct sv1 sv2
and Δ: Δ *v w1 w2 s1 s2 statA sv1' sv2' statO'*
unfolding *move-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'* **using** *r1 r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *w' w1' w2' trv1 trv2 statOO* **where** χ*4*: χ*4* Δ *w' w1 w2 w1' w2' s1*

s2 tr2 statA sv1′ trv1 sv2′ trv2 statOO
using *less*(*2*)[*OF v*, *of tr2 w1 w2 s1 s2 statA sv1′ sv2′ statO′*,
simplified, *OF* Δ *r34 r12′ vtr2 nis1 nis2 inter4 sec4*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w′*]) **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])
apply(*rule exI*[*of - statOO*])
using *χ4 0 sv12* **unfolding** *χ4-def sstatO′-def*
by (*auto simp*: *Van.eqAct-def*)
qed
next
assume *m*: *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
define *statA′* **where** *statA′*: *statA′* = *sstatA′ statA s1 s2*
show *?thesis*
proof(*cases length tr2* ≤ *Suc 0*)
case *True*
hence *tr2e*: *tr2* = [] ∨ *tr2* = [*s2*]
by (*metis Opt.validFromS-def Suc-length-conv le-Suc-eq le-zero-eq length-0-conv list.sel*(*1*) *vtr2*)
hence *Opt.A tr2* = [] **by** (*simp add*: *True*)
have *is2*: *isSecO s2*
by (*metis last.simps lastt-def sec4*(*2*) *tr2e*)
hence ¬ *finalO s2* **using** *Opt.final-not-isSec* **by** *blast*
then obtain *s2′* **where** *s24′*: *validTransO* (*s2*, *s2′*) **unfolding** *Opt.final-def* **by** *auto*
hence *isv2*: *isSecV sv2* ∧ *getSecV sv2* = *getSecO s2* **using** *m is2 nis2*
unfolding *match-def match2-def eqSec-def* **by** *auto*
show *?thesis* **using** *tr2e isv2* **apply**−
apply(*rule exI*[*of - w*]) **apply**(*rule exI*[*of - w1*]) **apply**(*rule exI*[*of - w2*])
apply(*rule exI*[*of -* [*sv1*]], *rule exI*[*of -* [*sv2*]], *rule exI*[*of - statO*])
using *tr2e*
using *f34* Δ **by**(*clarsimp simp*: *χ4-def lastt-def*)
next
case *False*
then obtain *s24 tr2′* **where** *tr2*: *tr2* = *s24* # *tr2′* **and** *tr2′NE*: *tr2′* ≠ []
by (*cases tr2*, *auto*)
have *s24*[*simp*]: *s24* = *s2* **using** ‹*Opt.validFromS s2 tr2*›
by (*simp add*: *Opt.validFromS-Cons-iff tr2*)
obtain *s2′* **where**
trn4: *validTransO* (*s2*,*s2′*) **and**
tr2′: *Opt.validFromS s2′ tr2′* **using** ‹*Opt.validFromS s2 tr2*›
unfolding *tr2 s24* **by** (*metis tr2′NE Simple-Transition-System.validFromS-Cons-iff*)
have *r4′*: *reachO s2′* **using** *r4 trn4* **by** (*metis Opt.reach.Step fst-conv snd-conv*)
have *f4*: ¬ *finalO s2* **using** *Opt.final-def trn4* **by** *blast*
hence *f3*: ¬ *finalO s1* **using** *f34* **by** *blast*
have *nev4′*: *never isIntO tr2′*
using *inter4 tr2 tr2′NE* **by** *auto*
have *isAO4*: ¬ *isIntO s2* **by** (*simp add*: *nis2*)
have *O44′*: *Opt.O tr2* = *Opt.O tr2′ Opt.A tr2* = *Opt.A tr2′*
using *isAO4* **unfolding** *tr2* **by** *auto*

have *m*: *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*

have ($\exists$ *w1'<w1*. $\exists$ *w2'<w2*. $\neg$ *isSecO s2* $\wedge$ Δ ∞ *w1' w2' s1 s2' statA sv1 sv2 statO*) $\vee$

($\exists$ *w1'<w1*. *eqSec sv2 s2* $\wedge$ $\neg$ *isIntV sv2* $\wedge$ *match2-1* Δ *w1'* ∞ *s1 s2 s2' statA sv1 sv2 statO*) $\vee$

(*eqSec sv2 s2* $\wedge$ $\neg$ *isSecV sv1* $\wedge$ *Van.eqAct sv1 sv2* $\wedge$ *match2-12* Δ ∞ ∞ *s1 s2 s2' statA sv1 sv2 statO*)

using *m isAO4 trn4* **unfolding** *match2-def* **by** *auto*

thus *?thesis*

proof *safe*

fix *w1'' w2''* **assume** *w12'*: *w1'' < w1 w2'' < w2*

assume $\neg$ *isSecO s2* **and** Δ: Δ ∞ *w1'' w2'' s1 s2' statA sv1 sv2 statO*

hence *S4*: *Opt.S tr2'* = *Opt.S tr2* **unfolding** *tr2* **by** *auto*

obtain *w' w1' w2' trv1 trv2 statOO* **where** $\chi 4$: $\chi 4$ Δ *w' w1'' w2'' w1' w2' s1 s2' tr2' statA sv1 trv1 sv2 trv2 statOO*

using *less*(*1*)[*of tr2'*, *OF - Δ r3 r4' r12*] **unfolding** *tr2*

by *simp* (*metis Opt.S.eq-Nil-iff*(*2*) *S4 Simple-Transition-System.validFromS-def last.simps lastt-def*

list.discI list-all-hd nev4' nis1 sec4(*1*) *sec4*(*2*) *tr2 tr2' tr2'NE*)

show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - trv2*])

using $\chi 4$ *O44'* **unfolding** $\chi 4$*-def tr2 Van.completedFrom-def*

using *Van.validFromS-Cons tr2'NE tr2' trn4 isAO4 w12'* **by** *auto*

next

fix *w1''* **assume** *w1'*: *w1'' < w1*

assume *trn24*: *eqSec sv2 s2* **and**

Atrn2: $\neg$ *isIntV sv2* **and** *match2-1* Δ *w1''* ∞ *s1 s2 s2' statA sv1 sv2 statO*

then obtain *sv2'* **where**

trn2: *validTransV* (*sv2*,*sv2'*) **and**

Δ: Δ ∞ *w1''* ∞ *s1 s2' statA sv1 sv2' statO*

unfolding *match2-1-def* **by** *auto*

have *r2'*: *reachV sv2'* **using** *r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)

obtain *w' w1' w2' trv1 trv2 statOO* **where** $\chi 4$: $\chi 4$ Δ *w' w1''* ∞ *w1' w2' s1 s2' tr2' statA sv1 trv1 sv2' trv2 statOO*

using *less*(*1*)[*of tr2'*, *OF - Δ r3 r4' r1 r2'*, *unfolded O44'*, *simplified*]

using *less.prems tr2' f3 f4 tr2'NE trn4 O44'*(*1*)

unfolding *tr2*

by *simp* (*metis Opt.validFromS-def list-all-hd*)

show *?thesis* **apply**(*rule exI*[*of - w'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv2 # trv2*])

using $\chi 4$ *O44'* **unfolding** $\chi 4$*-def tr2 Van.completedFrom-def*

using *Van.validFromS-Cons trn2 tr2'NE tr2' trn4*

using *isAO4 Atrn2 eqSec-S-Cons trn24 w1'*

by *simp* (*metis Opt.S.Nil-iff Opt.S.eq-Nil-iff*(*1*) *eqSec-def nless-le order-le-less-trans s24 sec4*(*1*) *tr2*)

next

assume *sv1*: $\neg$ *isSecV sv1* **and** *trn24*: *eqSec sv2 s2* **and**

Atrn12: *Van.eqAct sv1 sv2* **and** *match2-12* Δ ∞ ∞ *s1 s2 s2' statA sv1 sv2 statO*
then obtain *sv1' sv2' statO'* **where**
statO': *statO' = sstatO' statO sv1 sv2* **and**
trn1: *validTransV* (*sv1*,*sv1'*) **and**
trn2: *validTransV* (*sv2*,*sv2'*) **and**
Δ: Δ ∞ ∞ ∞ *s1 s2' statA sv1' sv2' statO'*
unfolding *match2-12-def* **by** *auto*
have *r12'*: *reachV sv1' reachV sv2'*
using *r1 trn1 r2 trn2* **by** (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *w' w1' w2' trv1 trv2 statOO* **where** *χ4*: *χ4* Δ *w'* ∞ ∞ *w1' w2' s1 s2' tr2' statA sv1' trv1 sv2' trv2 statOO*
using *less*(*1*)[*of tr2'*, *OF* - Δ *r3 r4' r12'*, *unfolded O44'*, *simplified*]
using *less.prems tr2' f3 f4 tr2'NE trn4 O44'*(*1*) **unfolding** *tr2 statO' sstatO'-def*
by *simp* (*metis Simple-Transition-System.validFromS-def list-all-hd*)
show *?thesis* **apply**(*rule exI*[*of* - *w'*]) **apply**(*rule exI*[*of* - *w1'*]) **apply**(*rule exI*[*of* - *w2'*])
apply(*rule exI*[*of* - *sv1* # *trv1*]) **apply**(*rule exI*[*of* - *sv2* # *trv2*])
using *χ4 O44' tr2'NE sv1*
using *Van.validFromS-Cons trn1 trn2*
using *isAO4 Atrn12 eqSec-S-Cons trn24 f3 f34 s24 tr2' trn4*
unfolding *χ4-def tr2 Van.completedFrom-def Van.eqAct-def*
using *Van.A.Cons-unfold eqSec-def sec4*(*1*) *tr2* **by** *auto*
qed
qed
qed
qed

definition *χ4a* **where** *χ4a* Δ *w w1* (*w2*::*enat*) *w1' w2' s1 s2 s2' statAA sv1 trv1 sv2 trv2 statOO* ≡
trv1 ≠ [] ∧ *trv2* ≠ [] ∧ (*length trv1* > *Suc 0* ∨ *w1'* < *w1*) ∧
Van.validFromS sv1 trv1 ∧ *Van.validFromS sv2 trv2* ∧
never isSecV (*butlast trv1*) ∧
Van.S trv2 = [*getSecO s2*] ∧
Van.A trv1 = *Van.A trv2* ∧
Δ *w w1' w2' s1 s2' statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *unwindCond-ex-χ4a-getSec*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r34*: *reachO s1 reachO s2* **and** *r12*: *reachV sv1 reachV sv2*
and *v*: *validTransO* (*s2*, *s2'*)
and *ii4*: ¬ *isIntO s2*
and *is2*: *isSecO s2* **and** *isv24*: *isSecV sv2 getSecO s2* = *getSecV sv2*
shows ∃ *w1' w2' trv1 trv2 statOO*.
χ4a Δ ∞ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv2 trv2 statOO*
using Δ *r12 isv24*
proof(*induction w arbitrary*: *w1 w2 sv1 sv2 statO rule*: *less-induct*)

case (*less w w1 w2 sv1 sv2 statO*)
note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›
note *r12* = *less.prems*(*2*,*3*)
note *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)
note *r* = *r34 r12*
note *isv24* = ‹*isSecV sv2*› ‹*getSecO s2* = *getSecV sv2*›

have *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1′*
where *0*: *validTransV* (*sv1*,*sv1′*)
and Δ: Δ *v w1 w2 s1 s2 statA sv1′ sv2 statO*
unfolding *move-1-def* **by** *auto*
have *r1′*: *reachV sv1′* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1′ w2′ trv1 trv2 statOO* **where**
χ4a: *χ4a* Δ ∞ *w1 w2 w1′ w2′ s1 s2 s2′ statA sv1′ trv1 sv2 trv2 statOO*
using *less*(*1*)[*OF v* Δ *r1′ r2 isv24*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - trv2*])
using *χ4a 0 sv1* **unfolding** *χ4a-def* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
hence *False* **using** *isv24* **by** *blast*
thus *?thesis* **by** *auto*
next
assume *sv12*: ¬ *isSecV sv1* ¬ *isSecV sv2 Van.eqAct sv1 sv2*
and *move-12* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
hence *False* **using** *isv24* **by** *blast*
thus *?thesis* **by** *auto*
qed
next
assume *m*: *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
have *m*: *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*
have (∃ *w1′ w2′*. *w1′*<*w1* ∧ *w2′*<*w2* ∧ ¬ *isSecO s2* ∧ Δ ∞ *w1′ w2′ s1 s2′ statA sv1 sv2 statO*) ∨
(∃ *w1′*<*w1*. *eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧ *match2-1* Δ *w1′* ∞ *s1 s2 s2′*

statA sv1 sv2 statO) ∨
(*eqSec sv2 s2* ∧ ¬ *isSecV sv1* ∧ *Van.eqAct sv1 sv2* ∧ *match2-12* Δ ∞ ∞
s1 s2 s2′ statA sv1 sv2 statO)
using *m v ii4* **unfolding** *match2-def* **by** *auto*

thus *?thesis*
apply(*elim disjE exE*)
subgoal for *w1′ w2′* **using** *is2* **by** *auto*
subgoal for *w1′* **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of -* ∞])
unfolding *match2-1-def* **apply**(*elim conjE exE*) **subgoal for** *sv2′*
apply(*rule exI*[*of -* [*sv1*]]) **apply**(*rule exI*[*of -* [*sv2*,*sv2′*]])
apply(*rule exI*[*of - statO*])
using *is2 isv24* **unfolding** *χ4a-def*
by (*auto simp* : *sstatA′-def lastt-def*) **.**
subgoal apply(*rule exI*[*of -* ∞]) **apply**(*rule exI*[*of -* ∞])
unfolding *match2-12-def* **apply**(*elim conjE exE*) **subgoal for** *sv1′ sv2′*
apply(*rule exI*[*of -* [*sv1*,*sv1′*]]) **apply**(*rule exI*[*of -* [*sv2*,*sv2′*]])
apply(*rule exI*[*of - sstatO′ statO sv1 sv2*])
using *is2 isv24* **unfolding** *χ4a-def*
by (*auto simp* : *sstatA′-def sstatO′-def lastt-def Van.eqAct-def*) **. .**
qed
qed

definition *χ4b* Δ *w w1 w2 w1′* (*w2′*::*enat*) *s1 s2 tr2 statAA sv1 trv1 sv2 trv2*
statOO ≡
trv1 ≠ [] ∧ *trv2* ≠ [] ∧ (*length trv1* > *Suc 0* ∨ *w1′* < *w1*) ∧
Van.validFromS sv1 trv1 ∧ *Van.validFromS sv2 trv2* ∧
never isSecV (*butlast trv1*) ∧
Van.S trv2 = *Opt.S tr2* ∧
Van.A trv1 = *Van.A trv2* ∧
Δ *w w1′ w2′ s1* (*lastt s2 tr2*) *statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *unwindCond-ex-χ4b-aux*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *tr2NE*: *tr2* ≠ []
and *v4′*: *Opt.validFromS s2* (*tr2* ## *s2′*)
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter4′*: *never isIntO* (*tr2* ## *s2′*)
and *sec*: *never isSecO* (*butlast tr2*) *isSecO* (*lastt s2 tr2*)
shows ∃ *w1′ w2′ trv1 trv2 statOO*. *χ4b* Δ ∞ *w1 w2 w1′ w2′ s1 s2* (*tr2* ## *s2′*)
statA sv1 trv1 sv2 trv2 statOO
proof−
have *v4*: *Opt.validFromS s2 tr2* **and** *s24′*: *validTransO* (*lastt s2 tr2*,*s2′*)
apply (*metis v4′ Opt.validFromS-def Opt.validS-append1 Nil-is-append-conv hd-append2*)
by (*metis Opt.validFromS-def Opt.validS-validTrans append-is-Nil-conv lastt-def*
list.distinct(*1*) *list.sel*(*1*) *tr2NE v4′*)

have *ninter4*: *never isIntO tr2* **and** *nis2'*: ¬ *isIntO s2'*
using *ninter4'* **by** *auto*
obtain *ww ww1 ww2 trv1 trv2 statOO* **where** *χ4*: *χ4* Δ *ww w1 w2 ww1 ww2 s1 s2 tr2 statA sv1 trv1 sv2 trv2 statOO*
using *unwindCond-ex-χ4*[*OF unwind* Δ *r v4 nis1 nis2 ninter4 sec*]
by *auto*

have *trv12NE*: *trv1* ≠ [] *trv2* ≠ [] **using** *χ4* **unfolding** *χ4-def* **by** *auto*

define *ss2 ssv1 ssv2* **where** *ss2*: *ss2* ≡ *lastt s2 tr2*
and *ssv1*: *ssv1* ≡ *lastt sv1 trv1* **and** *ssv2*: *ssv2* ≡ *lastt sv2 trv2*

have *ss2l*: *ss2* = *last tr2* **by** (*simp add*: *lastt-def ss2 tr2NE*)
have *tr2l*: *tr2* = *butlast tr2* @ [*ss2*] **by** (*simp add*: *ss2l tr2NE*)
have *ssv1l*: *ssv1* = *last trv1* **using** *χ4* **unfolding** *χ4-def* **by** (*metis lastt-def ssv1*)
have *trv1l*: *trv1* = *butlast trv1* @ [*ssv1*] **by** (*simp add*: *ssv1l trv12NE*(*1*))
have *ssv2l*: *ssv2* = *last trv2* **using** *χ4* **unfolding** *χ4-def* **by** (*metis lastt-def ssv2*)
have *trv2l*: *trv2* = *butlast trv2* @ [*ssv2*] **by** (*simp add*: *ssv2l trv12NE*(*2*))

have *iss2*[*simp*]: *isSecO ss2* **using** *sec*(*2*) **unfolding** *ss2* **by** *auto*
have *issv2*[*simp*]: *isSecV ssv2* **and** *gissv24*[*simp*]: *getSecO ss2* = *getSecV ssv2*
using *χ4* **unfolding** *χ4-def ssv2 ss2* **by** *auto*

have *niss2*: ¬ *isIntO ss2*
by (*metis list-all-append list-all-simps*(*1*) *ninter4 tr2l*)

have *rss2*: *reachO ss2* **and** *rssv12*: *reachV ssv1 reachV ssv2*
using *Opt.reach-validFromS-reach r ss2l tr2NE v4* **apply** *blast*
unfolding *ssv1 ssv2* **using** *r*(*3*,*4*) **using** *χ4* **unfolding** *χ4-def*
using *Van.reach-validFromS-reach ssv1 ssv2 ssv1l ssv2l* **by** *auto metis+*

have Δ: Δ *ww ww1 ww2 s1 ss2 statA ssv1 ssv2 statOO*
using *χ4* **unfolding** *χ4-def ss2*[*symmetric*] *ssv1*[*symmetric*] *ssv2*[*symmetric*] **by** *auto*

have *s13'*: *validTransO* (*ss2*, *s2'*)
by (*simp add*: *s24' ss2*)

note *vs24* = *s24'*[*unfolded ss2*[*symmetric*]]
obtain *w1' w2' trv1' trv2' statO'* **where**
χ4a: *χ4a* Δ ∞ *ww1 ww2 w1' w2' s1 ss2 s2' statA ssv1 trv1' ssv2 trv2' statO'*
using *unwindCond-ex-χ4a-getSec*[*OF unwind* Δ *r*(*1*) *rss2 rssv12 s13' niss2 iss2 issv2 gissv24*]
by *blast*

have *trv12'NE*: *trv1'* ≠ [] *trv2'* ≠ [] **using** *χ4a* **unfolding** *χ4a-def* **by** *auto*

have [*simp*]: *Van.O* (*butlast trv1* @ *trv1'*) = *Van.O trv1* @ *Van.O trv1'*
using *trv12'NE* **unfolding** *χ4-def Van.O.map-filter Opt.O.map-filter* **apply**(*subst butlast-append*) **by** *simp*

have [*simp*]: *Van.O* (*butlast trv2* @ *trv2'*) = *Van.O trv2* @ *Van.O trv2'*
using *trv12'NE* **unfolding** *χ4-def Van.O.map-filter Opt.O.map-filter* **apply**(*subst butlast-append*) **by** *simp*

have *Van.A trv1'* = *Van.A trv2'* **using** *χ4a* **unfolding** *χ4a-def* **by** *auto*
moreover have *length* (*Van.O trv1'*) = *length* (*Van.A trv1'*) ∧ *length* (*Van.O trv2'*) = *length* (*Van.A trv2'*)
unfolding *Van.A.map-filter Van.O.map-filter* **by** *auto*
ultimately have *length* (*Van.O trv1'*) = *length* (*Van.O trv2'*) **by** *auto*
hence [*simp*]: *Van.O trv1* @ *Van.O trv1'* = *Van.O trv2* @ *Van.O trv2'* ⟷
Van.O trv1 = *Van.O trv2* ∧ *Van.O trv1'* = *Van.O trv2'* **by** *auto*

show *?thesis*
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - butlast trv1* @ *trv1'*]) **apply**(*rule exI*[*of - butlast trv2* @ *trv2'*])
apply(*rule exI*[*of - statO'*])
unfolding *χ4b-def* **apply**(*intro conjI*)
subgoal using *χ4 χ4a* **unfolding** *χ4-def χ4a-def* **by** *auto*
subgoal using *χ4 χ4a* **unfolding** *χ4-def χ4a-def* **by** *auto*
subgoal using *χ4 χ4a* **unfolding** *χ4-def χ4a-def*
by *simp* (*metis Simple-Transition-System.fromS-eq-Nil Van.toS-Nil Van.toS-fromS-nonSingl*

diff-add-inverse2 linorder-not-less order-le-less-trans trans-less-add2 zero-less-diff)

subgoal using *χ4 χ4a* **unfolding** *χ4-def χ4a-def ssv1*
using *Van.validFromS-append* **by** *auto*
subgoal using *χ4 χ4a* **unfolding** *χ4-def χ4a-def ssv2*
using *Van.validFromS-append* **by** *auto*
subgoal using *χ4 χ4a* **unfolding** *χ4-def χ4a-def*
by (*simp add*: *butlast-append*)
subgoal using *χ4 χ4a* **unfolding** *χ4-def χ4a-def* **unfolding** *Van.S.map-filter Opt.S.map-filter*
apply(*subst tr2l*) **apply**(*subst butlast-append*)
by *simp* (*metis Opt.S.map-filter Opt.S.eq-Nil-iff Van.S.map-filter Van.S.eq-Nil-iff sec*(*1*))
subgoal using *χ4 χ4a* **unfolding** *χ4-def χ4a-def Van.A.map-filter Opt.A.map-filter*

apply(*subst trv1l*) **apply**(*subst trv2l*)
apply(*subst butlast-append*) **apply** *simp* **apply**(*subst butlast-append*) **by** *simp*
subgoal using *χ4a trv12'NE tr2NE* **unfolding** *χ4a-def lastt-def* **by** *simp* **.**
qed

lemma *unwindCond-ex-χ4b-aux2*:
assumes *unwind*: *unwindCond* Δ

and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* **and**
r: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *v4′*: *Opt.validFromS s2* (*tr2* @ [*s2′*,*s2′′*])
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter4′*: *never isIntO* (*tr2* @ [*s2′*,*s2′′*])
and *sec*: *never isSecO tr2 isSecO s2′*
shows ∃ *w1′ w2′ trv1 trv2 statOO*. *χ4b* Δ ∞ *w1 w2 w1′ w2′ s1 s2* (*tr2* @ [*s2′*,*s2′′*]) *statA sv1 trv1 sv2 trv2 statOO*
proof−
  have *0*: *lastt s2* (*tr2* ## *s2′*) = *s2′*
  unfolding *lastt-def* **by** *auto*
  show *?thesis*
  using *unwindCond-ex-χ4b-aux*[*OF unwind* Δ *r*, *of tr2* ## *s2′*, *unfolded 0*, *simplified*]
  using *assms* **by** *auto*
qed

definition *χ4′* Δ *w1 w2 w1′* (*w2′*::*enat*) *s1 s2 tr2 s2′ s2′′ statAA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO* ≡
  Van.validFromS sv1 (*trv1* ## *sv1′′*) ∧ *Van.validFromS sv2* (*trv2* ## *sv2′′*) ∧
  never isSecV (*butlast* (*trv1* ## *sv1′′*)) ∧
  Van.S (*trv2* ## *sv2′′*) = *Opt.S* ((*tr2* ## *s2′*) ## *s2′′*) ∧
  Van.A (*trv1* ## *sv1′′*) = *Van.A* (*trv2* ## *sv2′′*) ∧
  trv2 ≠ [] ∧ (*trv1* ≠ [] ∨ *w1′* < *w1*) ∧
  Δ ∞ *w1′ w2′ s1 s2′′ statAA sv1′′ sv2′′ statOO*

proposition *unwindCond-ex-χ4′*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* **and**
r: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *v4′*: *Opt.validFromS s2* ((*tr2* ## *s2′*) ## *s2′′*)
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter4′*: *never isIntO* ((*tr2* ## *s2′*) ## *s2′′*)
and *sec*: *never isSecO tr2 isSecO s2′*
shows ∃ *w1′ w2′ trv1 sv1′′ trv2 sv2′′ statOO*. *χ4′* Δ *w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2′′ statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
using *unwindCond-ex-χ4b-aux2*[*unfolded φ-def*, *unfolded lastt-snoc lastt-snoc2 append-snoc2*, *OF assms*]
unfolding *χ4b-def* **apply**(*elim exE*) **subgoal for** *w1′ w2′ trv1 trv2 statOO*
apply(*cases trv1 rule*: *rev-cases*)
  subgoal by *auto*
  subgoal for *trv1′ sv1′′* **apply**(*cases trv2 rule*: *rev-cases*)
    subgoal by *auto*
    subgoal for *trv2′ sv2′′* **unfolding** *χ4′-def*
    apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
    apply(*rule exI*[*of - trv1′*]) **apply**(*rule exI*[*of - sv1′′*])

apply(*rule exI*[*of - trv2'*]) **apply**(*rule exI*[*of - sv2''*])
apply(*rule exI*[*of - statOO*])
by *simp* (*metis Opt.S.Nil-iff Opt.S.eq-Nil-iff*(*1*) *Van.S.simps*(*4*) *butlast-append*

list.discI list-all-append sec(*2*) *self-append-conv2*) . . .

definition *ω4* Δ *w1 w2 w1' (w2'::enat) s1 s2 s2' statAA sv1 trv1 sv1' sv2 trv2 sv2' statOO* ≡
Van.validFromS sv1 (*trv1* ## *sv1'*) ∧ *Van.validFromS sv2* (*trv2* ## *sv2'*) ∧
never isSecV trv1 ∧ *never isSecV trv2* ∧
Van.A (*trv1* ## *sv1'*) = *Van.A* (*trv2* ## *sv2'*) ∧
(*trv1* ≠ [] ∨ *w1'* < *w1*) ∧ (*trv2* ≠ [] ∨ *w2'* < *w2*) ∧
Δ ∞ *w1' w2' s1 s2' statAA sv1' sv2' statOO*

proposition *unwindCond-ex-ω4*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r34*: *reachO s1 reachO s2* **and** *r12*: *reachV sv1 reachV sv2*
and *nis1*: ¬ *isIntO s1*
and *v4*: *validTransO* (*s2*,*s2'*)
and *nis2*: ¬ *isIntO s2* ¬ *isIntO s2'* ¬ *isSecO s2*
shows ∃ *w1' w2' trv1 sv1' trv2 sv2' statOO*. *ω4* Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*
using Δ *r12*
proof(*induction w arbitrary*: *w1 w2 sv1 sv2 statO rule*: *less-induct*)
case (*less w w1 w2 sv1 sv2 statO*)
note Δ = ‹Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*›
note *r12* = *less.prems*(*2*,*3*)
note *r1* = *r12*(*1*) **note** *r2* = *r12*(*2*)
note *r* = *r34 r12*

have *f34*: *finalO s1* = *finalO s2* ∧ *finalV sv1* = *finalO s1* ∧ *finalV sv2* = *finalO s2*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

have *proact-match*: (∃ *v*<*w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*) ∨ *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
using Δ *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*
show *?case* **using** *proact-match* **proof** *safe*
fix *v* **assume** *v*: *v* < *w*
assume *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
thus *?thesis* **unfolding** *proact-def* **proof** *safe*
assume *sv1*: ¬ *isSecV sv1* ¬ *isIntV sv1* **and** *move-1* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv1'* **where** *0*: *validTransV* (*sv1*, *sv1'*) **and** Δ: Δ *v w1 w2 s1 s2 statA sv1' sv2 statO*
unfolding *move-1-def* **by** *auto*

have *r1'*: *reachV sv1'* **using** *r1 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1' w2' trv1 sv1'' trv2 sv2' statOO* **where**
ω4: *ω4 Δ w1 w2 w1' w2' s1 s2 s2' statA sv1' trv1 sv1'' sv2 trv2 sv2' statOO*

using *less*(*1*)[*OF v Δ r1' r2*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv1''*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2'*])
using *ω4 0 sv1* **unfolding** *ω4-def* **by** *auto*
next
assume *sv2*: ¬ *isSecV sv2* ¬ *isIntV sv2* **and** *move-2 Δ v w1 w2 s1 s2 statA sv1 sv2 statO*
then obtain *sv2'*
where *0*: *validTransV* (*sv2*,*sv2'*)
and Δ: *Δ v w1 w2 s1 s2 statA sv1 sv2' statO*
unfolding *move-2-def* **by** *auto*
have *r2'*: *reachV sv2'* **using** *r2 0* **by** (*metis Van.reach.Step fst-conv snd-conv*)
obtain *w1' w2' trv1 sv1' trv2 sv2'' statOO* **where**
ω4: *ω4 Δ w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2' trv2 sv2'' statOO*

using *less*(*1*)[*OF v Δ r1 r2'*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1'*])
apply(*rule exI*[*of - sv2 # trv2*]) **apply**(*rule exI*[*of - sv2''*])
using *ω4 0 sv2* **unfolding** *ω4-def* **by** *auto*
next
assume *sv1*: ¬ *isSecV sv1* **and** *sv2*: ¬ *isSecV sv2* **and**
move-12 Δ v w1 w2 s1 s2 statA sv1 sv2 statO **and**
sv12: *Van.eqAct sv1 sv2*
then obtain *sv1' sv2' statO'*
where *statO'*: *statO' = sstatO' statO sv1 sv2*
and *0*: *validTransV* (*sv1*,*sv1'*) *validTransV* (*sv2*,*sv2'*)
and Δ: *Δ v w1 w2 s1 s2 statA sv1' sv2' statO'*
unfolding *move-12-def* **by** *auto*
have *r1'*: *reachV sv1'* **and** *r2'*: *reachV sv2'* **using** *r1 r2 0*
by (*metis Van.reach.Step fst-conv snd-conv*)+
obtain *w1' w2' trv1 sv1'' trv2 sv2'' statOO* **where**
ω4: *ω4 Δ w1 w2 w1' w2' s1 s2 s2' statA sv1' trv1 sv1'' sv2' trv2 sv2'' statOO*

using *less*(*1*)[*OF v Δ r1' r2'*] **by** *auto*
show *?thesis* **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - sv1 # trv1*]) **apply**(*rule exI*[*of - sv1''*])
apply(*rule exI*[*of - sv2 # trv2*]) **apply**(*rule exI*[*of - sv2''*])
using *ω4 0 sv1 sv2 sv12* **unfolding** *ω4-def statO'* **by** (*auto simp*: *Van.eqAct-def*)
qed
next
assume *m*: *match Δ w1 w2 s1 s2 statA sv1 sv2 statO*
have *m*: *match2 Δ w1 w2 s1 s2 statA sv1 sv2 statO* **using** *m* **unfolding** *match-def* **by** *auto*

have (∃ *w1′ w2′*. *w1′*<*w1* ∧ *w2′*<*w2* ∧ ¬ *isSecO s2* ∧ Δ ∞ *w1′ w2′ s1 s2′ statA sv1 sv2 statO*) ∨
(∃ *w1′*<*w1*. *eqSec sv2 s2* ∧ ¬ *isIntV sv2* ∧ *match2-1* Δ *w1′* ∞ *s1 s2 s2′ statA sv1 sv2 statO*) ∨
¬ *isSecV sv1* ∧ *eqSec sv2 s2* ∧ *Van.eqAct sv1 sv2* ∧ *match2-12* Δ ∞ ∞ *s1 s2 s2′ statA sv1 sv2 statO*
using *m v4 nis2* **unfolding** *match2-def* **by** *auto*

thus *?thesis*
apply(*elim disjE exE*)
subgoal for *w1′ w2′* **apply**(*rule exI*[*of* - *w1′*]) **apply**(*rule exI*[*of* - *w2′*])
apply(*rule exI*[*of* - []]) **apply**(*rule exI*[*of* - *sv1*])
apply(*rule exI*[*of* - []]) **apply**(*rule exI*[*of* - *sv2*])
apply(*rule exI*[*of* - *statO*]) **unfolding** *ω4-def*
by *auto*
subgoal for *w1′* **apply**(*rule exI*[*of* - *w1′*]) **apply**(*rule exI*[*of* - ∞])
unfolding *match2-1-def* **apply**(*elim conjE exE*) **subgoal for** *sv2′*
apply(*rule exI*[*of* - []]) **apply**(*rule exI*[*of* - *sv1*])
apply(*rule exI*[*of* - [*sv2*]]) **apply**(*rule exI*[*of* - *sv2′*])
apply(*rule exI*[*of* - *statO*])
unfolding *ω4-def* **using** *nis2*(*3*) **by** (*auto simp*: *eqSec-def*) **.**
subgoal **apply**(*rule exI*[*of* - ∞]) **apply**(*rule exI*[*of* - ∞])
unfolding *match2-12-def* **apply**(*elim conjE exE*) **subgoal for** *sv1′ sv2′*
apply(*rule exI*[*of* - [*sv1*]]) **apply**(*rule exI*[*of* - *sv1′*])
apply(*rule exI*[*of* - [*sv2*]]) **apply**(*rule exI*[*of* - *sv2′*])
apply(*rule exI*[*of* - *sstatO′ statO sv1 sv2*])
unfolding *ω4-def* **using** *nis2*(*3*) **apply** (*auto simp*: *eqSec-def sstatA′-def sstatO′-def lastt-def Van.eqAct-def*) **. . .**
qed
qed

definition *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′* ≡
ltr1 = *lappend* (*llist-of* (*tr1* ## *s1′*)) (*s1″* $ *ltr1′*) ∧
ltr2 = *lappend* (*llist-of* (*tr2* ## *s2′*)) (*s2″* $ *ltr2′*) ∧
Opt.validFromS s1 ((*tr1* ## *s1′*) ## *s1″*) ∧ *Opt.validFromS s2* ((*tr2* ## *s2′*) ## *s2″*) ∧
never isIntO tr1 ∧ *never isIntO tr2* ∧
isIntO s1′ ∧ *isIntO s2′* ∧ *getActO s1′* = *getActO s2′* ∧
Opt.lvalidFromS s1″ (*s1″* $ *ltr1′*) ∧ *Opt.lcompletedFrom s1″* (*s1″* $ *ltr1′*) ∧
Opt.lvalidFromS s2″ (*s2″* $ *ltr2′*) ∧ *Opt.lcompletedFrom s2″* (*s2″* $ *ltr2′*) ∧
Opt.lA (*s1″* $ *ltr1′*) = *Opt.lA* (*s2″* $ *ltr2′*)

lemma *isIntO-φφ*:
assumes *vltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *vltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A*: *Opt.lA ltr1* = *Opt.lA ltr2* **and** *inter3*: ¬ *lnever isIntO ltr1*
shows ∃ *tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′*. *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′*
proof−
  have *03*: ∃ *s*∈*lset ltr1*. *isIntO s* **using** *inter3* **unfolding** *llist.pred-set* **by** *auto*
  define *ttr1* **where** *ttr1*: *ttr1* ≡ *ltakeUntil isIntO ltr1*
  define *lltr1′* **where** *lltr1′*: *lltr1′* ≡ *ldropUntil isIntO ltr1*
  have *ltr1*: *ltr1* = *lappend* (*llist-of ttr1*) *lltr1′*
  unfolding *ttr1 lltr1′ lappend-ltakeUntil-ldropUntil*[*OF 03*] **..**
  have *13*: *ttr1* ≠ [] ∧ *never isIntO* (*butlast ttr1*) ∧ *isIntO* (*last ttr1*)
  unfolding *ttr1*
  using *ltakeUntil-last*[*OF 03*] *ltakeUntil-not-Nil*[*OF 03*] *ltakeUntil-never-butlast*[*OF 03*] **by** *simp*
  then obtain *tr1 s1′* **where** *ttr1-eq*: *ttr1* = *tr1* ## *s1′*
  using *rev-exhaust* **by** *blast*
  hence *tr1s1′*: *never isIntO tr1 isIntO s1′* **using** *13* **by** *auto*
  have *lfinite ltr1* ⟹ *s1′* ≠ *llast ltr1*
  by (*metis Opt.final-not-isInt Opt.lcompletedFrom-def llast-last-llist-of tr1s1′*(*2*) *vltr1*(*2*))
  hence *ne*: *lltr1′* ≠ [[]]
  using *ltr1* **unfolding** *ttr1-eq* **by** *auto*
  then obtain *s1″ ltr1′* **where** *lltr1′*: *lltr1′* = *s1″* $ *ltr1′*
  by (*meson llist.exhaust*)
  have [*simp*]: *filter isIntO tr1* = []
    by (*metis never-Nil-filter tr1s1′*(*1*))
  have *clltr1′*: *Opt.lcompletedFrom s1 lltr1′*
  by (*metis Opt.lcompletedFrom-def lfinite-lappend lfinite-llist-of llast-lappend-LCons*

  llast-last-llist-of lltr1′ ltr1 ne vltr1(*2*))

  have *inter4*: ¬ *lnever isIntO ltr2* **using** *A inter3*
  by (*metis Opt.lA.eq-LNil-iff Opt.lO Opt.lO.eq-LNil-iff lfiltermap-LNil-never lfiltermap-lmap-lfilter vltr1*(*2*) *vltr2*(*2*))
  have *04*: ∃ *s*∈*lset ltr2*. *isIntO s* **using** *inter4* **unfolding** *llist.pred-set* **by** *auto*
  define *ttr2* **where** *ttr2*: *ttr2* ≡ *ltakeUntil isIntO ltr2*
  define *lltr2′* **where** *lltr2′*: *lltr2′* ≡ *ldropUntil isIntO ltr2*
  have *ltr2*: *ltr2* = *lappend* (*llist-of ttr2*) *lltr2′*
  unfolding *ttr2 lltr2′ lappend-ltakeUntil-ldropUntil*[*OF 04*] **..**
  have *14*: *ttr2* ≠ [] ∧ *never isIntO* (*butlast ttr2*) ∧ *isIntO* (*last ttr2*)
  unfolding *ttr2*
  using *ltakeUntil-last*[*OF 04*] *ltakeUntil-not-Nil*[*OF 04*] *ltakeUntil-never-butlast*[*OF 04*] **by** *simp*
  then obtain *tr2 s2′* **where** *ttr2-eq*: *ttr2* = *tr2* ## *s2′*
  using *rev-exhaust* **by** *blast*
  hence *tr2s2′*: *never isIntO tr2 isIntO s2′* **using** *14* **by** *auto*

have *lfinite ltr2* ⟹ *s2'* ≠ *llast ltr2*
by (*metis Opt.final-not-isInt Opt.lcompletedFrom-def llast-last-llist-of tr2s2'*(*2*) *vltr2*(*2*))
hence *ne*: *lltr2'* ≠ [[]]
using *ltr2* **unfolding** *ttr2-eq* **by** *auto*
then obtain *s2'' ltr2'* **where** *lltr2'*: *lltr2'* = *s2''* $ *ltr2'*
by (*meson llist.exhaust*)
have [*simp*]: *filter isIntO tr2* = []
by (*metis never-Nil-filter tr2s2'*(*1*))
have *clltr2'*: *Opt.lcompletedFrom s2 lltr2'*
by (*metis Opt.lcompletedFrom-def lfinite-lappend lfinite-llist-of llast-lappend-LCons*

llast-last-llist-of lltr2' ltr2 ne vltr2(*2*))

have *AA*: *Opt.lA lltr1'* = *Opt.lA lltr2'*
unfolding *Opt.lA*[*OF clltr1'*] *Opt.lA*[*OF clltr2'*]
using *A*[*unfolded Opt.lA*[*OF vltr1*(*2*)] *Opt.lA*[*OF vltr2*(*2*)]] *tr1s1' tr2s2'*
unfolding *ltr1 ltr2 ttr1-eq ttr2-eq*
unfolding *lfilter-lappend-llist-of* **by** *simp*

show *?thesis* **apply**(*rule exI*[*of - tr1*]) **apply**(*rule exI*[*of - s1'*])
apply(*rule exI*[*of - s1''*]) **apply**(*rule exI*[*of - ltr1'*])
apply(*rule exI*[*of - tr2*]) **apply**(*rule exI*[*of - s2'*])
apply(*rule exI*[*of - s2''*]) **apply**(*rule exI*[*of - ltr2'*])
unfolding *φφ-def* **apply**(*intro conjI*)
subgoal unfolding *ltr1 ttr1-eq lltr1'* **..**
subgoal unfolding *ltr2 ttr2-eq lltr2'* **..**
subgoal using *vltr1*(*1*) **unfolding** *ltr1 ttr1-eq lltr1'*
by (*simp add*: *Opt.lvalidFromS-lappend-finite lappend-llist-of-LCons*)
subgoal using *vltr2*(*1*) **unfolding** *ltr2 ttr2-eq lltr2'*
by (*simp add*: *Opt.lvalidFromS-lappend-finite lappend-llist-of-LCons*)
subgoal using *tr1s1'* **by** *simp*
subgoal using *tr2s2'* **by** *simp*
subgoal using *tr1s1'* **by** *simp*
subgoal using *tr2s2'* **by** *simp*
subgoal using *A*[*unfolded Opt.lA*[*OF vltr1*(*2*)] *Opt.lA*[*OF vltr2*(*2*)]]
tr1s1' tr2s2'
unfolding *ltr1 ttr1-eq ltr2 ttr2-eq lltr1' lltr2'*
unfolding *lfilter-lappend-llist-of* **by** *simp*
subgoal using *vltr1*(*1*) **unfolding** *ltr1 ttr1-eq lltr1'*
using *Opt.lvalidFromS-lappend-LCons* **by** *blast*
subgoal using *vltr1*(*2*) **unfolding** *ltr1 ttr1-eq lltr1'*
by (*metis Opt.lcompletedFrom-def lfinite-lappend lfinite-llist-of*
llast-lappend-LCons llast-last-llist-of llist.distinct(*1*))
subgoal using *vltr2*(*1*) **unfolding** *ltr2 ttr2-eq lltr2'*
using *Opt.lvalidFromS-lappend-LCons* **by** *blast*
subgoal using *vltr2*(*2*) **unfolding** *ltr2 ttr2-eq lltr2'*
by (*metis Opt.lcompletedFrom-def lfinite-lappend lfinite-llist-of*
llast-lappend-LCons llast-last-llist-of llist.distinct(*1*))

```
    subgoal using AA unfolding lltr1' lltr2' . .
qed

definition χχ s1 ltr1 tr1 s1' s1'' ltr1' ≡
   ltr1 = lappend (llist-of (tr1 ## s1')) (s1'' $ ltr1') ∧
   Opt.validFromS s1 ((tr1 ## s1') ## s1'') ∧
   never isIntO tr1 ∧ ¬ isIntO s1' ∧ ¬ isIntO s1'' ∧
   never isSecO tr1 ∧ isSecO s1' ∧
   Opt.lvalidFromS s1'' (s1'' $ ltr1') ∧ Opt.lcompletedFrom s1'' (s1'' $ ltr1')

lemma isSecO-χχ:
assumes vltr1: Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1
and inter: lnever isIntO ltr1 and isec: ¬ lnever isSecO ltr1
shows ∃ tr1 s1' s1'' ltr1'. χχ s1 ltr1 tr1 s1' s1'' ltr1'
proof−
  have 0: ∃ s∈lset ltr1. isSecO s using isec unfolding llist.pred-set by auto
  define ttr1 where ttr1: ttr1 ≡ ltakeUntil isSecO ltr1
  define lltr1' where lltr1': lltr1' ≡ ldropUntil isSecO ltr1
  have ltr1: ltr1 = lappend (llist-of ttr1) lltr1'
  unfolding ttr1 lltr1' lappend-ltakeUntil-ldropUntil[OF 0] ..
  have 1: ttr1 ≠ [] ∧ never isSecO (butlast ttr1) ∧ isSecO (last ttr1)
  unfolding ttr1
  using ltakeUntil-last[OF 0] ltakeUntil-not-Nil[OF 0] ltakeUntil-never-butlast[OF
0] by simp
  then obtain tr1 s1' where ttr1-eq: ttr1 = tr1 ## s1'
  using rev-exhaust by blast
  hence tr1s1': never isSecO tr1 isSecO s1' using 1 by auto
  have 2: never isIntO tr1 ∧ ¬ isIntO s1' ∧ lnever isIntO lltr1'
  using inter unfolding ltr1 ttr1-eq
  unfolding llist-all-lappend-llist-of list-all-append by simp
  have lfinite ltr1 ⟹ s1' ≠ llast ltr1
  by (metis Opt.final-not-isSec Opt.lcompletedFrom-def llast-last-llist-of tr1s1'(2)
vltr1(2))
  hence ne: lltr1' ≠ [[]]
  using ltr1 unfolding ttr1-eq by auto
  then obtain s1'' ltr1' where lltr1': lltr1' = s1'' $ ltr1'
  by (meson llist.exhaust)
  show ?thesis apply(rule exI[of - tr1]) apply(rule exI[of - s1'])
  apply(rule exI[of - s1'']) apply(rule exI[of - ltr1'])
  unfolding χχ-def apply(intro conjI)
    subgoal unfolding ltr1 ttr1-eq lltr1' ..
    subgoal using vltr1(1) unfolding ltr1 ttr1-eq lltr1'
    by (simp add: Opt.lvalidFromS-lappend-finite lappend-llist-of-LCons)
    subgoal using 2 by simp
    subgoal using 2 by simp
    subgoal using 2 unfolding lltr1' by simp
    subgoal using tr1s1' by simp
    subgoal using tr1s1' by simp
    subgoal using vltr1(1) unfolding ltr1 ttr1-eq lltr1'
```

using *Opt.lvalidFromS-lappend-LCons* **by** *blast*
subgoal using *vltr1*(*2*) **unfolding** *ltr1 ttr1-eq lltr1'*
by (*metis Opt.lcompletedFrom-def ne lfinite-lappend lfinite-llist-of llast-lappend-LCons llast-last-llist-of lltr1'*) **.**
qed

type-synonym (*'stA*,*'stO*) *tuple34* =
enat × *enat* ×
'stA × *'stA llist* ×
'stA × *'stA llist* ×
status ×
'stO × *'stO* × *status*

type-synonym (*'stA*,*'stO*) *tuple12* =
'stO list × *'stO* × *'stO list* × *'stO* × *status* × *status*

context
fixes Δ :: *enat* ⇒ *enat* ⇒ *enat* ⇒ *'stateO* ⇒ *'stateO* ⇒ *status* ⇒ *'stateV* ⇒ *'stateV* ⇒ *status* ⇒ *bool*
begin

fun *isn* :: *turn* × (*'stateO*,*'stateV*) *tuple34* ⇒ *bool*
where
isn (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ⟷ *ltr1* = [[]] ∧ *ltr2* = [[]]

fun *h-t* ::
turn × (*'stateO*,*'stateV*)*tuple34* ⇒
(*'stateO*,*'stateV*)*tuple12* ×
turn × (*'stateO*,*'stateV*)*tuple34*
where
h-t (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
(*if trn* = *L*
then if lnever isSecO ltr1
then let (*s1'*,*ltr1'*) = (*lhd* (*ltl ltr1*), *ltl ltr1*)
in let (*w1'*,*w2'*,*trv1*,*sv1'*,*trv2*,*sv2'*,*statOO*) =
(*SOME k. case k of* (*w1'*,*w2'*,*trv1*,*sv1'*,*trv2*,*sv2'*,*statOO*) ⇒
ω3 Δ *w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*)
in ((*trv1*,*sv1'*,*trv2*,*sv2'*,*statA*,*statOO*),
(*if trv1* = [] *then L else R*,
w1',*w2'*,*s1'*,*ltr1'*,*s2*,*ltr2*,*statA*,*sv1'*,*sv2'*,*statOO*))
else
let (*tr1*,*s1'*,*s1''*,*ltr1'*) =
(*SOME k. case k of* (*tr1*,*s1'*,*s1''*,*ltr1'*) ⇒

χχ s1 ltr1 tr1 s1′ s1′′ ltr1′)
in let (w1′,w2′,trv1,sv1′′,trv2,sv2′′,statOO) =
(SOME k′. case k′ of (w1′,w2′,trv1,sv1′′,trv2,sv2′′,statOO) ⇒
χ3′ Δ w1 w2 w1′ w2′ s1 tr1 s1′ s1′′ s2 statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO)
in ((trv1,sv1′′,trv2,sv2′′,statA,statOO),
(R,w1′,w2′,s1′′,s1′′ $ ltr1′,s2,ltr2,statA,sv1′′,sv2′′,statOO))
—
else if lnever isSecO ltr2
then let (s2′,ltr2′) = (lhd (ltl ltr2), ltl ltr2)
in let (w1′,w2′,trv1,sv1′,trv2,sv2′,statOO) =
(SOME k. case k of (w1′,w2′,trv1,sv1′,trv2,sv2′,statOO) ⇒
ω4 Δ w1 w2 w1′ w2′ s1 s2 s2′ statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO)
in ((trv1,sv1′,trv2,sv2′,statA,statOO),
(if trv2 = [] then R else L,
w1′,w2′,s1,ltr1,s2′,ltr2′,statA,sv1′,sv2′,statOO))
else
let (tr2,s2′,s2′′,ltr2′) =
(SOME k. case k of (tr2,s2′,s2′′,ltr2′) ⇒
χχ s2 ltr2 tr2 s2′ s2′′ ltr2′)
in let (w1′,w2′,trv1,sv1′′,trv2,sv2′′,statOO) =
(SOME k′. case k′ of (w1′,w2′,trv1,sv1′′,trv2,sv2′′,statOO) ⇒
χ4′ Δ w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2′′ statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO)
in ((trv1,sv1′′,trv2,sv2′′,statA,statOO),
(L,w1′,w2′,s1, ltr1, s2′′,s2′′ $ ltr2′,statA,sv1′′,sv2′′,statOO))
)

declare *h-t.simps*[*simp del*]

definition *h* ≡ *fst o h-t*
definition *t* ≡ *snd o h-t*

fun *econd* **where** *econd* (*trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO*) =
(*llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)

fun *e* **where** *e* (*trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO*) = [[([*sv1*],*sv1*,[*sv2*],*sv2*,*statA*,*statO*)]]

definition *f* :: *turn* × (′*stateO*,′*stateV*)*tuple34* ⇒ (′*stateO*,′*stateV*)*tuple12 llist*
where *f* ≡ *ccorec-llist isn h econd e t*

lemma *f-LNil*:
ltr1 = [[]] ⟹ *ltr2* = [[]] ⟹ *f* (*trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO*) = [[]]
unfolding *f-def* **apply**(*subst llist-ccorec*(*1*)) **by** *auto*

lemma *f-length-1*:

assumes *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] *llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*
shows *f* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[([*sv1*],*sv1*,[*sv2*],*sv2*,*statA*,*statO*)]]

using *assms* **unfolding** *f-def* **apply**(*subst llist-ccorec*(*2*))
  subgoal unfolding *e.simps lnull-def* **by** *auto*
  subgoal by *auto*
  subgoal unfolding *econd.simps* **by** *simp* **.**

lemma *f-length-ge1*:
assumes *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
shows *f* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
  LCons (*h* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) (*f* (*t* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*sta*
))
proof−
  show *?thesis* **using** *assms* **unfolding** *f-def* **apply**(*subst llist-ccorec*(*2*))
  subgoal unfolding *e.simps lnull-def* **by** *auto*
  subgoal by *auto*
  subgoal unfolding *econd.simps* **by** *auto* **.**
qed

definition *lltrv1* :: *turn* × (′*stateO*,′*stateV*)*tuple34* ⇒ ′*stateV llist* **where**
lltrv1 trn-tp = *lconcat* (*lmap* (λ(*trv1*,*sv1*′′,*trv2*,*sv2*′′,*statAA*,*statOO*). *llist-of trv1*)
(*f trn-tp*))

definition *then1* :: *turn* × (′*stateO*,′*stateV*)*tuple34* ⇒ *nat* **where**
then1 trn-tp = *firstNC* (*lmap* (λ(*trv1*,*sv1*′′,*trv2*,*sv2*′′,*statAA*,*statOO*). *trv1*) (*f trn-tp*))

definition *lltrv2* :: *turn* × (′*stateO*,′*stateV*)*tuple34* ⇒ ′*stateV llist* **where**
lltrv2 trn-tp = *lconcat* (*lmap* (λ(*trv1*,*sv1*′′,*trv2*,*sv2*′′,*statAA*,*statOO*). *llist-of trv2*)
(*f trn-tp*))

definition *then2* :: *turn* × (′*stateO*,′*stateV*)*tuple34* ⇒ *nat* **where**
then2 trn-tp = *firstNC* (*lmap* (λ(*trv1*,*sv1*′′,*trv2*,*sv2*′′,*statAA*,*statOO*). *trv2*) (*f trn-tp*))

lemma *lltrv1-ne-imp*:
assumes *lltrv1 trn-tp* ≠ [[]]
shows ∃ *trv1 sv1*′′ *trv2 sv2*′′ *statAA statOO*. (*trv1*,*sv1*′′,*trv2*,*sv2*′′,*statAA*,*statOO*)
∈ *lset* (*f trn-tp*) ∧
    trv1 ≠ []
using *assms* **unfolding** *lltrv1-def* **unfolding** *lconcat-eq-LNil-iff* **by** *force*

lemma *lltrv2-ne-imp*:
assumes *lltrv2 trn-tp* ≠ [[]]
shows ∃ *trv1 sv1*′′ *trv2 sv2*′′ *statAA statOO*. (*trv1*,*sv1*′′,*trv2*,*sv2*′′,*statAA*,*statOO*)
∈ *lset* (*f trn-tp*) ∧
    trv2 ≠ []

using *assms* **unfolding** *lltrv2-def* **unfolding** *lconcat-eq-LNil-iff* **by** *force*

lemma *lltrv1-LNil*[*simp*]:
ltr1 = [[]] $\Longrightarrow$ *ltr2* = [[]] $\Longrightarrow$ *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[]]
unfolding *lltrv1-def f-LNil* **by** *simp*
lemma *lltrv2-LNil*[*simp*]:
ltr1 = [[]] $\Longrightarrow$ *ltr2* = [[]] $\Longrightarrow$ *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[]]
unfolding *lltrv2-def f-LNil* **by** *simp*

lemma *lltrv1-lnever*[*simp*]:
assumes *ltr1* $\neq$ [[]] $\vee$ *ltr2* $\neq$ [[]] *llength ltr1* $\leq$ *Suc 0* $\vee$ *llength ltr2* $\leq$ *Suc 0*
shows *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[*sv1*]]
unfolding *lltrv1-def* **using** *f-length-1*[*OF assms*] **by** *auto*

lemma *lltrv2-lnever*[*simp*]:
assumes *ltr1* $\neq$ [[]] $\vee$ *ltr2* $\neq$ [[]] *llength ltr1* $\leq$ *Suc 0* $\vee$ *llength ltr2* $\leq$ *Suc 0*
shows *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[*sv2*]]
unfolding *lltrv2-def* **using** *f-length-1*[*OF assms*] **by** *auto*

lemma *h-t-lnever-L*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *l'*: *lnever isIntO ltr1* ¬ *isIntO s2*
and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *L* *lnever isSecO ltr1*
shows ∃ *w1' w2' s1' ltr1' trv1 sv1' trv2 sv2' statOO*.
ltr1 = *s1* $ *ltr1'* ∧ *validTransO* (*s1*,*s1'*) ∧
Opt.lvalidFromS s1' ltr1' ∧ *Opt.lcompletedFrom s1' ltr1'* ∧ *lnever isIntO ltr1'* ∧

ω*3* Δ *w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO* ∧
h-t (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
((*trv1*,*sv1'*,*trv2*,*sv2'*,*statA*,*statOO*),
(*if trv1* = [] *then L else R*,
w1',*w2'*, *s1'*, *ltr1'*,*s2*,*ltr2*,*statA*,*sv1'*,*sv2'*,*statOO*))
proof−
have *s1*: ¬ *isIntO s1* **using** *l' ltr1*

by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil lhd-LCons llist.exhaust llist.pred-inject*(*2*))

obtain *ltr1'* **where** *ltr13*: *ltr1* = *s1* \$ *ltr1'*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel ltr1*(*1*) *ltr1*(*2*))
hence *ltr1'*: *ltr1'* = *ltl ltr1* **by** *auto*
have *ltr1'ne*: *ltr1'* ≠ [[]] **using** *len*(*1*) **unfolding** *ltr13*
by (*metis One-nat-def llength-LCons llength-LNil one-eSuc one-enat-def order-less-irrefl*)
define *s1'* **where** *s1'*: *s1'* = *lhd* (*ltl ltr1*)
have *v3*: *validTransO* (*s1*,*s1'*) **and** *vv3*: *Opt.lvalidFromS s1' ltr1' Opt.lcompletedFrom s1' ltr1'*
using *ltr1 ltr1'ne* **unfolding** *ltr13 s1'*
by (*metis Opt.lcompletedFrom-LCons Opt.lcompletedFrom-def Opt.lvalidFromS-Cons-iff ltr1' ltr13*)+

have *is1'*: ¬ *isIntO s1'* **and** *lnever isIntO ltr1'*
using *l'*(*1*) **unfolding** *ltr13*
by (*metis llist.exhaust-sel llist.pred-inject*(*2*) *ltr1' ltr1'ne s1'*)+

have *iss1*: ¬ *isSecO s1*
using *l*(*2*) *ltr13* **by** *auto*

obtain *w1' w2' trv1 sv1' trv2 sv2' statOO*
where *ω3*: *ω3* Δ *w1 w2 w1' w2' s1* (*lhd* (*ltl ltr1*)) *s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*
using *unwindCond-ex-ω3*[*OF unw* Δ *r v3 s1 is1' iss1 l'*(*2*)] *s1'* **by** *auto*

define *tp'* **where**
tp' = (*SOME k'*. *case k' of* (*w1'*,*w2'*,*trv1*,*sv1'*,*trv2*,*sv2'*,*statOO*) ⇒
ω3 Δ *w1 w2 w1' w2' s1* (*lhd* (*ltl ltr1*)) *s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*)

have *1*: *case tp' of* (*w1'*,*w2'*,*trv1*,*sv1'*,*trv2*,*sv2'*,*statOO*) ⇒
ω3 Δ *w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*
using *ω3* **unfolding** *tp'-def s1'* **apply**− **apply**(*rule someI-ex*)
apply(*rule exI*[*of* - (*w1'*,*w2'*,*trv1*,*sv1'*,*trv2*,*sv2'*,*statOO*)]) **by** *auto*

obtain *w1' w2' trv1 sv1' trv2 sv2' statOO* **where**
tp': *tp'* = (*w1'*,*w2'*,*trv1*,*sv1'*,*trv2*,*sv2'*,*statOO*) **by**(*cases tp'*, *auto*)

have *ω3*: *ω3* Δ *w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*

using *1* **unfolding** *tp'* **by** *auto*

show *?thesis*

apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1'*]) **apply**(*rule exI*[*of - ltr1'*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1'*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal using *len l* **unfolding** *h-t.simps* **apply** *simp*
unfolding *tp'-def*[*symmetric*] *tp' s1' ltr1'* **by** *simp* **.**
qed

lemma *lltrv1-lltrv2-lnever-L*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *l'*: *lnever isIntO ltr1* ¬ *isIntO s2*
and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *L lnever isSecO ltr1*
shows ∃ *w1' w2' s1' ltr1' trv1 sv1' trv2 sv2' statOO*.
ltr1 = *s1* $ *ltr1'* ∧ *validTransO* (*s1*,*s1'*) ∧
Opt.lvalidFromS s1' ltr1' ∧ *Opt.lcompletedFrom s1' ltr1'* ∧ *lnever isIntO ltr1'* ∧

ω3 Δ *w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO* ∧
lltrv1 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv1*) (*lltrv1* (*if trv1* = [] *then L else R*,
w1',*w2'*,*s1'*,*ltr1'*,*s2*,*ltr2*,*statA*,*sv1'*,*sv2'*,*statOO*)) ∧
lltrv2 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv2*) (*lltrv2* (*if trv1* = [] *then L else R*,
w1',*w2'*,*s1'*,*ltr1'*,*s2*,*ltr2*,*statA*,*sv1'*,*sv2'*,*statOO*))
proof−
show *?thesis*
using *h-t-lnever-L*[*OF assms*] **apply**(*elim exE*)
subgoal for *w1' w2' s1' ltr1' trv1 sv1' trv2 sv2' statOO*
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1'*]) **apply**(*rule exI*[*of - ltr1'*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1'*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*

subgoal by *simp*
subgoal by *simp*
subgoal unfolding *lltrv1-def* **apply**(*subst f-length-ge1*[*OF len*])
unfolding *h-def t-def* **by** *auto*
subgoal unfolding *lltrv2-def* **apply**(*subst f-length-ge1*[*OF len*])
unfolding *h-def t-def* **by** *auto* **. .**
qed

lemma *h-t-not-lnever-L*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *l′*: *lnever isIntO ltr1* ¬ *isIntO s2*
and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *L* ¬ *lnever isSecO ltr1*
shows ∃ *w1′ w2′ tr1 s1′ s1″ ltr1′ trv1 sv1″ trv2 sv2″ statOO*.
χχ s1 ltr1 tr1 s1′ s1″ ltr1′ ∧
χ3′ Δ *w1 w2 w1′ w2′ s1 tr1 s1′ s1″ s2 statA sv1 trv1 sv1″ sv2 trv2 sv2″ statOO* ∧
h-t (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
((*trv1*,*sv1″*,*trv2*,*sv2″*,*statA*,*statOO*),
(*R*,*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2*,*ltr2*,*statA*,*sv1″*,*sv2″*,*statOO*))
proof−
have *s1*: ¬ *isIntO s1* **using** *l′ ltr1*
by (*metis Opt.lvalidFromS-def l*(*2*) *lhd-LCons llist.exhaust llist.pred-inject*(*1*) *llist.pred-inject*(*2*))

obtain *tr1 s1′ s1″ ltr1′*
where *χχ*: *χχ s1 ltr1 tr1 s1′ s1″ ltr1′*
using *isSecO-χχ*[*OF ltr1 l′*(*1*) *l*(*2*)] **by** *auto*

define *tp* **where**
tp = (*SOME k. case k of* (*tr1*,*s1′*,*s1″*,*ltr1′*) ⇒
χχ s1 ltr1 tr1 s1′ s1″ ltr1′)

have *0*: *case tp of* (*tr1*,*s1′*,*s1″*,*ltr1′*) ⇒
χχ s1 ltr1 tr1 s1′ s1″ ltr1′
using *χχ* **unfolding** *tp-def* **apply**− **apply**(*rule someI-ex*)
apply(*rule exI*[*of* - (*tr1*,*s1′*,*s1″*,*ltr1′*)]) **by** *auto*

obtain *tr1 s1′ s1″ ltr1′* **where**
tp: *tp* = (*tr1*,*s1′*,*s1″*,*ltr1′*) **by**(*cases tp*, *auto*)

have *χχ*: *χχ s1 ltr1 tr1 s1′ s1″ ltr1′*
using *0* **unfolding** *tp* **by** *auto*

obtain *w1′ w2′ trv1 sv1′′ trv2 sv2′′ statOO*
where *χ3′*: *χ3′* Δ *w1 w2 w1′ w2′ s1 tr1 s1′ s1′′ s2 statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
using *unwindCond-ex-χ3′*[*OF unw* Δ *r*, *of tr1 s1′ s1′′*]
using *χχ l′ s1* **unfolding** *χχ-def* **by** *auto*

define *tp′* **where**
tp′ = (*SOME k′. case k′ of* (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statOO*) ⇒
χ3′ Δ *w1 w2 w1′ w2′ s1 tr1 s1′ s1′′ s2 statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*)

have *1*: *case tp′ of* (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statOO*) ⇒
χ3′ Δ *w1 w2 w1′ w2′ s1 tr1 s1′ s1′′ s2 statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
using *χ3′* **unfolding** *tp′-def* **apply**− **apply**(*rule someI-ex*)
apply(*rule exI*[*of -* (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statOO*)]) **by** *auto*

obtain *w1′ w2′ trv1 sv1′′ trv2 sv2′′ statOO* **where**
tp′: *tp′* = (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statOO*) **by**(*cases tp′*, *auto*)

have *χ3′*: *χ3′* Δ *w1 w2 w1′ w2′ s1 tr1 s1′ s1′′ s2 statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
using *1* **unfolding** *tp′* **by** *auto*

show *?thesis*
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - tr1*]) **apply**(*rule exI*[*of - s1′*]) **apply**(*rule exI*[*of - s1′′*]) **apply**(*rule exI*[*of - ltr1′*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1′′*]) **apply**(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2′′*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal using *χχ* **.**
subgoal using *χ3′* **.**
subgoal using *l* **unfolding** *h-t.simps*
unfolding *tp-def*[*symmetric*] *tp* **apply** *simp*
unfolding *tp′-def*[*symmetric*] *tp′* **by** *simp* **.**
qed

lemma *lltrv1-lltrv2-not-lnever-L*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *l′*: *lnever isIntO ltr1* ¬ *isIntO s2*
and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *L* ¬ *lnever isSecO ltr1*

shows ∃ *w1′ w2′ tr1 s1′ s1′′ ltr1′ trv1 sv1′′ trv2 sv2′′ statOO*.
χχ s1 ltr1 tr1 s1′ s1′′ ltr1′ ∧
χ3′ Δ *w1 w2 w1′ w2′ s1 tr1 s1′ s1′′ s2 statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
∧
lltrv1 (*trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO*) =
lappend (*llist-of trv1*) (*lltrv1* (*R,w1′,w2′,s1′′,s1′′* $ *ltr1′,s2,ltr2,statA,sv1′′,sv2′′,statOO*))
∧
lltrv2 (*trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO*) =
lappend (*llist-of trv2*) (*lltrv2* (*R,w1′,w2′,s1′′,s1′′* $ *ltr1′,s2,ltr2,statA,sv1′′,sv2′′,statOO*))

proof −
show *?thesis*
using *h-t-not-lnever-L*[*OF assms*] **apply**(*elim exE*)
subgoal for *w1′ w2′ tr1 s1′ s1′′ ltr1′ trv1 sv1′′ trv2 sv2′′ statOO*
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - tr1*]) **apply**(*rule exI*[*of - s1′*]) **apply**(*rule exI*[*of - s1′′*])
apply(*rule exI*[*of - ltr1′*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1′′*]) **apply**(*rule exI*[*of - trv2*])
apply(*rule exI*[*of - sv2′′*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal by *simp*
subgoal by *simp*
subgoal unfolding *lltrv1-def* **apply**(*subst f-length-ge1*[*OF len*])
unfolding *h-def t-def* **by** *simp*
subgoal unfolding *lltrv2-def* **apply**(*subst f-length-ge1*[*OF len*])
unfolding *h-def t-def* **by** *simp* **. .**
qed

lemma *h-t-lnever-R*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l′*: ¬ *isIntO s1 lnever isIntO ltr2*
and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *R lnever isSecO ltr2*
shows ∃ *w1′ w2′ s2′ ltr2′ trv1 sv1′ trv2 sv2′ statOO*.
ltr2 = *s2* $ *ltr2′* ∧ *validTransO* (*s2,s2′*) ∧
Opt.lvalidFromS s2′ ltr2′ ∧ *Opt.lcompletedFrom s2′ ltr2′* ∧ *lnever isIntO ltr2′* ∧

ω4 Δ *w1 w2 w1′ w2′ s1 s2 s2′ statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO* ∧
h-t (*trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO*) =
((*trv1,sv1′,trv2,sv2′,statA,statOO*),
(*if trv2* = [] *then R else L*,
w1′,w2′,s1,ltr1,s2′,ltr2′,statA,sv1′,sv2′,statOO))
proof −

have *s2*: ¬ *isIntO s2* **using** *l′ ltr2*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil lhd-LCons llist.exhaust llist.pred-inject*(*2*))

obtain *ltr2′* **where** *ltr24*: *ltr2* = *s2* $ *ltr2′*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel ltr2*(*1*) *ltr2*(*2*))
hence *ltr2′*: *ltr2′* = *ltl ltr2* **by** *auto*
have *ltr2′ne*: *ltr2′* ≠ [[]] **using** *len*(*2*) **unfolding** *ltr24*
by (*metis One-nat-def llength-LCons llength-LNil one-eSuc one-enat-def order-less-irrefl*)
define *s2′* **where** *s2′*: *s2′* = *lhd* (*ltl ltr2*)
have *v4*: *validTransO* (*s2*,*s2′*) **and** *vv4*: *Opt.lvalidFromS s2′ ltr2′ Opt.lcompletedFrom s2′ ltr2′*
using *ltr2 ltr2′ne* **unfolding** *ltr24 s2′*
by (*metis Opt.lcompletedFrom-LCons Opt.lcompletedFrom-def Opt.lvalidFromS-Cons-iff ltr2′ ltr24*)+

have *is2′*: ¬ *isIntO s2′* **and** *lnever isIntO ltr2′*
using *l′*(*2*) **unfolding** *ltr24*
by (*metis llist.exhaust-sel llist.pred-inject*(*2*) *ltr2′ ltr2′ne s2′*)+

have *iss2*: ¬ *isSecO s2*
using *l*(*2*) *ltr24* **by** *auto*

obtain *w1′ w2′ trv1 sv1′ trv2 sv2′ statOO*
where *ω4*: *ω4* Δ *w1 w2 w1′ w2′ s1 s2* (*lhd* (*ltl ltr2*)) *statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
using *unwindCond-ex-ω4*[*OF unw* Δ *r l′*(*1*) *v4 s2 is2′ iss2*] *s2′* **by** *auto*

define *tp′* **where**
tp′ = (*SOME k′*. *case k′ of* (*w1′*,*w2′*,*trv1*,*sv1′*,*trv2*,*sv2′*,*statOO*) ⇒
ω4 Δ *w1 w2 w1′ w2′ s1 s2* (*lhd* (*ltl ltr2*)) *statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*)

have *1*: *case tp′ of* (*w1′*,*w2′*,*trv1*,*sv1′*,*trv2*,*sv2′*,*statOO*) ⇒
ω4 Δ *w1 w2 w1′ w2′ s1 s2 s2′ statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
using *ω4* **unfolding** *tp′-def s2′* **apply**− **apply**(*rule someI-ex*)
apply(*rule exI*[*of* - (*w1′*,*w2′*,*trv1*,*sv1′*,*trv2*,*sv2′*,*statOO*)]) **by** *auto*

obtain *w1′ w2′ trv1 sv1′ trv2 sv2′ statOO* **where**
tp′: *tp′* = (*w1′*,*w2′*,*trv1*,*sv1′*,*trv2*,*sv2′*,*statOO*) **by**(*cases tp′*, *auto*)

have *ω4*: *ω4* Δ *w1 w2 w1′ w2′ s1 s2 s2′ statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*

using *1* **unfolding** *tp′* **by** *auto*

show *?thesis*
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s2'*]) **apply**(*rule exI*[*of - ltr2'*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1'*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal using *len l* **unfolding** *h-t.simps* **apply** *simp*
unfolding *tp'-def*[*symmetric*] *tp' s2' ltr2'* **by** *simp* **.**
qed

lemma *lltrv1-lltrv2-lnever-R*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l'*: ¬ *isIntO s1 lnever isIntO ltr2*
and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *R lnever isSecO ltr2*
shows ∃ *w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO*.
ltr2 = *s2* $ *ltr2'* ∧ *validTransO* (*s2*,*s2'*) ∧
Opt.lvalidFromS s2' ltr2' ∧ *Opt.lcompletedFrom s2' ltr2'* ∧ *lnever isIntO ltr2'* ∧

ω4 Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO* ∧
lltrv1 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv1*) (*lltrv1* (*if trv2* = [] *then R else L*,
w1',*w2'*,*s1*,*ltr1*,*s2'*,*ltr2'*,*statA*,*sv1'*,*sv2'*,*statOO*)) ∧
lltrv2 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv2*) (*lltrv2* (*if trv2* = [] *then R else L*,
w1',*w2'*,*s1*,*ltr1*,*s2'*,*ltr2'*,*statA*,*sv1'*,*sv2'*,*statOO*))
proof−
show *?thesis*
using *h-t-lnever-R*[*OF assms*] **apply**(*elim exE*)
subgoal for *w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO*
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s2'*]) **apply**(*rule exI*[*of - ltr2'*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1'*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*

subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal unfolding *lltrv1-def* **apply**(*subst f-length-ge1*[*OF len*])
unfolding *h-def t-def* **by** *auto*
subgoal unfolding *lltrv2-def* **apply**(*subst f-length-ge1*[*OF len*])
unfolding *h-def t-def* **by** *auto* **. .**
qed

lemma *h-t-not-lnever-R*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l'*: ¬ *isIntO s1 lnever isIntO ltr2*
and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *R* ¬ *lnever isSecO ltr2*
shows ∃ *w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO*.
χχ s2 ltr2 tr2 s2' s2'' ltr2' ∧
χ4' Δ *w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
∧
h-t (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
((*trv1*,*sv1''*,*trv2*,*sv2''*,*statA*,*statOO*),
(*L*,*w1'*,*w2'*,*s1*,*ltr1*,*s2''*,*s2''* $ *ltr2'*,*statA*,*sv1''*,*sv2''*,*statOO*))
proof−
have *s2*: ¬ *isIntO s2* **using** *l' ltr2*
by (*metis Simple-Transition-System.lvalidFromS-def l*(*2*) *lhd-LCons llist.pred-inject*(*1*)

llist.pred-inject(*2*) *neq-LNil-conv*)

obtain *tr2 s2' s2'' ltr2'*
where *χχ*: *χχ s2 ltr2 tr2 s2' s2'' ltr2'*
using *isSecO-χχ*[*OF ltr2 l'*(*2*) *l*(*2*)] **by** *auto*

define *tp* **where**
tp = (*SOME k. case k of* (*tr2*,*s2'*,*s2''*,*ltr2'*) ⇒
χχ s2 ltr2 tr2 s2' s2'' ltr2')

have *0*: *case tp of* (*tr2*,*s2'*,*s2''*,*ltr2'*) ⇒
χχ s2 ltr2 tr2 s2' s2'' ltr2'
using *χχ* **unfolding** *tp-def* **apply**− **apply**(*rule someI-ex*)
apply(*rule exI*[*of* - (*tr2*,*s2'*,*s2''*,*ltr2'*)]) **by** *auto*

obtain *tr2 s2' s2'' ltr2'* **where**
tp: *tp* = (*tr2*,*s2'*,*s2''*,*ltr2'*) **by**(*cases tp*, *auto*)

have *χχ*: *χχ s2 ltr2 tr2 s2' s2'' ltr2'*

using *0* **unfolding** *tp* **by** *auto*

obtain *w1′ w2′ trv1 sv1′′ trv2 sv2′′ statOO*
where *χ4′*: *χ4′* Δ *w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2′′ statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
using *unwindCond-ex-χ4′*[*OF unw* Δ *r*, *of tr2 s2′ s2′′*]
using *χχ l′ s2* **unfolding** *χχ-def* **by** *auto*

define *tp′* **where**
tp′ = (*SOME k′*. *case k′ of* (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statOO*) ⇒
χ4′ Δ *w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2′′ statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*)

have *1*: *case tp′ of* (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statOO*) ⇒
χ4′ Δ *w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2′′ statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
using *χ4′* **unfolding** *tp′-def* **apply**− **apply**(*rule someI-ex*)
apply(*rule exI*[*of* - (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statOO*)]) **by** *auto*

obtain *w1′ w2′ trv1 sv1′′ trv2 sv2′′ statOO* **where**
tp′: *tp′* = (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statOO*) **by**(*cases tp′*, *auto*)

have *χ4′*: *χ4′* Δ *w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2′′ statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
using *1* **unfolding** *tp′* **by** *auto*

show *?thesis*
apply(*rule exI*[*of* - *w1′*]) **apply**(*rule exI*[*of* - *w2′*])
apply(*rule exI*[*of* - *tr2*]) **apply**(*rule exI*[*of* - *s2′*]) **apply**(*rule exI*[*of* - *s2′′*])
apply(*rule exI*[*of* - *ltr2′*])
apply(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *sv1′′*]) **apply**(*rule exI*[*of* - *trv2*])
apply(*rule exI*[*of* - *sv2′′*])
apply(*rule exI*[*of* - *statOO*])
apply(*intro conjI*)
subgoal using *χχ* **.**
subgoal using *χ4′* **.**
subgoal using *l* **unfolding** *h-t.simps*
unfolding *tp-def*[*symmetric*] *tp* **apply** *simp*
unfolding *tp′-def*[*symmetric*] *tp′* **by** *auto* **.**
qed

lemma *lltrv1-lltrv2-not-lnever-R*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l′*: ¬ *isIntO s1 lnever isIntO ltr2*

and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *R* ¬ *lnever isSecO ltr2*
shows ∃ *w1′ w2′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statOO*.
χχ s2 ltr2 tr2 s2′ s2′′ ltr2′ ∧
χ4′ Δ *w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2′′ statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
∧
lltrv1 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv1*) (*lltrv1* (*L*,*w1′*,*w2′*,*s1*,*ltr1*,*s2′′*,*s2′′* $ *ltr2′*,*statA*,*sv1′′*,*sv2′′*,*statOO*))
∧
lltrv2 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv2*) (*lltrv2* (*L*,*w1′*,*w2′*,*s1*,*ltr1*,*s2′′*,*s2′′* $ *ltr2′*,*statA*,*sv1′′*,*sv2′′*,*statOO*))
proof−
show *?thesis*
using *h-t-not-lnever-R*[*OF assms*] **apply**(*elim exE*)
subgoal for *w1′ w2′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statOO*
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - tr2*]) **apply**(*rule exI*[*of - s2′*]) **apply**(*rule exI*[*of - s2′′*])
apply(*rule exI*[*of - ltr2′*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1′′*]) **apply**(*rule exI*[*of - trv2*])
apply(*rule exI*[*of - sv2′′*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal by *simp*
subgoal by *simp*
subgoal unfolding *lltrv1-def* **apply**(*subst f-length-ge1*[*OF len*])
unfolding *h-def t-def* **by** *simp*
subgoal unfolding *lltrv2-def* **apply**(*subst f-length-ge1*[*OF len*])
unfolding *h-def t-def* **by** *simp* **. .**
qed

lemma *f-not-LNil*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] ⟹
f (*w1*,*w2*,*trn*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*) ≠ [[]]
apply(*cases llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)
subgoal apply(*subst f-length-1*) **by** *auto*
subgoal apply(*subst f-length-ge1*) **by** *auto* **.**

lemma *lvalidFromS-lltrv1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lvalidFromS sv1* (*lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
proof−
{**fix** *n sv1 ltrv1*
assume ∃ *trn w1 w2 s1 ltr1 s2 ltr2 statA sv2 statO*.

n = *w1* ∧
ltrv1 = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*
hence *Van.llvalidFromS n sv1 ltrv1*
proof(*coinduct rule*: *Van.llvalidFromS.coinduct*[*of* λ*n sv1 ltrv1*.
∃ *trn w1 w2 s1 ltr1 s2 ltr2 statA sv2 statO*.
n = *w1* ∧
ltrv1 = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*])
case (*llvalidFromS n sv1 ltrv1*)
then obtain *trn w1 w2 s1 ltr1 s2 ltr2 statA sv2 statO*
where *n*: *n* = *w1* **and**
ltrv1: *ltrv1* = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
by *auto*
have *isi3*: ¬ *isIntO s1* **using** *ltr1*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))
have *isi4*: ¬ *isIntO s2* **using** *ltr2*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))

show *?case* **proof**(*cases ltr1* = [[]] ∧ *ltr2* = [[]])
case *True* **note** *ltr14* = *True*
hence *ltrv1*: *ltrv1* = [[]] **unfolding** *ltrv1* **by** *simp*
show *?thesis* **unfolding** *ltrv1* **apply**(*rule Van.llvalidFromS-selectLNil*) **by** *auto*
next
case *False* **hence** *ltr14*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] **by** *auto*
show *?thesis* **proof**(*cases llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)
case *True* **note** *ltr14* = *ltr14 True*
hence *ltrv1*: *ltrv1* = [[*sv1*]] **unfolding** *ltrv1* **by** *simp*
show *?thesis* **unfolding** *ltrv1* **apply**(*rule Van.llvalidFromS-selectSingl*) **by** *auto*
next
case *False* **hence** *current*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
by *auto*
show *?thesis* **proof**(*cases trn*)

case *L* **note** *trn* = *L* **note** *current* = *current L*
show *?thesis*
proof(*cases lnever isSecO ltr1*)
case *True* **note** *current* = *current True*
obtain *trn′ w1′ w2′ s1′ ltr1′ trv1 sv1′ trv2 sv2′ statOO* **where**
ωω: *ltr1* = *s1* $ *ltr1′ validTransO* (*s1*, *s1′*) *Opt.lvalidFromS s1′ ltr1′*
lcompletedFromO s1′ ltr1′ lnever isIntO ltr1′ **and**
ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
and *trn′*: *trn′* = (*if trv1* = [] *then L else R*)
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*, *sv1′*, *sv2′*, *statOO*))
using *lltrv1-lltrv2-lnever-L*[*OF unw* Δ *r ltr1 isi4 current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1′* **where** *ltrv1′*: *ltrv1′* ≡ *lltrv1* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*, *sv1′*, *sv2′*, *statOO*)
have *ltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1′*
unfolding *ltrv1 ltrv1′* **..**

show *?thesis*
proof(*cases trv1* = [])
case *True* **note** *trv1* = *True*
have *sv1′*: *sv1′* = *sv1*
using *ω3* **unfolding** *ω3-def* **by** (*simp add*: *trv1*)
show *?thesis*
apply(*rule Van.llvalidFromS-selectDelay*)
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - n*])
apply(*rule exI*[*of - sv1*]) **apply**(*rule exI*[*of - ltrv1′*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv1 trv1* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def trv1 n* **by** *simp*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn′*])
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - s1′*]) **apply**(*rule exI*[*of - ltr1′*])
apply(*rule exI*[*of - s2*]) **apply**(*rule exI*[*of - ltr2*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv2′*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal ..
subgoal unfolding *ltrv1′ trn′ trv1 sv1′* **using** *trn* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def sv1′* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact* **subgoal by** *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Nil-is-append-conv Van.reach-validFromS-reach last-snoc*

not-Cons-self2 r(*4*))
subgoal using *ωω* **by** *simp* **subgoal using** *ωω* **by** *simp* **subgoal using** *ωω* **by** *simp*
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **. .**
next
case *False* **note** *trv1* = *False*
show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of - sv1*]) **apply**(*rule exI*[*of - trv1*])
apply(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - w1'*])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - n*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal using *ltrv1* **.**
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Nil-is-append-conv Van.validFromS-def Van.validS-append1 hd-append2*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Van.validFromS-def Van.validS-validTrans list.sel*(*1*) *not-Cons-self2 snoc-eq-iff-butlast trv1*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn'*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - s1'*]) **apply**(*rule exI*[*of - ltr1'*])
apply(*rule exI*[*of - s2*]) **apply**(*rule exI*[*of - ltr2*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv2'*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal ..
subgoal using *trv1* **unfolding** *ltrv1' trn'* **by** *auto*
subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal using *ωω*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal using *ωω* **using** *ltr2*(*1*) **by** *fastforce*
subgoal by *fact*
subgoal by *fact* **. .**
qed

```
next
  case False note current = current False
  obtain w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO where
  χχ: χχ s1 ltr1 tr1 s1' s1'' ltr1' and
   χ3': χ3' Δ w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2
trv2 sv2'' statOO
  and ltrv1: ltrv1 =
lappend (llist-of trv1) (lltrv1 (R,w1',w2',s1'',s1'' $ ltr1',s2,ltr2,statA,sv1'',sv2'',statOO))

  using lltrv1-lltrv2-not-lnever-L[OF unw Δ r ltr1 isi4 current]
  unfolding ltrv1 by blast
    define ltrv1' where ltrv1': ltrv1' ≡ lltrv1 (R,w1',w2',s1'',s1'' $
ltr1',s2,ltr2,statA,sv1'',sv2'',statOO)
  have ltrv1: ltrv1 = lappend (llist-of trv1) ltrv1'
  unfolding ltrv1 ltrv1' ..

  show ?thesis apply(rule Van.llvalidFromS-selectlappend)
  apply(rule exI[of - sv1]) apply(rule exI[of - trv1])
  apply(rule exI[of - sv1'']) apply(rule exI[of - w1'])
  apply(rule exI[of - ltrv1']) apply(rule exI[of - w1])
  apply(intro conjI)
    subgoal unfolding n .. subgoal ..
    subgoal using ltrv1 .
    subgoal using χ3' unfolding χ3'-def
     by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
    subgoal using χ3' unfolding χ3'-def by simp
    subgoal using χ3' unfolding χ3'-def
  by (metis Van.validFromS Van.validS-validTrans Simple-Transition-System.validFromS-def

     append-is-Nil-conv not-Cons-self2)
    subgoal apply(rule disjI1)
    apply(rule exI[of - R])
    apply(rule exI[of - w1']) apply(rule exI[of - w2'])
    apply(rule exI[of - s1'']) apply(rule exI[of - s1'' $ ltr1'])
    apply(rule exI[of - s2]) apply(rule exI[of - ltr2])
  apply(rule exI[of - statA]) apply(rule exI[of - sv2'']) apply(rule exI[of
- statOO])
    apply(intro conjI)
      subgoal ..
      subgoal unfolding ltrv1' ..
      subgoal using χ3' unfolding χ3'-def by simp
      subgoal using χ3' unfolding χ3'-def
     by (metis Simple-Transition-System.reach-validFromS-reach χχ χχ-def

      append-is-Nil-conv last-snoc not-Cons-self2 r(1))
      subgoal by fact
      subgoal using χ3' unfolding χ3'-def
        by (metis Simple-Transition-System.reach-validFromS-reach r(3)
```

snoc-eq-iff-butlast)
subgoal using *χ3'* **unfolding** *χ3'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*)
snoc-eq-iff-butlast)
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal using *χχ* **unfolding** *χχ-def* **using** *ltr2*(*1*) **by** *fastforce*
subgoal by *fact*
subgoal by *fact* **. .**
qed
next
case *R* **note** *trn* = *R* **note** *current* = *current R*
show *?thesis*
proof(*cases lnever isSecO ltr2*)
case *True* **note** *current* = *current True*
obtain *trn' w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO* **where**
ωω: *ltr2* = *s2* $ *ltr2' validTransO* (*s2*, *s2'*) *Opt.lvalidFromS s2' ltr2'*
lcompletedFromO s2' ltr2' lnever isIntO ltr2' **and**
ω4: *ω4* Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2'*
statOO
and *trn'*: *trn'* = (*if trv2* = [] *then R else L*)
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*,
sv1', *sv2'*, *statOO*))
using *lltrv1-lltrv2-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1'* **where** *ltrv1'*: *ltrv1'* ≡ *lltrv1* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*,
ltr2', *statA*, *sv1'*, *sv2'*, *statOO*)
have *ltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1'*
unfolding *ltrv1 ltrv1'* **..**

show *?thesis*
proof(*cases trv1* = [])
case *True* **note** *trv1* = *True*
have *sv1'*: *sv1'* = *sv1*
using *ω4* **unfolding** *ω4-def* **by** (*simp add*: *trv1*)
show *?thesis*
apply(*rule Van.llvalidFromS-selectDelay*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - n*])
apply(*rule exI*[*of - sv1*]) **apply**(*rule exI*[*of - ltrv1'*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv1 trv1* **by** *simp*
subgoal using *ω4* **unfolding** *ω4-def trv1 n* **by** *simp*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn'*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])

apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - s2'*]) **apply**(*rule exI*[*of - ltr2'*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv2'*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal ..
subgoal unfolding *ltrv1' trn' trv1 sv1'* **using** *trn* **by** *simp*
subgoal using *ω4* **unfolding** *ω4-def sv1'* **by** *simp*
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Nil-is-append-conv Van.reach-validFromS-reach last-snoc not-Cons-self2 r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *ωω* **by** *simp* **subgoal using** *ωω* **by** *simp* **subgoal using** *ωω* **by** *simp* **. .**
next
case *False* **note** *trv1* = *False*
show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of - sv1*]) **apply**(*rule exI*[*of - trv1*])
apply(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - w1'*])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - n*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal using *ltrv1* **.**
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Nil-is-append-conv Van.validFromS-def Van.validS-append1 hd-append2*)
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.validFromS-def Van.validS-validTrans list.sel*(*1*) *not-Cons-self2 snoc-eq-iff-butlast trv1*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn'*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - s2'*]) **apply**(*rule exI*[*of - ltr2'*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv2'*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal ..
subgoal using *trv1* **unfolding** *ltrv1' trn'* **by** *auto*
subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)

subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal using *ωω*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast* **. .**
qed
next
case *False* **note** *current* = *current False*
obtain *w1′ w2′ tr2 s2′ s2″ ltr2′ trv1 sv1″ trv2 sv2″ statOO* **where**
χχ: *χχ s2 ltr2 tr2 s2′ s2″ ltr2′* **and**
χ4′: *χ4′ Δ w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2″ statA sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2″*, *s2″* $ *ltr2′*, *statA*, *sv1″*, *sv2″*, *statOO*))
using *lltrv1-lltrv2-not-lnever-R*[*OF unw Δ r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1′* **where** *ltrv1′*: *ltrv1′* ≡ *lltrv1* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2″*, *s2″* $ *ltr2′*, *statA*, *sv1″*, *sv2″*, *statOO*)
have *ltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1′*
unfolding *ltrv1 ltrv1′* **..**

show *?thesis*
proof(*cases trv1* = [])
case *True* **note** *trv1* = *True*
hence *sv1″*: *sv1″* = *sv1*
by (*metis χ4′-def Simple-Transition-System.validFromS-Cons-iff χ4′ append.simps*(*1*))
have *w1′* < *w1* **using** *trv1 χ4′* **unfolding** *χ4′-def* **by** *auto*
show *?thesis*
apply(*rule Van.llvalidFromS-selectDelay*)
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - n*])
apply(*rule exI*[*of - sv1*]) **apply**(*rule exI*[*of - ltrv1*])
apply(*intro conjI*)
subgoal ..
subgoal .. subgoal .. subgoal unfolding *n* **by** *fact*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - L*])
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - s2″*]) **apply**(*rule exI*[*of - s2″* $ *ltr2′*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv2″*]) **apply**(*rule*

exI[*of* - *statOO*])
apply(*intro conjI*)
subgoal ..
subgoal unfolding *ltrv1 ltrv1' trv1 sv1''* **by** *simp*
subgoal using *χ4'* **unfolding** *χ4'-def sv1''* **by** *simp*
subgoal by *fact*
subgoal using *χχ* **unfolding** *χχ-def*
by (*metis Opt.reach-validFromS-reach Nil-is-append-conv last-snoc not-Cons-self2 r*(*2*))
subgoal by *fact*
subgoal using *χ4' r*(*4*) **unfolding** *χ4'-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr2*(*3*) **by** *blast* **. .**
next
case *False* **note** *trv1* = *False*
show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of* - *sv1*]) **apply**(*rule exI*[*of* - *trv1*])
apply(*rule exI*[*of* - *sv1''*])
apply(*rule exI*[*of* - *w1'*])
apply(*rule exI*[*of* - *ltrv1'*])
apply(*rule exI*[*of* - *n*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal using *ltrv1* **.**
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Nil-is-append-conv Van.validFromS-def Van.validS-append1 hd-append2*)
subgoal using *trv1* **.**
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Simple-Transition-System.validFromS-def Van.validS-validTrans list.sel*(*1*)
not-Cons-self2 snoc-eq-iff-butlast trv1)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of* - *L*])
apply(*rule exI*[*of* - *w1'*]) **apply**(*rule exI*[*of* - *w2'*])
apply(*rule exI*[*of* - *s1*]) **apply**(*rule exI*[*of* - *ltr1*])
apply(*rule exI*[*of* - *s2''*]) **apply**(*rule exI*[*of* - *s2''* $ *ltr2'*])
apply(*rule exI*[*of* - *statA*]) **apply**(*rule exI*[*of* - *sv2''*]) **apply**(*rule exI*[*of* - *statOO*])
apply(*intro conjI*)
subgoal .. subgoal unfolding *ltrv1'* **..**
subgoal using *χ4'* **unfolding** *χ4'-def* **by** *simp*
subgoal by *fact*
subgoal using *χ4'* **unfolding** *χ4'-def*

by (*metis Simple-Transition-System.reach-validFromS-reach* $\chi\chi$ $\chi\chi$*-def*
append-is-Nil-conv last-snoc not-Cons-self2 r(*2*))
subgoal using $\chi 4'$ **unfolding** $\chi 4'$*-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using $\chi 4'$ **unfolding** $\chi 4'$*-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def*
using *llist-all-lappend-llist-of ltr2*(*3*) **by** *blast* **. .**
qed
qed
qed
qed
qed
qed
}
thus *?thesis* **apply**−**apply**(*rule Van.llvalidFromS-imp-lvalidFromS*)
using *assms* **by** *blast*
qed

lemma *lvalidFromS-lltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lvalidFromS sv2* (*lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
proof−
{**fix** *n sv2 ltrv2*
assume ∃ *trn w1 w2 s1 ltr1 s2 ltr2 statA sv1 statO*.
n = *w2* ∧
ltrv2 = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*
hence *Van.llvalidFromS n sv2 ltrv2*
proof(*coinduct rule*: *Van.llvalidFromS.coinduct*[*of* λ*n sv2 ltrv2*.
∃ *trn w1 w2 s1 ltr1 s2 ltr2 statA sv1 statO*.
n = *w2* ∧

ltrv2 = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*])
case (*llvalidFromS n sv2 ltrv2*)
then obtain *trn w1 w2 s1 ltr1 s2 ltr2 statA sv1 statO*
where *n*: *n* = *w2* **and**
ltrv2: *ltrv2* = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
by *auto*
have *isi3*: ¬ *isIntO s1* **using** *ltr1*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))
have *isi4*: ¬ *isIntO s2* **using** *ltr2*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))

show *?case* **proof**(*cases ltr1* = [[]] ∧ *ltr2* = [[]])
case *True* **note** *ltr14* = *True*
hence *ltrv2*: *ltrv2* = [[]] **unfolding** *ltrv2* **by** *simp*
show *?thesis* **unfolding** *ltrv2* **apply**(*rule Van.llvalidFromS-selectLNil*) **by** *auto*
next
case *False* **hence** *ltr14*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] **by** *auto*
show *?thesis* **proof**(*cases llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)
case *True* **note** *ltr14* = *ltr14 True*
hence *ltrv2*: *ltrv2* = [[*sv2*]] **unfolding** *ltrv2* **by** *simp*
show *?thesis* **unfolding** *ltrv2* **apply**(*rule Van.llvalidFromS-selectSingl*) **by** *auto*
next
case *False* **hence** *current*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
by *auto*
show *?thesis* **proof**(*cases trn*)
case *L* **note** *trn* = *L* **note** *current* = *current L*
show *?thesis*
proof(*cases lnever isSecO ltr1*)
case *True* **note** *current* = *current True*
obtain *trn′ w1′ w2′ s1′ ltr1′ trv1 sv1′ trv2 sv2′ statOO* **where**
ωω: *ltr1* = *s1* $ *ltr1′ validTransO* (*s1*, *s1′*) *Opt.lvalidFromS s1′ ltr1′*
lcompletedFromO s1′ ltr1′ lnever isIntO ltr1′ **and**
ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
and *trn′*: *trn′* = (*if trv1* = [] *then L else R*)
and *ltrv2*: *ltrv2* =

lappend (*llist-of trv2*) (*lltrv2*(*trn'*, *w1'*, *w2'*, *s1'*, *ltr1'*, *s2*, *ltr2*, *statA*, *sv1'*, *sv2'*, *statOO*))
using *lltrv1-lltrv2-lnever-L*[*OF unw* Δ *r ltr1 isi4 current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2'* **where** *ltrv2'*: *ltrv2'* ≡ *lltrv2* (*trn'*, *w1'*, *w2'*, *s1'*, *ltr1'*, *s2*, *ltr2*, *statA*, *sv1'*, *sv2'*, *statOO*)
have *ltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2'*
unfolding *ltrv2 ltrv2'* **..**

show *?thesis*
proof(*cases trv2* = [])
case *True* **note** *trv2* = *True*
have *sv2'*: *sv2'* = *sv2*
using *ω3* **unfolding** *ω3-def* **by** (*simp add*: *trv2*)
show *?thesis*
apply(*rule Van.llvalidFromS-selectDelay*)
apply(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - n*])
apply(*rule exI*[*of - sv2*]) **apply**(*rule exI*[*of - ltrv2'*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv2 trv2* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def trv2 n* **by** *simp*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn'*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1'*]) **apply**(*rule exI*[*of - ltr1'*])
apply(*rule exI*[*of - s2*]) **apply**(*rule exI*[*of - ltr2*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal ..
subgoal unfolding *ltrv2' trn' trv2 sv2'* **using** *trn* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def sv2'* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Nil-is-append-conv Van.reach-validFromS-reach last-snoc not-Cons-self2 r*(*3*))
subgoal by *fact*
subgoal using *ωω* **by** *simp* **subgoal using** *ωω* **by** *simp* **subgoal using** *ωω* **by** *simp*
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **. .**
next
case *False* **note** *trv2* = *False*
show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of - sv2*]) **apply**(*rule exI*[*of - trv2*])
apply(*rule exI*[*of - sv2'*]) **apply**(*rule exI*[*of - w2'*])

apply(*rule exI*[*of* - *ltrv2'*]) **apply**(*rule exI*[*of* - *n*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal using *ltrv2* **.**
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Nil-is-append-conv Van.validFromS-def Van.validS-append1 hd-append2*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Van.validFromS-def Van.validS-validTrans append-is-Nil-conv list.sel*(*1*) *not-Cons-self2 trv2*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of* - *trn'*])
apply(*rule exI*[*of* - *w1'*]) **apply**(*rule exI*[*of* - *w2'*]) **apply**(*rule exI*[*of* - *s1'*]) **apply**(*rule exI*[*of* - *ltr1'*])
apply(*rule exI*[*of* - *s2*]) **apply**(*rule exI*[*of* - *ltr2*])
apply(*rule exI*[*of* - *statA*]) **apply**(*rule exI*[*of* - *sv1'*]) **apply**(*rule exI*[*of* - *statOO*])
apply(*intro conjI*)
subgoal ..
subgoal using *trv2* **unfolding** *ltrv2' trn'* **by** *auto*
subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal using *ωω*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal using *ωω* **using** *ltr2*(*1*) **by** *fastforce*
subgoal by *fact*
subgoal by *fact* **. .**
qed
next
case *False* **note** *current* = *current False*
obtain *w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO* **where**
χχ: *χχ s1 ltr1 tr1 s1' s1'' ltr1'* **and**
χ3': *χ3' Δ w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
and *ltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*R*,*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2*,*ltr2*,*statA*,*sv1''*,*sv2''*,*statOO*))

using *lltrv1-lltrv2-not-lnever-L*[*OF unw Δ r ltr1 isi4 current*]

unfolding *ltrv2* **by** *blast*
define *ltrv2′* **where** *ltrv2′*: *ltrv2′* ≡ *lltrv2* (*R*,*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2*,*ltr2*,*statA*,*sv1″*,*sv2″*,*statOO*)
have *ltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2′*
unfolding *ltrv2 ltrv2′* **..**

show *?thesis*
proof(*cases trv2* = [])
case *True* **note** *trv2* = *True*
hence *sv2″*: *sv2″* = *sv2*
by (*metis* $\chi 3'$*-def Simple-Transition-System.validFromS-Cons-iff* $\chi 3'$ *append.simps*(*1*))
have *w2′* < *w2* **using** *trv2* $\chi 3'$ **unfolding** $\chi 3'$*-def* **by** *auto*
show *?thesis*
apply(*rule Van.llvalidFromS-selectDelay*)
apply(*rule exI*[*of - w2′*]) **apply**(*rule exI*[*of - n*])
apply(*rule exI*[*of - sv2*]) **apply**(*rule exI*[*of - ltrv2*])
apply(*intro conjI*)
subgoal ..
subgoal .. subgoal .. subgoal unfolding *n* **by** *fact*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - R*])
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - s1″*]) **apply**(*rule exI*[*of - s1″* $ *ltr1′*])
apply(*rule exI*[*of - s2*]) **apply**(*rule exI*[*of - ltr2*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv1″*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal ..
subgoal unfolding *ltrv2 ltrv2′ trv2 sv2″* **by** *simp*
subgoal using $\chi 3'$ **unfolding** $\chi 3'$*-def sv2″* **by** *simp*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def*
by (*metis Opt.reach-validFromS-reach Nil-is-append-conv last-snoc not-Cons-self2 r*(*1*))
subgoal by *fact*
subgoal using $\chi 3'$ *r*(*3*) **unfolding** $\chi 3'$*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal by *fact*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **. .**
next
case *False* **note** *trv2* = *False*
show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of - sv2*]) **apply**(*rule exI*[*of - trv2*])
apply(*rule exI*[*of - sv2″*]) **apply**(*rule exI*[*of - w2′*])

```
          apply(rule exI[of - ltrv2']) apply(rule exI[of - w2])
          apply(intro conjI)
            subgoal unfolding n .. subgoal ..
            subgoal using ltrv2 .
            subgoal using χ3' unfolding χ3'-def
          by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
            subgoal by fact
            subgoal using χ3' unfolding χ3'-def
        by (metis Simple-Transition-System.validFromS-def Van.validS-validTrans
append-is-Nil-conv list.sel(1) not-Cons-self2 trv2)
            subgoal apply(rule disjI1)
            apply(rule exI[of - R])
            apply(rule exI[of - w1']) apply(rule exI[of - w2'])
            apply(rule exI[of - s1'']) apply(rule exI[of - s1'' $ ltr1'])
            apply(rule exI[of - s2]) apply(rule exI[of - ltr2])
              apply(rule exI[of - statA]) apply(rule exI[of - sv1'']) apply(rule
exI[of - statOO])
            apply(intro conjI)
              subgoal ..
              subgoal unfolding ltrv2' ..
              subgoal using χ3' unfolding χ3'-def by simp
              subgoal using χ3' unfolding χ3'-def
                by (metis Simple-Transition-System.reach-validFromS-reach χχ
χχ-def
              append-is-Nil-conv last-snoc not-Cons-self2 r(1))
              subgoal by fact
              subgoal using χ3' unfolding χ3'-def
               by (metis Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
              subgoal using χ3' unfolding χ3'-def
               by (metis Simple-Transition-System.reach-validFromS-reach r(4)
snoc-eq-iff-butlast)
              subgoal using χχ unfolding χχ-def by auto
              subgoal using χχ unfolding χχ-def by auto
              subgoal using χχ unfolding χχ-def
              using llist-all-lappend-llist-of ltr1(3) by blast
              subgoal using χχ unfolding χχ-def using ltr2(1) by fastforce
              subgoal by fact
              subgoal by fact . .
      qed
    qed
  next
    case R note trn = R note current = current R
    show ?thesis
    proof(cases lnever isSecO ltr2)
      case True note current = current True
      obtain trn' w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO where
      ωω: ltr2 = s2 $ ltr2' validTransO (s2, s2') Opt.lvalidFromS s2' ltr2'
```

lcompletedFromO s2' ltr2' lnever isIntO ltr2' **and**
ω4: *ω4* Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*
and *trn'*: *trn'* = (*if trv2* = [] *then R else L*)
and *ltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*, *sv1'*, *sv2'*, *statOO*))
using *lltrv1-lltrv2-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2'* **where** *ltrv2'*: *ltrv2'* ≡ *lltrv2* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*, *sv1'*, *sv2'*, *statOO*)
have *ltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2'*
unfolding *ltrv2 ltrv2'* **..**

show *?thesis*
proof(*cases trv2* = [])
case *True* **note** *trv2* = *True*
have *sv2'*: *sv2'* = *sv2*
using *ω4* **unfolding** *ω4-def* **by** (*simp add*: *trv2*)
show *?thesis*
apply(*rule Van.llvalidFromS-selectDelay*)
apply(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - n*])
apply(*rule exI*[*of - sv2*]) **apply**(*rule exI*[*of - ltrv2'*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv2 trv2* **by** *simp*
subgoal using *ω4* **unfolding** *ω4-def trv2 n* **by** *simp*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn'*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - s2'*]) **apply**(*rule exI*[*of - ltr2'*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal ..
subgoal unfolding *ltrv2' trn' trv2 sv2'* **using** *trn* **by** *simp*
subgoal using *ω4* **unfolding** *ω4-def sv2'* **by** *simp*
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Nil-is-append-conv Van.reach-validFromS-reach last-snoc not-Cons-self2 r*(*3*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *ωω* **by** *simp* **subgoal using** *ωω* **by** *simp* **subgoal using** *ωω* **by** *simp* **. .**
next

case *False* **note** *trv2* = *False*
show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of - sv2*]) **apply**(*rule exI*[*of - trv2*])
apply(*rule exI*[*of - sv2'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - ltrv2'*]) **apply**(*rule exI*[*of - n*])
apply(*intro conjI*)
subgoal .. **subgoal** ..
subgoal using *ltrv2* .
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Nil-is-append-conv Van.validFromS-def Van.validS-append1 hd-append2*)
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.validFromS-def Van.validS-validTrans append-is-Nil-conv list.sel*(*1*) *not-Cons-self2 trv2*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn'*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - s2'*]) **apply**(*rule exI*[*of - ltr2'*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal ..
subgoal using *trv2* **unfolding** *ltrv2' trn'* **by** *auto*
subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal using *ωω*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast* . .
qed
next
case *False* **note** *current* = *current False*
obtain *w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO* **where**
χχ: *χχ s2 ltr2 tr2 s2' s2'' ltr2'* **and**
χ4': *χ4' Δ w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
and *ltrv2*: *ltrv2* =

lappend (*llist-of trv2*) (*lltrv2* (*L*, *w1'*, *w2'*, *s1*, *ltr1*, *s2''*, *s2''* $ *ltr2'*, *statA*, *sv1''*, *sv2''*, *statOO*))
using *lltrv1-lltrv2-not-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2'* **where** *ltrv2'*: *ltrv2'* ≡ *lltrv2* (*L*, *w1'*, *w2'*, *s1*, *ltr1*, *s2''*, *s2''* $ *ltr2'*, *statA*, *sv1''*, *sv2''*, *statOO*)
have *ltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2'*
unfolding *ltrv2 ltrv2'* **..**
have *trv2*: *trv2* ≠ [] **using** *χ4'* **unfolding** *χ4'-def* **by** *auto*

show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of - sv2*]) **apply**(*rule exI*[*of - trv2*])
apply(*rule exI*[*of - sv2''*])
apply(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - ltrv2'*])
apply(*rule exI*[*of - n*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal using *ltrv2* **.**
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Nil-is-append-conv Van.validFromS-def Van.validS-append1 hd-append2*)
subgoal using *trv2* **.**
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Simple-Transition-System.validFromS-def Van.validS-validTrans append-is-Nil-conv list.sel*(*1*) *not-Cons-self2*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - L*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - s2''*]) **apply**(*rule exI*[*of - s2''* $ *ltr2'*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal unfolding *ltrv2'* **..**
subgoal using *χ4'* **unfolding** *χ4'-def* **by** *simp*
subgoal by *fact*
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach χχ χχ-def*

append-is-Nil-conv last-snoc not-Cons-self2 r(*2*))
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)

subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr2*(*3*) **by** *blast* **. .**
qed
qed
qed
qed
qed
}
thus *?thesis* **apply**−**apply**(*rule Van.llvalidFromS-imp-lvalidFromS*)
using *assms* **by** *blast*
qed

lemma *lcompletedFrom-lltrv1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lcompletedFrom sv1* (*lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
proof−
{fix *ltrv1* **assume** *ltrv1*: *ltrv1* = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *lfin*: *lfinite ltrv1*
hence *list-of ltrv1* ≠ [] ∧ *finalV* (*last* (*list-of ltrv1*))
using *assms*(*2*−) **proof**(*induct length* (*list-of ltrv1*) *w1*
arbitrary: *trn w2 ltrv1 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*
rule: *less2-induct′*)
case (*less w1 ltrv1 trn w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*)
hence *ltrv1*: *ltrv1* = *lltrv1* (*trn*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
and *lfin*: *lfinite ltrv1*
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 lcompletedFromO s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 lcompletedFromO s2 ltr2 lnever isIntO ltr2*
by *auto*
have *isi3*: ¬ *isIntO s1* **using** *ltr1*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))
have *isi4*: ¬ *isIntO s2* **using** *ltr2*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))

show *?case* **proof**(*cases ltr1* = [[]] ∧ *ltr2* = [[]])
case *True* **note** *ltr14* = *True*
hence *False* **using** *ltr1*(*2*) *ltr2*(*2*) **unfolding** *Opt.lcompletedFrom-def* **by** *auto*
thus *?thesis* **by** *auto*
next
case *False* **hence** *ltr14*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] **by** *auto*
show *?thesis* **proof**(*cases llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)
case *True* **note** *ltr14* = *ltr14 True*
hence *ltrv1*: *list-of ltrv1* = [*sv1*] **unfolding** *ltrv1* **by** *simp*
have *llength ltr1* = *Suc 0* ∨ *llength ltr2* = *Suc 0*
using *ltr14*
by (*metis Opt.lcompletedFrom-def*
Suc-ile-eq i0-less lfinite-code(*1*) *llength-eq-0 llist.exhaust*
ltr1(*2*) *ltr2*(*2*) *nle-le not-lnull-conv zero-enat-def*)
hence *ltr1* = [[*s1*]] ∨ *ltr2* = [[*s2*]]
using *Opt.lcompletedFrom-singl ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) **by** *blast*
hence *finalO s1* ∨ *finalO s2*
using *Opt.lcompletedFrom-LCons ltr1*(*2*) *ltr2*(*2*) **by** *blast*
hence *finalV sv1*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *auto*
thus *?thesis* **unfolding** *ltrv1* **by** *auto*
next
case *False* **hence** *current*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0* **by** *auto*
show *?thesis*
proof(*cases trn*)
case *L* **note** *current* = *current L*
show *?thesis*
proof(*cases lnever isSecO ltr1*)
case *True* **note** *current* = *current True*
obtain *trn′ w1′ w2′ s1′ ltr1′ trv1 sv1′ trv2 sv2′ statOO* **where**
ωω: *ltr1* = *s1* $ *ltr1′ validTransO* (*s1*, *s1′*) *Opt.lvalidFromS s1′ ltr1′*
lcompletedFromO s1′ ltr1′ lnever isIntO ltr1′ **and**
ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
and *trn′* : *trn′* = (*if trv1* = [] *then L else R*)
and *lltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*, *sv1′*, *sv2′*, *statOO*))
using *lltrv1-lltrv2-lnever-L*[*OF unw* Δ *r ltr1 isi4 current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1′* **where** *ltrv1′*: *ltrv1′* = *lltrv1* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*, *sv1′*, *sv2′*, *statOO*)
have *lltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1′*
unfolding *lltrv1 ltrv1′* **..**

have *trv1ne*: *trv1* ≠ [] ∨ *w1′* < *w1* **using** *ω3* **unfolding** *ω3-def* **by** *auto*

```
have lfin': lfinite ltrv1'
using lfin trv1ne unfolding lltrv1 by simp
have len: length (list-of ltrv1') < length (list-of ltrv1) ∨
        length (list-of ltrv1') = length (list-of ltrv1) ∧ w1' < w1
using trv1ne lfin lfin' by (simp add: list-of-lappend lltrv1)

have 0: list-of ltrv1' ≠ [] ∧ finalV (last (list-of ltrv1'))
using len proof(elim disjE conjE)
  assume len: length (list-of ltrv1') < length (list-of ltrv1)
  show ?thesis
  apply(rule less(1)[OF - ltrv1'])
    subgoal by fact subgoal by fact
    subgoal using ω3 unfolding ω3-def by simp
    subgoal by (metis Opt.reach.Step ωω(2) fst-conv r(1) snd-conv)
    subgoal by fact
    subgoal using ω3 unfolding ω3-def
    by (metis Van.reach-validFromS-reach r(3) snoc-eq-iff-butlast)
    subgoal using ω3 unfolding ω3-def
     by (metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2 r(4))
   subgoal by fact subgoal by fact subgoal by fact  subgoal by fact
    subgoal by fact subgoal by fact .
next
  assume len: length (list-of ltrv1') = length (list-of ltrv1) w1' < w1
  show ?thesis
  apply(rule less(2)[OF - - ltrv1'])
      subgoal by fact subgoal using len by simp subgoal by fact

    subgoal using ω3 unfolding ω3-def by simp
    subgoal by (metis Opt.reach.Step ωω(2) fst-conv r(1) snd-conv)
    subgoal by fact
    subgoal using ω3 unfolding ω3-def
    by (metis Van.reach-validFromS-reach r(3) snoc-eq-iff-butlast)
    subgoal using ω3 unfolding ω3-def
     by (metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2 r(4))
   subgoal by fact subgoal by fact subgoal by fact  subgoal by fact
    subgoal by fact subgoal by fact .
qed
show ?thesis unfolding lltrv1 using 0
by (simp add: lfin' list-of-lappend)
next
  case False note current = current False
  obtain w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO where
  χχ: χχ s1 ltr1 tr1 s1' s1'' ltr1' and
   χ3': χ3' Δ w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2
trv2 sv2'' statOO
  and lltrv1: ltrv1 =
 lappend (llist-of trv1) (lltrv1 (R,w1',w2',s1'',s1'' $ ltr1',s2,ltr2,statA,sv1'',sv2'',statOO))
```

using *lltrv1-lltrv2-not-lnever-L*[*OF unw* Δ *r ltr1 isi4 current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1'* **where** *ltrv1'*: *ltrv1'* = *lltrv1* (*R*,*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2*,*ltr2*,*statA*,*sv1''*,*sv2''*,*statOO*)

have *lltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1'*
unfolding *lltrv1 ltrv1'* **..**
have *trv1ne*: *trv1* ≠ [] **using** *χ3'* **unfolding** *χ3'-def* **by** *auto*
have *lfin'*: *lfinite ltrv1'*
using *lfin trv1ne* **unfolding** *lltrv1* **by** *simp*
have *len*: *length* (*list-of ltrv1'*) < *length* (*list-of ltrv1*)
using *trv1ne lfin lfin'* **by** (*simp add*: *list-of-lappend lltrv1*)

have *0*: *list-of ltrv1'* ≠ [] ∧ *finalV* (*last* (*list-of ltrv1'*))
apply(*rule less*(*1*)[*OF - ltrv1'*])
subgoal by *fact* **subgoal by** *fact*
subgoal using *χ3'* **unfolding** *χ3'-def* **by** *simp*
subgoal using *χχ* **unfolding** *χχ-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*1*) *snoc-eq-iff-butlast*)
subgoal by *fact*
subgoal using *χ3'* **unfolding** *χ3'-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc r*(*3*))
subgoal using *χ3'* **unfolding** *χ3'-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal using *χχ* **unfolding** *χχ-def* **by** *simp*
subgoal using *χχ* **unfolding** *χχ-def* **by** *simp*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr1* **by** *blast*
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **.**
show *?thesis* **unfolding** *lltrv1* **using** *0*
by (*simp add*: *lfin' list-of-lappend*)
qed
next
case *R* **note** *current* = *current R*
show *?thesis*
proof(*cases lnever isSecO ltr2*)
case *True* **note** *current* = *current True*
obtain *trn' w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO* **where**
ωω: *ltr2* = *s2* $ *ltr2' validTransO* (*s2*, *s2'*) *Opt.lvalidFromS s2' ltr2'*
lcompletedFromO s2' ltr2' lnever isIntO ltr2' **and**
ω4: *ω4* Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*
and *trn'*: *trn'* = (*if trv2* = [] *then R else L*)
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*,

sv1′, *sv2′*, *statOO*))
using *lltrv1-lltrv2-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1′* **where** *ltrv1′*: *ltrv1′* = *lltrv1* (*trn′*, *w1′*, *w2′*, *s1*, *ltr1*, *s2′*, *ltr2′*, *statA*, *sv1′*, *sv2′*, *statOO*)
have *lltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1′*
unfolding *ltrv1 ltrv1′* **..**

have *trv1ne*: *trv1* ≠ [] ∨ *w1′* < *w1* **using** *ω4* **unfolding** *ω4-def* **by** *auto*
have *lfin′*: *lfinite ltrv1′*
using *lfin trv1ne* **unfolding** *lltrv1* **by** *simp*
have *len*: *length* (*list-of ltrv1′*) < *length* (*list-of ltrv1*) ∨
length (*list-of ltrv1′*) = *length* (*list-of ltrv1*) ∧ *w1′* < *w1*
using *trv1ne lfin lfin′* **by** (*simp add*: *list-of-lappend lltrv1*)

have *0*: *list-of ltrv1′* ≠ [] ∧ *finalV* (*last* (*list-of ltrv1′*))
using *len* **proof**(*elim disjE conjE*)
assume *len*: *length* (*list-of ltrv1′*) < *length* (*list-of ltrv1*)
show *?thesis*
apply(*rule less*(*1*)[*OF - ltrv1′*])
subgoal by *fact* **subgoal by** *fact*
subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal using *r*(*2*) *ωω* **by** (*metis Opt.reach.Step fst-conv snd-conv*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*3*))
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal by *fact* **subgoal by** *fact* **.**
next
assume *len*: *length* (*list-of ltrv1′*) = *length* (*list-of ltrv1*) *w1′* < *w1*
show *?thesis*
apply(*rule less*(*2*)[*OF - - ltrv1′*])
subgoal by *fact* **subgoal using** *len* **by** *simp* **subgoal by** *fact*

subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal by *fact* **subgoal by** *fact* **.**

qed
show *?thesis* **unfolding** *lltrv1* **using** *0*
by (*simp add*: *lfin′ list-of-lappend*)
next
case *False* **note** *current* = *current False*
obtain *w1′ w2′ tr2 s2′ s2″ ltr2′ trv1 sv1″ trv2 sv2″ statOO* **where**
χχ: *χχ s2 ltr2 tr2 s2′ s2″ ltr2′* **and**
χ4′: *χ4′ Δ w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2″ statA sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2″*, *s2″* $ *ltr2′*, *statA*, *sv1″*, *sv2″*, *statOO*))
using *lltrv1-lltrv2-not-lnever-R*[*OF unw Δ r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1′* **where** *ltrv1′*: *ltrv1′* = *lltrv1* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2″*, *s2″* $ *ltr2′*, *statA*, *sv1″*, *sv2″*, *statOO*)
have *lltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1′*
unfolding *ltrv1 ltrv1′* **..**

have *trv1ne*: *trv1* ≠ [] ∨ *w1′* < *w1* **using** *χ4′* **unfolding** *χ4′-def* **by** *auto*
have *lfin′*: *lfinite ltrv1′*
using *lfin trv1ne* **unfolding** *lltrv1* **by** *simp*
have *len*: *length* (*list-of ltrv1′*) < *length* (*list-of ltrv1*) ∨
length (*list-of ltrv1′*) = *length* (*list-of ltrv1*) ∧ *w1′* < *w1*
using *trv1ne lfin lfin′* **by** (*simp add*: *list-of-lappend lltrv1*)

have *0*: *list-of ltrv1′* ≠ [] ∧ *finalV* (*last* (*list-of ltrv1′*))
using *len* **proof**(*elim disjE conjE*)
assume *len*: *length* (*list-of ltrv1′*) < *length* (*list-of ltrv1*)
show *?thesis*
apply(*rule less*(*1*)[*OF - ltrv1′*])
subgoal by *fact* **subgoal by** *fact*
subgoal using *χ4′* **unfolding** *χ4′-def* **by** *simp*
subgoal by *fact*
subgoal using *r*(*2*) *χχ* **unfolding** *χχ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *χ4′* **unfolding** *χ4′-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*3*))
subgoal using *χ4′* **unfolding** *χ4′-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def*

using *llist-all-lappend-llist-of ltr2*(*3*) **by** *blast* **.**
next
assume *len*: *length* (*list-of ltrv1′*) = *length* (*list-of ltrv1*) *w1′* < *w1*
show *?thesis*
apply(*rule less*(*2*)[*OF* - - *ltrv1′*])
subgoal by *fact* **subgoal using** *len* **by** *simp* **subgoal by** *fact*

subgoal using $\chi 4'$ **unfolding** $\chi 4'$*-def* **by** *simp*
subgoal by *fact*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*2*))
subgoal using $\chi 4'$ **unfolding** $\chi 4'$*-def*
by (*metis Van.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using $\chi 4'$ **unfolding** $\chi 4'$*-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def*
using *llist-all-lappend-llist-of ltr2*(*3*) **by** *blast* **.**
qed
show *?thesis* **unfolding** *lltrv1* **using** *0*
by (*simp add*: *lfin′ list-of-lappend*)
qed
qed
qed
qed
qed
}
thus *?thesis* **unfolding** *Van.lcompletedFrom-def* **by** *auto*
qed

lemma *lcompletedFrom-lltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lcompletedFrom sv2* (*lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
proof−
{**fix** *ltrv2* **assume** *ltrv2*: *ltrv2* = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *lfin*: *lfinite ltrv2*
hence *list-of ltrv2* ≠ [] ∧ *finalV* (*last* (*list-of ltrv2*))
using *assms*(*2*−) **proof**(*induct length* (*list-of ltrv2*) *w2*
arbitrary: *ltrv2 trn w1 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*
rule: *less2-induct′*)
case (*less w2 ltrv2 trn w1 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*)

hence *ltrv2*: *ltrv2* = *lltrv2* (*trn*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
and *lfin*: *lfinite ltrv2*
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 lcompletedFromO s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 lcompletedFromO s2 ltr2 lnever isIntO ltr2*
by *auto*
have *isi3*: ¬ *isIntO s1* **using** *ltr1*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))
have *isi4*: ¬ *isIntO s2* **using** *ltr2*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))

show *?case* **proof**(*cases ltr1* = [[]] ∧ *ltr2* = [[]])
case *True* **note** *ltr14* = *True*
hence *False* **using** *ltr1*(*2*) *ltr2*(*2*) **unfolding** *Opt.lcompletedFrom-def* **by** *auto*
thus *?thesis* **by** *auto*
next
case *False* **hence** *ltr14*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] **by** *auto*
show *?thesis* **proof**(*cases llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)
case *True* **note** *ltr14* = *ltr14 True*
hence *ltrv2*: *list-of ltrv2* = [*sv2*] **unfolding** *ltrv2* **by** *simp*
have *llength ltr1* = *Suc 0* ∨ *llength ltr2* = *Suc 0*
using *ltr14*
by (*metis Opt.lcompletedFrom-def*
Suc-ile-eq i0-less lfinite-code(*1*) *llength-eq-0 llist.exhaust*
ltr1(*2*) *ltr2*(*2*) *nle-le not-lnull-conv zero-enat-def*)
hence *ltr1* = [[*s1*]] ∨ *ltr2* = [[*s2*]]
using *Opt.lcompletedFrom-singl ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) **by** *blast*
hence *finalO s1* ∨ *finalO s2*
using *Opt.lcompletedFrom-LCons ltr1*(*2*) *ltr2*(*2*) **by** *blast*
hence *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *auto*
thus *?thesis* **unfolding** *ltrv2* **by** *auto*
next
case *False* **hence** *current*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0* **by** *auto*
show *?thesis*
proof(*cases trn*)
case *L* **note** *current* = *current L*
show *?thesis*
proof(*cases lnever isSecO ltr1*)
case *True* **note** *current* = *current True*
obtain *trn′ w1′ w2′ s1′ ltr1′ trv1 sv1′ trv2 sv2′ statOO* **where**
ωω: *ltr1* = *s1* $ *ltr1′ validTransO* (*s1*, *s1′*) *Opt.lvalidFromS s1′ ltr1′*
lcompletedFromO s1′ ltr1′ lnever isIntO ltr1′ **and**

ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
and *trn′* : *trn′* = (*if trv1* = [] *then L else R*)
and *lltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*, *sv1′*, *sv2′*, *statOO*))
using *lltrv1-lltrv2-lnever-L*[*OF unw* Δ *r ltr1 isi4 current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2′* **where** *ltrv2′*: *ltrv2′* = *lltrv2* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*, *sv1′*, *sv2′*, *statOO*)
have *lltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2′*
unfolding *lltrv2 ltrv2′* **..**

have *trv2ne*: *trv2* ≠ [] ∨ *w2′* < *w2* **using** *ω3* **unfolding** *ω3-def* **by** *auto*
have *lfin′*: *lfinite ltrv2′*
using *lfin trv2ne* **unfolding** *lltrv2* **by** *simp*
have *len*: *length* (*list-of ltrv2′*) < *length* (*list-of ltrv2*) ∨
length (*list-of ltrv2′*) = *length* (*list-of ltrv2*) ∧ *w2′* < *w2*
using *trv2ne lfin lfin′* **by** (*simp add*: *list-of-lappend lltrv2*)

have *0*: *list-of ltrv2′* ≠ [] ∧ *finalV* (*last* (*list-of ltrv2′*))
using *len* **proof**(*elim disjE conjE*)
assume *len*: *length* (*list-of ltrv2′*) < *length* (*list-of ltrv2*)
show *?thesis*
apply(*rule less*(*1*)[*OF - ltrv2′*])
subgoal by *fact* **subgoal by** *fact*
subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Van.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal by *fact* **subgoal by** *fact* **.**
next
assume *len*: *length* (*list-of ltrv2′*) = *length* (*list-of ltrv2*) *w2′* < *w2*
show *?thesis*
apply(*rule less*(*2*)[*OF - - ltrv2′*])
subgoal by *fact* **subgoal using** *len* **by** *simp* **subgoal by** *fact*

subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Van.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω3* **unfolding** *ω3-def*

by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal by *fact* **subgoal by** *fact* **.**
qed
show *?thesis* **unfolding** *lltrv2* **using** *0*
by (*simp add*: *lfin′ list-of-lappend*)
next
case *False* **note** *current* = *current False*
obtain *w1′ w2′ tr1 s1′ s1″ ltr1′ trv1 sv1″ trv2 sv2″ statOO* **where**
$\chi\chi$: $\chi\chi$ *s1 ltr1 tr1 s1′ s1″ ltr1′* **and**
$\chi 3'$: $\chi 3'$ Δ *w1 w2 w1′ w2′ s1 tr1 s1′ s1″ s2 statA sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
and *lltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*R*,*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2*,*ltr2*,*statA*,*sv1″*,*sv2″*,*statOO*))

using *lltrv1-lltrv2-not-lnever-L*[*OF unw* Δ *r ltr1 isi4 current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2′* **where** *ltrv2′*: *ltrv2′* = *lltrv2* (*R*,*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2*,*ltr2*,*statA*,*sv1″*,*sv2″*,*statOO*)
have *lltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2′*
unfolding *lltrv2 ltrv2′* **..**

have *trv2ne*: *trv2* ≠ [] ∨ *w2′* < *w2* **using** $\chi 3'$ **unfolding** $\chi 3'$*-def* **by** *auto*
have *lfin′*: *lfinite ltrv2′*
using *lfin trv2ne* **unfolding** *lltrv2* **by** *simp*
have *len*: *length* (*list-of ltrv2′*) < *length* (*list-of ltrv2*) ∨
length (*list-of ltrv2′*) = *length* (*list-of ltrv2*) ∧ *w2′* < *w2*
using *trv2ne lfin lfin′* **by** (*simp add*: *list-of-lappend lltrv2*)

have *0*: *list-of ltrv2′* ≠ [] ∧ *finalV* (*last* (*list-of ltrv2′*))
using *len* **proof**(*elim disjE conjE*)
assume *len*: *length* (*list-of ltrv2′*) < *length* (*list-of ltrv2*)
show *?thesis*
apply(*rule less*(*1*)[*OF - ltrv2′*])
subgoal by *fact* **subgoal by** *fact*
subgoal using $\chi 3'$ **unfolding** $\chi 3'$*-def* **by** *simp*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*1*) *snoc-eq-iff-butlast*)
subgoal by *fact*
subgoal using $\chi 3'$ **unfolding** $\chi 3'$*-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc r*(*3*))
subgoal using $\chi 3'$ **unfolding** $\chi 3'$*-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *simp*

subgoal using *χχ* **unfolding** *χχ-def* **by** *simp*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr1* **by** *blast*
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **.**
next
assume *len*: *length* (*list-of ltrv2'*) = *length* (*list-of ltrv2*) *w2'* < *w2*
show *?thesis*
apply(*rule less*(*2*)[*OF* - - *ltrv2'*])
subgoal by *fact* **subgoal using** *len* **by** *simp* **subgoal by** *fact*

subgoal using *χ3'* **unfolding** *χ3'-def* **by** *simp*
subgoal using *χχ* **unfolding** *χχ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*1*))
subgoal by *fact*
subgoal using *χ3'* **unfolding** *χ3'-def*
by (*metis Van.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *χ3'* **unfolding** *χ3'-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **.**
qed
show *?thesis* **unfolding** *lltrv2* **using** *0*
by (*simp add*: *lfin' list-of-lappend*)
qed
next
case *R* **note** *current* = *current R*
show *?thesis*
proof(*cases lnever isSecO ltr2*)
case *True* **note** *current* = *current True*
obtain *trn' w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO* **where**
ωω: *ltr2* = *s2* $ *ltr2' validTransO* (*s2*, *s2'*) *Opt.lvalidFromS s2' ltr2'*
lcompletedFromO s2' ltr2' lnever isIntO ltr2' **and**
ω4: *ω4* Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*
and *trn'*: *trn'* = (*if trv2* = [] *then R else L*)
and *ltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*, *sv1'*, *sv2'*, *statOO*))
using *lltrv1-lltrv2-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2'* **where** *ltrv2'*: *ltrv2'* = *lltrv2* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*, *sv1'*, *sv2'*, *statOO*)
have *lltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2'*
unfolding *ltrv2 ltrv2'* **..**

have *trv2ne*: *trv2* ≠ [] ∨ *w2′* < *w2* **using** *ω4* **unfolding** *ω4-def* **by** *auto*
have *lfin′*: *lfinite ltrv2′*
using *lfin trv2ne* **unfolding** *lltrv2* **by** *simp*
have *len*: *length* (*list-of ltrv2′*) < *length* (*list-of ltrv2*) ∨
length (*list-of ltrv2′*) = *length* (*list-of ltrv2*) ∧ *w2′* < *w2*
using *trv2ne lfin lfin′* **by** (*simp add*: *list-of-lappend lltrv2*)

have *0*: *list-of ltrv2′* ≠ [] ∧ *finalV* (*last* (*list-of ltrv2′*))
using *len* **proof**(*elim disjE conjE*)
assume *len*: *length* (*list-of ltrv2′*) < *length* (*list-of ltrv2*)
show *?thesis*
apply(*rule less*(*1*)[*OF - ltrv2′*])
subgoal by *fact* **subgoal by** *fact*
subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal using *r*(*2*) *ωω* **by** (*metis Opt.reach.Step fst-conv snd-conv*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*3*))
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal by *fact* **subgoal by** *fact* **.**
next
assume *len*: *length* (*list-of ltrv2′*) = *length* (*list-of ltrv2*) *w2′* < *w2*
show *?thesis*
apply(*rule less*(*2*)[*OF - - ltrv2′*])
subgoal by *fact* **subgoal using** *len* **by** *simp* **subgoal by** *fact*

subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal by *fact* **subgoal by** *fact* **.**
qed
show *?thesis* **unfolding** *lltrv2* **using** *0*
by (*simp add*: *lfin′ list-of-lappend*)
next
case *False* **note** *current* = *current False*
obtain *w1′ w2′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statOO* **where**
χχ: *χχ s2 ltr2 tr2 s2′ s2′′ ltr2′* **and**

χ4′: *χ4′* Δ *w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2″ statA sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
and *ltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2″*, *s2″* $ *ltr2′*, *statA*, *sv1″*, *sv2″*, *statOO*))
using *lltrv1-lltrv2-not-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2′* **where** *ltrv2′*: *ltrv2′* = *lltrv2* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2″*, *s2″* $ *ltr2′*, *statA*, *sv1″*, *sv2″*, *statOO*)
have *lltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2′*
unfolding *ltrv2 ltrv2′* **..**

have *trv2ne*: *trv2* ≠ [] **using** *χ4′* **unfolding** *χ4′-def* **by** *auto*
have *lfin′*: *lfinite ltrv2′*
using *lfin trv2ne* **unfolding** *lltrv2* **by** *simp*
have *len*: *length* (*list-of ltrv2′*) < *length* (*list-of ltrv2*)
using *trv2ne lfin lfin′* **by** (*simp add*: *list-of-lappend lltrv2*)

have *0*: *list-of ltrv2′* ≠ [] ∧ *finalV* (*last* (*list-of ltrv2′*))
apply(*rule less*(*1*)[*OF* - *ltrv2′*])
subgoal by *fact* **subgoal by** *fact*
subgoal using *χ4′* **unfolding** *χ4′-def* **by** *simp*
subgoal by *fact*
subgoal using *r*(*2*) *χχ* **unfolding** *χχ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *χ4′* **unfolding** *χ4′-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2 r*(*3*))
subgoal using *χ4′* **unfolding** *χ4′-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc r*(*4*))
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr2*(*3*) **by** *blast* **.**
show *?thesis* **unfolding** *lltrv2* **using** *0*
by (*simp add*: *lfin′ list-of-lappend*)
qed
qed
qed
qed
qed
}
thus *?thesis* **unfolding** *Van.lcompletedFrom-def* **by** *auto*
qed

lemma *lS-lltrv1-ltr1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lS* (*lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) = *Opt.lS ltr1*
proof−
have *cltrv1*: *Van.lcompletedFrom sv1* (*lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-lltrv1*[*OF assms*] **.**
{**fix** *trn nL nR ltrv1 ltr1*
assume ∃ *w1 w2 s1 s2 ltr2 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w2* ∧
ltrv1 = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*
hence *TwoFuncPred.sameFM1 isSecV isSecO getSecV getSecO trn nL nR ltrv1 ltr1*
proof(*coinduct rule*: *TwoFuncPred.sameFM1.coinduct*[*of* λ*trn nL nR ltrv1 ltr1*.

∃ *w1 w2 s1 s2 ltr2 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w2* ∧
ltrv1 = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*,
where *pred* = *isSecV* **and** *pred′* = *isSecO* **and** *func* = *getSecV* **and** *func′* = *getSecO*])
case (*2 trn nL nR ltrv1 ltr1*)
then obtain *w1 w2 sv1 s1 s2 ltr2 statA sv2 statO*
where *nL*: *nL* = *w1* **and** *nR*: *nR* = *w2*
and *ltrv1*: *ltrv1* = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
by *auto*
have *isi3*: ¬ *isIntO s1* **using** *ltr1*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))
have *isi4*: ¬ *isIntO s2* **using** *ltr2*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))

show *?case* **proof**(*cases ltr1* = [[]] ∧ *ltr2* = [[]])
case *True* **note** *ltr14* = *True*
hence *ltrv1*: *ltrv1* = [[]] **unfolding** *ltrv1* **by** *simp*
show *?thesis* **using** *ltr14* **unfolding** *ltrv1* **apply**−**apply**(*rule TwoFuncPred.sameFM1-selectLNil*)
by *auto*
next
case *False* **hence** *ltr14*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] **by** *auto*
show *?thesis* **proof**(*cases llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)
case *True* **note** *ltr14* = *ltr14 True*
hence *ltrv1*: *ltrv1* = [[*sv1*]] **unfolding** *ltrv1* **by** *simp*
have *llength ltr1* = *Suc 0* ∨ *llength ltr2* = *Suc 0*
by (*metis Opt.lcompletedFrom-def Suc-ile-eq True*
lfinite-LNil llength-LNil llist-eq-cong ltr1(*2*)
ltr2(*2*) *nle-le order-le-imp-less-or-eq zero-enat-def zero-order*(*3*))
hence *finalO s1* ∨ *finalO s2*
using *Opt.lcompletedFrom-singl ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) **by** *blast*
hence *fs1*: *finalO s1*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *auto*
hence *ltr1*: *ltr1* = [[*s1*]]
by (*metis Opt.final-def Opt.lcompletedFrom-def*
Opt.lvalidFromS-Cons-iff lfinite-code(*1*) *llist.exhaust ltr1*(*1*) *ltr1*(*2*))
have *fsv1*: *finalV sv1*
using Δ *fs1 r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-final* **by** *blast*
have *isv13*: ¬ *isSecV sv1* ∧ ¬ *isSecO s1*
using *fsv1 fs1 Opt.final-not-isSec Van.final-not-isSec* **by** *blast*
show *?thesis* **unfolding** *ltrv1 ltr1* **apply**(*rule TwoFuncPred.sameFM1-selectSingl*)

using *isv13* **by** *auto*
next
case *False* **hence** *current*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
by *auto*
show *?thesis* **proof**(*cases trn*)
case *L* **note** *trn* = *L*[*simp*] **note** *current* = *current L*
show *?thesis*
proof(*cases lnever isSecO ltr1*)
case *True* **note** *current* = *current True*
obtain *trn′ w1′ w2′ s1′ ltr1′ trv1 sv1′ trv2 sv2′ statOO* **where**
ωω: *ltr1* = *s1* $ *ltr1′ validTransO* (*s1*, *s1′*) *Opt.lvalidFromS s1′ ltr1′*
lcompletedFromO s1′ ltr1′ lnever isIntO ltr1′ **and**
ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′*
statOO
and *trn′*: *trn′* = (*if trv1* = [] *then L else R*)
and *lltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*,
sv1′, *sv2′*, *statOO*))
using *lltrv1-lltrv2-lnever-L*[*OF unw* Δ *r ltr1 isi4 current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1′* **where** *ltrv1′*: *ltrv1′* ≡ *lltrv1* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*,

s2, *ltr2*, *statA*, *sv1'*, *sv2'*, *statOO*)
have *ltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1'*
unfolding *lltrv1 ltrv1'* **..**
have *nis1*: ¬ *isSecO s1* **using** *True* ωω(*1*) **by** *force*
show *?thesis*
proof(*cases trv1* = [])
case *True* **note** *trv1* = *True*
hence *w1'* < *w1* **using** ω*3* **unfolding** ω*3-def* **by** *auto*
have [*simp*]: *trn'* = *trn* **by** (*simp add*: *trv1 trn'*)
show *?thesis*
apply(*rule TwoFuncPred.sameFM1-selectDelayL*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w1*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of -* [*s1*]])
apply(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - ltr1'*])
apply(*rule exI*[*of - w2*])
apply(*intro conjI*)
subgoal by *fact*
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal unfolding *ltrv1 trv1* **by** *simp*
subgoal unfolding ωω(*1*) **by** *simp*
subgoal by *fact* **subgoal unfolding** *trv1* **using** ω*3-def nis1* **by**
simp
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1'*]) **apply**(*rule exI*[*of - s2*])
apply(*rule exI*[*of - ltr2*]) **apply**(*rule exI*[*of - statA*])
apply(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. **subgoal ..**
subgoal unfolding *ltrv1'* **by** *simp*
subgoal using ω*3* **unfolding** ω*3-def* **by** *simp*
subgoal using ω*3* **unfolding** ω*3-def*
by (*metis Opt.reach.Step* ωω(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact*
subgoal using ω*3* **unfolding** ω*3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*)
snoc-eq-iff-butlast)
subgoal using ω*3* **unfolding** ω*3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*)
snoc-eq-iff-butlast)
subgoal using ωω **by** *auto*
subgoal using ωω **by** *auto*
subgoal using ωω
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal using ωω **using** *ltr2*(*1*) **by** *fastforce*
subgoal by *fact*
subgoal by *fact* **. .**

next
case *False* **note** *trv1* = *False*
show *?thesis*
apply(*rule TwoFuncPred.sameFM1-selectlappend*)
apply(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - [*s1*]])
apply(*rule exI*[*of* - *trn'*]) **apply**(*rule exI*[*of* - *w1'*])
apply(*rule exI*[*of* - *w2'*])
apply(*rule exI*[*of* - *ltrv1'*]) **apply**(*rule exI*[*of* - *ltr1'*])
apply(*rule exI*[*of* - *trn*])
apply(*rule exI*[*of* - *w1*])
apply(*rule exI*[*of* - *w2*])
apply(*intro conjI*)
subgoal ..
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal using *ltrv1* **.**
subgoal unfolding *ωω*(*1*) **by** *simp*
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal using *ltr1*(*3*) *ω3* **unfolding** *ω3-def*
by (*metis Opt.S.map-filter Opt.S.simps*(*4*) *Van.S.map-filter Van.S.eq-Nil-iff*(*2*) *append-Nil*
butlast-snoc filter.simps(*2*) *nis1*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of* - *w1'*]) **apply**(*rule exI*[*of* - *w2'*])
apply(*rule exI*[*of* - *s1'*]) **apply**(*rule exI*[*of* - *s2*])
apply(*rule exI*[*of* - *ltr2*]) **apply**(*rule exI*[*of* - *statA*])
apply(*rule exI*[*of* - *sv1'*]) **apply**(*rule exI*[*of* - *sv2'*])
apply(*rule exI*[*of* - *statOO*])
apply(*intro conjI*)
subgoal .. **subgoal ..**
subgoal unfolding *ltrv1'* **..**
subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal using *ωω*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal using *ωω* **using** *ltr2*(*1*) **by** *fastforce*
subgoal by *fact*
subgoal by *fact* **. .**
qed

next
case *False* **note** *current* = *current False*
obtain *w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO* **where**
χχ: *χχ s1 ltr1 tr1 s1' s1'' ltr1'* **and**
χ3': *χ3' Δ w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
and *lltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*R*,*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2*,*ltr2*,*statA*,*sv1''*,*sv2''*,*statOO*))

using *lltrv1-lltrv2-not-lnever-L*[*OF unw Δ r ltr1 isi4 current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1'* **where** *ltrv1'*: *ltrv1'* ≡ *lltrv1* (*R*,*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2*,*ltr2*,*statA*,*sv1''*,*sv2''*,*statOO*)
have *ltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1'*
unfolding *lltrv1 ltrv1'* **..**

show *?thesis* **apply**(*rule TwoFuncPred.sameFM1-selectlappend*)
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - tr1 ## s1'*])
apply(*rule exI*[*of - R*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - s1''* $ *ltr1'*])
apply(*rule exI*[*of - trn*])
apply(*rule exI*[*of - w1*]) **apply**(*rule exI*[*of - w2*])
apply(*intro conjI*)
subgoal .. **subgoal unfolding** *nL* **..** **subgoal unfolding** *nR* **..**
subgoal using *ltrv1* **.**
subgoal using *χχ* **unfolding** *χχ-def* **by** *simp*
subgoal using *χ3'* **unfolding** *χ3'-def* **by** *simp*
subgoal by *simp*
subgoal using *χ3'* **unfolding** *χ3'-def*
by (*simp add*: *Opt.S.map-filter Van.S.map-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1''*])
apply(*rule exI*[*of - s2*]) **apply**(*rule exI*[*of - ltr2*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - sv2''*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. **subgoal ..**
subgoal unfolding *ltrv1'* **..**
subgoal using *χ3'* **unfolding** *χ3'-def* **by** *simp*
subgoal using *χ3'* **unfolding** *χ3'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach χχ χχ-def*
append-is-Nil-conv last-snoc not-Cons-self2 r(*1*))
subgoal by *fact*
subgoal using *χ3'* **unfolding** *χ3'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*)

snoc-eq-iff-butlast)
subgoal using $\chi 3'$ **unfolding** $\chi 3'$*-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*)
snoc-eq-iff-butlast)
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **using** *ltr2*(*1*) **by** *fastforce*
subgoal by *fact*
subgoal by *fact* **. .**
qed
next
case *R* **note** *trn* = *R*[*simp*] **note** *current* = *current R*
show *?thesis*
proof(*cases lnever isSecO ltr2*)
case *True* **note** *current* = *current True*
obtain *trn' w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO* **where**
$\omega\omega$: *ltr2* = *s2* $ *ltr2' validTransO* (*s2*, *s2'*) *Opt.lvalidFromS s2' ltr2'*
lcompletedFromO s2' ltr2' lnever isIntO ltr2' **and**
$\omega 4$: $\omega 4$ Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2'*
statOO
and *trn'*: *trn'* = (*if trv2* = [] *then R else L*)
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*,
sv1', *sv2'*, *statOO*))
using *lltrv1-lltrv2-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1'* **where** *ltrv1'*: *ltrv1'* ≡ *lltrv1* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*,
ltr2', *statA*, *sv1'*, *sv2'*, *statOO*)
have *ltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1'*
unfolding *ltrv1 ltrv1'* **..**
have *nev1*: *never isSecV trv1* **using** $\omega 4$ **unfolding** $\omega 4$*-def* **by** *auto*
show *?thesis*
proof(*cases trv2* = [])
case *True* **note** *trv2* = *True*
have [*simp*]: *trn'* = *trn* **using** *R trv2 trn'* **by** *auto*
have *w2'* < *w2* **using** $\omega 4$ *trv2* **unfolding** $\omega 4$*-def* **by** *auto*
show *?thesis*
apply(*rule TwoFuncPred.sameFM1-selectDelayR*)
apply(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - nR*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of -* []])
apply(*rule exI*[*of - w1'*])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - nL*])
apply(*intro conjI*)
subgoal by *simp* **subgoal .. subgoal ..**
subgoal by *fact* **subgoal by** *simp*
subgoal unfolding *nR* **by** *fact*

subgoal using *nev1* **by** (*simp add*: *never-Nil-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - s2'*])
apply(*rule exI*[*of - ltr2'*]) **apply**(*rule exI*[*of - statA*])
apply(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv1'* **by** *simp*
subgoal using $\omega 4$ **unfolding** $\omega 4$*-def* **by** *simp*
subgoal by *fact*
subgoal using $\omega 4$ **unfolding** $\omega 4$*-def*
by (*metis Opt.reach.Step* $\omega\omega(2)$ *fst-conv* $r(2)$ *snd-conv*)
subgoal using $\omega 4$ **unfolding** $\omega 4$*-def*
by (*metis Van.reach-validFromS-reach* $r(3)$ *snoc-eq-iff-butlast*)
subgoal using $\omega 4$ **unfolding** $\omega 4$*-def*
by (*metis Van.reach-validFromS-reach* $r(4)$ *snoc-eq-iff-butlast*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using $\omega\omega$ **by** *auto*
subgoal using $\omega\omega$ **by** *auto*
subgoal using $\omega\omega$ **by** *auto* **. .**
next
case *False* **note** *trv2* = *False*
have [*simp*]: *trn'* = *L* **using** *R trv2 trn'* **by** *auto*
show *?thesis*
apply(*rule TwoFuncPred.sameFM1-selectRL*)
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of -* []])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - w1*]) **apply**(*rule exI*[*of - w2*])
apply(*intro conjI*)
subgoal by *fact*
subgoal unfolding *nL* **.. subgoal unfolding** *nR* **..**
subgoal unfolding *ltrv1* **..**
subgoal unfolding $\omega\omega(1)$ **by** *simp*
subgoal using $\omega 4$ **unfolding** $\omega 4$*-def* **by** (*simp add*: *never-Nil-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - s2'*])
apply(*rule exI*[*of - ltr2'*]) **apply**(*rule exI*[*of - statA*])
apply(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv1'* **by** *simp*
subgoal using $\omega 4$ **unfolding** $\omega 4$*-def* **by** *simp*
subgoal by *fact*
subgoal using $\omega 4$ **unfolding** $\omega 4$*-def*

by (*metis Opt.reach.Step* ωω(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal using ω*4* **unfolding** ω*4-def*
by (*metis Van.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using ω*4* **unfolding** ω*4-def*
by (*metis Van.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using ωω **by** *auto*
subgoal using ωω **by** *auto*
subgoal using ωω **by** *auto* **. .**
qed
next
case *False* **note** *current* = *current False*
obtain *w1′ w2′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statOO* **where**
χχ: χχ *s2 ltr2 tr2 s2′ s2′′ ltr2′* **and**
χ*4′*: χ*4′* Δ *w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2′′ statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2′′*, *s2′′* $ *ltr2′*, *statA*, *sv1′′*, *sv2′′*, *statOO*))
using *lltrv1-lltrv2-not-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv1* **by** *blast*
define *ltrv1′* **where** *ltrv1′*: *ltrv1′* ≡ *lltrv1* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2′′*, *s2′′* $ *ltr2′*, *statA*, *sv1′′*, *sv2′′*, *statOO*)
have *ltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1′*
unfolding *ltrv1 ltrv1′* **..**

show *?thesis*
apply(*rule TwoFuncPred.sameFM1-selectRL*)
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of -* []])
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - ltrv1′*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - w1*]) **apply**(*rule exI*[*of - w2*])
apply(*intro conjI*)
subgoal by *fact*
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal unfolding *ltrv1* **..** **subgoal by** *simp*
subgoal using χ*4′* **unfolding** χ*4′-def* **by** (*simp add*: *never-Nil-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - s2′′*])
apply(*rule exI*[*of - s2′′* $ *ltr2′*]) **apply**(*rule exI*[*of - statA*])
apply(*rule exI*[*of - sv1′′*]) **apply**(*rule exI*[*of - sv2′′*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. **subgoal ..**
subgoal unfolding *ltrv1′* **by** *simp*
subgoal using χ*4′* **unfolding** χ*4′-def* **by** *simp*
subgoal by *fact*

subgoal using *r*(*2*) *χχ* **unfolding** *χχ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) *χ4′* **unfolding** *χ4′-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) *χ4′* **unfolding** *χ4′-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr2*(*3*) **by** *blast* **. .**
qed
qed
qed
qed
qed
}
thus *?thesis* **unfolding** *Van.lS*[*OF cltrv1*] *Opt.lS*[*OF ltr1*(*2*)]
apply− **apply**(*rule TwoFuncPred.sameFM1-lmap-lfilter*)
using *assms* **by** *blast*
qed

lemma *lS-lltrv2-ltr2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lS* (*lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) = *Opt.lS ltr2*
proof−
have *cltrv2*: *Van.lcompletedFrom sv2* (*lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-lltrv2*[*OF assms*] **.**
{fix *trn nL nR ltrv2 ltr2*
assume ∃ *w1 w2 s1 s2 ltr1 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w2* ∧
ltrv2 = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*
hence *TwoFuncPred.sameFM2 isSecV isSecO getSecV getSecO trn nL nR ltrv2 ltr2*
proof(*coinduct rule*: *TwoFuncPred.sameFM2.coinduct*[*of λtrn nL nR ltrv2 ltr2*.

∃ *w1 w2 s1 s2 ltr1 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w2* ∧

ltrv2 = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*,
where *pred* = *isSecV* **and** *pred′* = *isSecO* **and** *func* = *getSecV* **and** *func′* = *getSecO*])
case (*2 trn nL nR ltrv2 ltr2*)
then obtain *w1 w2 sv1 s1 s2 ltr1 statA sv2 statO*
where *nL*: *nL* = *w1* **and** *nR*: *nR* = *w2*
and *ltrv2*: *ltrv2* = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
by *auto*
have *isi3*: ¬ *isIntO s1* **using** *ltr1*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))
have *isi4*: ¬ *isIntO s2* **using** *ltr2*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))

show *?case* **proof**(*cases ltr1* = [[]] ∧ *ltr2* = [[]])
case *True* **note** *ltr14* = *True*
hence *ltrv2*: *ltrv2* = [[]] **unfolding** *ltrv2* **by** *simp*
show *?thesis* **using** *ltr14* **unfolding** *ltrv2* **apply**−**apply**(*rule TwoFuncPred.sameFM2-selectLNil*) **by** *auto*
next
case *False* **hence** *ltr14*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] **by** *auto*
show *?thesis* **proof**(*cases llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)
case *True* **note** *ltr14* = *ltr14 True*
hence *ltrv2*: *ltrv2* = [[*sv2*]] **unfolding** *ltrv2* **by** *simp*
have *llength ltr1* = *Suc 0* ∨ *llength ltr2* = *Suc 0*
by (*metis Opt.lcompletedFrom-def Suc-ile-eq True*
lfinite-LNil llength-LNil llist-eq-cong ltr1(*2*)
ltr2(*2*) *nle-le order-le-imp-less-or-eq zero-enat-def zero-order*(*3*))
hence *finalO s1* ∨ *finalO s2*
using *Opt.lcompletedFrom-singl ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) **by** *blast*
hence *fs2*: *finalO s2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *auto*
hence *ltr2*: *ltr2* = [[*s2*]]
by (*metis Opt.final-def Opt.lcompletedFrom-def*
Opt.lvalidFromS-Cons-iff lfinite-code(*1*) *llist.exhaust ltr2*(*1*) *ltr2*(*2*))
have *fsv2*: *finalV sv2*
using Δ *fs2 r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-final* **by** *blast*
have *isv24*: ¬ *isSecV sv2* ∧ ¬ *isSecO s2*
using *fsv2 fs2 Opt.final-not-isSec Van.final-not-isSec* **by** *blast*

show *?thesis* **unfolding** *ltrv2 ltr2* **apply**(*rule TwoFuncPred.sameFM2-selectSingl*)

using *isv24* **by** *auto*
next
case *False* **hence** *current*: *llength ltr1 > Suc 0 llength ltr2 > Suc 0*
by *auto*
show *?thesis* **proof**(*cases trn*)
case *L* **note** *trn = L*[*simp*] **note** *current = current L*
show *?thesis*
proof(*cases lnever isSecO ltr1*)
case *True* **note** *current = current True*
obtain *trn′ w1′ w2′ s1′ ltr1′ trv1 sv1′ trv2 sv2′ statOO* **where**
ωω: *ltr1 = s1 $ ltr1′ validTransO (s1, s1′) Opt.lvalidFromS s1′ ltr1′*
lcompletedFromO s1′ ltr1′ lnever isIntO ltr1′ **and**
ω3: *ω3 Δ w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
and *trn′*: *trn′ = (if trv1 = [] then L else R)*
and *lltrv2*: *ltrv2 =*
lappend (llist-of trv2) (lltrv2 (trn′, w1′, w2′, s1′, ltr1′, s2, ltr2, statA, sv1′, sv2′, statOO))
using *lltrv1-lltrv2-lnever-L*[*OF unw Δ r ltr1 isi4 current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2′* **where** *ltrv2′*: *ltrv2′ ≡ lltrv2 (trn′, w1′, w2′, s1′, ltr1′, s2, ltr2, statA, sv1′, sv2′, statOO)*
have *ltrv2*: *ltrv2 = lappend (llist-of trv2) ltrv2′*
unfolding *lltrv2 ltrv2′* **..**
have *nev2*: *never isSecV trv2* **using** *ω3* **unfolding** *ω3-def* **by** *auto*
show *?thesis*
proof(*cases trv1 = []*)
case *True* **note** *trv1 = True*
have [*simp*]: *trn′ = trn* **using** *L trv1 trn′* **by** *auto*
have *w1′ < w1* **using** *ω3 trv1* **unfolding** *ω3-def* **by** *auto*
show *?thesis*
apply(*rule TwoFuncPred.sameFM2-selectDelayL*)
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - nL*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - []*])
apply(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - ltrv2′*]) **apply**(*rule exI*[*of - ltr2*])
apply(*rule exI*[*of - nR*])
apply(*intro conjI*)
subgoal by *simp* **subgoal .. subgoal ..**
subgoal by *fact* **subgoal by** *simp*
subgoal unfolding *nL* **by** *fact*
subgoal using *nev2* **by** (*simp add*: *never-Nil-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - s1′*]) **apply**(*rule exI*[*of - s2*])
apply(*rule exI*[*of - ltr1′*]) **apply**(*rule exI*[*of - statA*])
apply(*rule exI*[*of - sv1′*]) **apply**(*rule exI*[*of - sv2′*])

```
      apply(rule exI[of - statOO])
      apply(intro conjI)
        subgoal .. subgoal ..
        subgoal unfolding ltrv2' by simp
        subgoal using ω3 unfolding ω3-def by simp
        subgoal using ω3 unfolding ω3-def
        by (metis Opt.reach.Step ωω(2) fst-conv r(1) snd-conv)
        subgoal by fact
        subgoal using ω3 unfolding ω3-def
        by (metis Van.reach-validFromS-reach r(3) snoc-eq-iff-butlast)
        subgoal using ω3 unfolding ω3-def
        by (metis Van.reach-validFromS-reach r(4) snoc-eq-iff-butlast)
        subgoal using ωω by auto
        subgoal using ωω by auto
        subgoal using ωω by auto
    subgoal by fact subgoal by fact subgoal by fact . .
next
  case False note trv1 = False
  have [simp]: trn' = R using L trv1 trn' by auto
  show ?thesis
  apply(rule TwoFuncPred.sameFM2-selectLR)
  apply(rule exI[of - trv2]) apply(rule exI[of - []])
  apply(rule exI[of - w1']) apply(rule exI[of - w2'])
  apply(rule exI[of - ltrv2']) apply(rule exI[of - ltr2])
  apply(rule exI[of - w1]) apply(rule exI[of - w2])
  apply(intro conjI)
    subgoal by fact
    subgoal unfolding nL .. subgoal unfolding nR ..
    subgoal unfolding ltrv2 ..
    subgoal unfolding ωω(1) by simp
  subgoal using ω3 unfolding ω3-def by (simp add: never-Nil-filter)
    subgoal apply(rule disjI1)
    apply(rule exI[of - w1']) apply(rule exI[of - w2'])
    apply(rule exI[of - s1']) apply(rule exI[of - s2])
    apply(rule exI[of - ltr1']) apply(rule exI[of - statA])
    apply(rule exI[of - sv1']) apply(rule exI[of - sv2'])
    apply(rule exI[of - statOO])
    apply(intro conjI)
      subgoal .. subgoal ..
      subgoal unfolding ltrv2' by simp
        subgoal using ω3 unfolding ω3-def by simp
        subgoal using ω3 unfolding ω3-def
        by (metis Opt.reach.Step ωω(2) fst-conv r(1) snd-conv)
        subgoal by fact
        subgoal using ω3 unfolding ω3-def
        by (metis Van.reach-validFromS-reach r(3) snoc-eq-iff-butlast)
        subgoal using ω3 unfolding ω3-def
        by (metis Van.reach-validFromS-reach r(4) snoc-eq-iff-butlast)
        subgoal using ωω by auto
```

subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **. .**
qed
next
case *False* **note** *current = current False*
obtain *w1′ w2′ tr1 s1′ s1′′ ltr1′ trv1 sv1′′ trv2 sv2′′ statOO* **where**
χχ: *χχ s1 ltr1 tr1 s1′ s1′′ ltr1′* **and**
χ3′: *χ3′ Δ w1 w2 w1′ w2′ s1 tr1 s1′ s1′′ s2 statA sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
and *lltrv2*: *ltrv2 =*
lappend (*llist-of trv2*) (*lltrv2* (*R*,*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2*,*ltr2*,*statA*,*sv1′′*,*sv2′′*,*statOO*))

using *lltrv1-lltrv2-not-lnever-L*[*OF unw Δ r ltr1 isi4 current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2′* **where** *ltrv2′*: *ltrv2′ ≡ lltrv2* (*R*,*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2*,*ltr2*,*statA*,*sv1′′*,*sv2′′*,*statOO*)
have *ltrv2*: *ltrv2 = lappend* (*llist-of trv2*) *ltrv2′*
unfolding *lltrv2 ltrv2′* **..**

show *?thesis*
apply(*rule TwoFuncPred.sameFM2-selectLR*)
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of -* []])
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - ltrv2′*]) **apply**(*rule exI*[*of - ltr2*])
apply(*rule exI*[*of - w1*]) **apply**(*rule exI*[*of - w2*])
apply(*intro conjI*)
subgoal by *fact*
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal unfolding *ltrv2* **..** **subgoal by** *simp*
subgoal using *χ3′* **unfolding** *χ3′-def* **by** (*simp add*: *never-Nil-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - s1′′*]) **apply**(*rule exI*[*of - s2*])
apply(*rule exI*[*of - s1′′* $ *ltr1′*]) **apply**(*rule exI*[*of - statA*])
apply(*rule exI*[*of - sv1′′*]) **apply**(*rule exI*[*of - sv2′′*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. **subgoal ..**
subgoal unfolding *ltrv2′* **by** *simp*
subgoal using *χ3′* **unfolding** *χ3′-def* **by** *simp*
subgoal using *r*(*1*) *χχ* **unfolding** *χχ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal by *fact*
subgoal using *r*(*3*) *χ3′* **unfolding** *χ3′-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) *χ3′* **unfolding** *χ3′-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)

subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def* **by** *auto*
subgoal using *χχ* **unfolding** *χχ-def*
using *llist-all-lappend-llist-of ltr1(3)* **by** *blast*
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **. .**
qed
next
case *R* **note** *trn = R[simp]* **note** *current = current R*
show *?thesis*
proof(*cases lnever isSecO ltr2*)
case *True* **note** *current = current True*
obtain *trn′ w1′ w2′ s2′ ltr2′ trv1 sv1′ trv2 sv2′ statOO* **where**
ωω: *ltr2 = s2 $ ltr2′ validTransO (s2, s2′) Opt.lvalidFromS s2′ ltr2′*
lcompletedFromO s2′ ltr2′ lnever isIntO ltr2′ **and**
ω4: *ω4 Δ w1 w2 w1′ w2′ s1 s2 s2′ statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO*
and *trn′*: *trn′ = (if trv2 = [] then R else L)*
and *ltrv2*: *ltrv2 =*
lappend (llist-of trv2) (lltrv2 (trn′, w1′, w2′, s1, ltr1, s2′, ltr2′, statA, sv1′, sv2′, statOO))
using *lltrv1-lltrv2-lnever-R[OF unw Δ r ltr2(1,2) isi3 ltr2(3) current]*
unfolding *ltrv2* **by** *blast*
define *ltrv2′* **where** *ltrv2′*: *ltrv2′ ≡ lltrv2 (trn′, w1′, w2′, s1, ltr1, s2′, ltr2′, statA, sv1′, sv2′, statOO)*
have *ltrv2*: *ltrv2 = lappend (llist-of trv2) ltrv2′*
unfolding *ltrv2 ltrv2′* **..**
have *nis2*: *¬ isSecO s2* **using** *True ωω(1)* **by** *force*

show *?thesis*
proof(*cases trv2 = []*)
case *True* **note** *trv2 = True*
hence *w2′ < w2* **using** *ω4* **unfolding** *ω4-def* **by** *auto*
have *[simp]*: *trn′ = trn* **by** (*simp add*: *trv2 trn′*)
show *?thesis*
apply(*rule TwoFuncPred.sameFM2-selectDelayR*)
apply(*rule exI[of - w2′]*) **apply**(*rule exI[of - w2]*)
apply(*rule exI[of - trv2]*) **apply**(*rule exI[of - [s2]]*)
apply(*rule exI[of - w1′]*)
apply(*rule exI[of - ltrv2′]*) **apply**(*rule exI[of - ltr2′]*)
apply(*rule exI[of - w1]*)
apply(*intro conjI*)
subgoal by *fact*
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal unfolding *ltrv2 trv2* **by** *simp*
subgoal unfolding *ωω(1)* **by** *simp*
subgoal by *fact* **subgoal unfolding** *trv2* **using** *ω4-def nis2* **by** *simp*
subgoal apply(*rule disjI1*)
apply(*rule exI[of - w1′]*) **apply**(*rule exI[of - w2′]*)

apply(*rule exI*[*of* - *s1*]) **apply**(*rule exI*[*of* - *s2'*])
apply(*rule exI*[*of* - *ltr1*]) **apply**(*rule exI*[*of* - *statA*])
apply(*rule exI*[*of* - *sv1'*]) **apply**(*rule exI*[*of* - *sv2'*])
apply(*rule exI*[*of* - *statOO*])
apply(*intro conjI*)
subgoal .. **subgoal ..**
subgoal unfolding *ltrv2'* **by** *simp*
subgoal using $\omega 4$ **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal using $\omega 4$ **unfolding** *ω4-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal using $\omega 4$ **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using $\omega 4$ **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal using *ωω*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast* **. .**
next
case *False* **note** *trv2* = *False*
show *?thesis*
apply(*rule TwoFuncPred.sameFM2-selectlappend*)
apply(*rule exI*[*of* - *trv2*]) **apply**(*rule exI*[*of* - [*s2*]])
apply(*rule exI*[*of* - *trn'*]) **apply**(*rule exI*[*of* - *w1'*])
apply(*rule exI*[*of* - *w2'*])
apply(*rule exI*[*of* - *ltrv2'*]) **apply**(*rule exI*[*of* - *ltr2'*])
apply(*rule exI*[*of* - *trn*])
apply(*rule exI*[*of* - *w1*])
apply(*rule exI*[*of* - *w2*])
apply(*intro conjI*)
subgoal ..
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal using *ltrv2* **.**
subgoal unfolding *ωω*(*1*) **by** *simp*
subgoal by *fact*
subgoal using $\omega 4$ **unfolding** *ω4-def* **by** *simp*
subgoal using *ltr1*(*3*) $\omega 4$ **unfolding** *ω4-def*
by (*simp add*: *never-Nil-filter nis2*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of* - *w1'*]) **apply**(*rule exI*[*of* - *w2'*])
apply(*rule exI*[*of* - *s1*]) **apply**(*rule exI*[*of* - *s2'*])
apply(*rule exI*[*of* - *ltr1*]) **apply**(*rule exI*[*of* - *statA*])
apply(*rule exI*[*of* - *sv1'*]) **apply**(*rule exI*[*of* - *sv2'*])
apply(*rule exI*[*of* - *statOO*])
apply(*intro conjI*)

subgoal .. **subgoal** ..
subgoal unfolding *ltrv2′* ..
subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal using *ωω*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast* . .
qed
next
case *False* **note** *current* = *current False*
obtain *w1′ w2′ tr2 s2′ s2″ ltr2′ trv1 sv1″ trv2 sv2″ statOO* **where**
χχ: *χχ s2 ltr2 tr2 s2′ s2″ ltr2′* **and**
χ4′: *χ4′ Δ w1 w2 w1′ w2′ s1 s2 tr2 s2′ s2″ statA sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
and *ltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2″*, *s2″* $ *ltr2′*, *statA*, *sv1″*, *sv2″*, *statOO*))
using *lltrv1-lltrv2-not-lnever-R*[*OF unw Δ r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv2* **by** *blast*
define *ltrv2′* **where** *ltrv2′*: *ltrv2′* ≡ *lltrv2* (*L*, *w1′*, *w2′*, *s1*, *ltr1*, *s2″*, *s2″* $ *ltr2′*, *statA*, *sv1″*, *sv2″*, *statOO*)
have *ltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2′*
unfolding *ltrv2 ltrv2′* ..
show *?thesis*
apply(*rule TwoFuncPred.sameFM2-selectlappend*)
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - tr2 ## s2′*])
apply(*rule exI*[*of - L*])
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - ltrv2′*]) **apply**(*rule exI*[*of - s2″* $ *ltr2′*])
apply(*rule exI*[*of - trn*])
apply(*rule exI*[*of - w1*]) **apply**(*rule exI*[*of - w2*])
apply(*intro conjI*)
subgoal .. **subgoal unfolding** *nL* .. **subgoal unfolding** *nR* ..
subgoal using *ltrv2* .
subgoal using *χχ* **unfolding** *χχ-def* **by** *simp*
subgoal using *χ4′* **unfolding** *χ4′-def* **by** *simp*
subgoal by *simp*
subgoal using *χ4′* **unfolding** *χ4′-def*

```
        by (simp add: Opt.S.map-filter Van.S.map-filter)
        subgoal apply(rule disjI1)
        apply(rule exI[of - w1']) apply(rule exI[of - w2'])
        apply(rule exI[of - s1])
        apply(rule exI[of - s2'']) apply(rule exI[of - ltr1])
       apply(rule exI[of - statA]) apply(rule exI[of - sv1'']) apply(rule exI[of
- sv2''])
        apply(rule exI[of - statOO])
        apply(intro conjI)
          subgoal .. subgoal ..
          subgoal unfolding ltrv2' ..
          subgoal using χ4' unfolding χ4'-def by simp
          subgoal by fact
          subgoal using χ4' unfolding χ4'-def
       by (metis Simple-Transition-System.reach-validFromS-reach χχ χχ-def

          append-is-Nil-conv last-snoc not-Cons-self2 r(2))
          subgoal using χ4' unfolding χ4'-def
          by (metis Van.reach-validFromS-reach r(3) snoc-eq-iff-butlast)
          subgoal using χ4' unfolding χ4'-def
          by (metis Van.reach-validFromS-reach r(4) snoc-eq-iff-butlast)
          subgoal by fact subgoal by fact subgoal by fact
          subgoal using χχ unfolding χχ-def by auto
          subgoal using χχ unfolding χχ-def by auto
          subgoal using χχ unfolding χχ-def
          using llist-all-lappend-llist-of ltr2(3) by blast . .
      qed
     qed
    qed
   qed
  qed
  }
  thus ?thesis unfolding Van.lS[OF cltrv2] Opt.lS[OF ltr2(2)]
  apply− apply(rule TwoFuncPred.sameFM2-lmap-lfilter)
  using assms by blast
qed


lemma lA-lltrv1-lltrv2:
assumes unw: unwindCond Δ
and Δ: Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO
and r: reachO s1 reachO s2 reachV sv1 reachV sv2
and ltr1: Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1
and ltr2: Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2
shows Van.lA (lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)) =
      Van.lA (lltrv2 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))
proof−
 have cltrv1: Van.lcompletedFrom sv1 (lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))
```

using *lcompletedFrom-llltrv1*[*OF assms*] **.**
have *cltrv2*: *Van.lcompletedFrom sv2* (*lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-lltrv2*[*OF assms*] **.**
{fix *nL nR ltrv1 ltrv2*
assume ∃ *trn w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w2* ∧
ltrv1 = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
ltrv2 = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*
hence *TwoFuncPred.sameFM isIntV isIntV getActV getActV nL nR ltrv1 ltrv2*
proof(*coinduct rule*: *TwoFuncPred.sameFM.coinduct*[*of* λ*nL nR ltrv1 ltrv2*.
∃ *trn w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w2* ∧
ltrv1 = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
ltrv2 = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧ *lnever isIntO ltr1* ∧

Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧ *lnever isIntO ltr2*])
case (*2 nL nR ltrv1 ltrv2*)
then obtain *trn w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*
where *nL*: *nL* = *w1* **and** *nR*: *nR* = *w2*
and *ltrv1*: *ltrv1* = *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *ltrv2*: *ltrv2* = *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
by *auto*
have *isi3*: ¬ *isIntO s1* **using** *ltr1*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))
have *isi4*: ¬ *isIntO s2* **using** *ltr2*
by (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel llist.pred-inject*(*2*))

show *?case* **proof**(*cases ltr1* = [[]] ∧ *ltr2* = [[]])
case *True* **note** *ltr14* = *True*
hence *ltrv1*: *ltrv1* = [[]] **unfolding** *ltrv1* **by** *simp*
show *?thesis* **using** *ltr14* **unfolding** *ltrv1 ltrv2* **apply**−**apply**(*rule TwoFuncPred.sameFM-selectLNil*) **by** *auto*
next
case *False* **hence** *ltr14*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] **by** *auto*
show *?thesis* **proof**(*cases llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*)

case *True* **note** *ltr14* = *ltr14 True*
hence *ltrv1*: *ltrv1* = [[*sv1*]] **and** *ltrv2*: *ltrv2* = [[*sv2*]] **unfolding** *ltrv1 ltrv2*
by *auto*
have *llength ltr1* = *Suc 0* ∨ *llength ltr2* = *Suc 0*
by (*metis Opt.lcompletedFrom-def Suc-ile-eq True*
lfinite-LNil llength-LNil llist-eq-cong ltr1(*2*)
ltr2(*2*) *nle-le order-le-imp-less-or-eq zero-enat-def zero-order*(*3*))
hence *finalO s1* ∨ *finalO s2*
using *Opt.lcompletedFrom-singl ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) **by** *blast*
hence *fs1*: *finalO s1* ∧ *finalO s2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *auto*

have *fsv12*: *finalV sv1* ∧ *finalV sv2*
using Δ *fs1 r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-final* **by** *blast*
have *isv12*: ¬ *isIntV sv1* ∧ ¬ *isIntV sv2*
using *fsv12 Van.final-not-isInt* **by** *blast*
show *?thesis* **unfolding** *ltrv1 ltrv2* **apply**(*rule TwoFuncPred.sameFM-selectSingl*)

using *isv12* **by** *auto*
next
case *False* **hence** *current*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
by *auto*
show *?thesis* **proof**(*cases trn*)
case *L* **note** *current* = *current L*
show *?thesis*
proof(*cases lnever isSecO ltr1*)
case *True* **note** *current* = *current True*
obtain *trn′ w1′ w2′ s1′ ltr1′ trv1 sv1′ trv2 sv2′ statOO* **where**
ωω: *ltr1* = *s1* $ *ltr1′ validTransO* (*s1*, *s1′*) *Opt.lvalidFromS s1′ ltr1′*
lcompletedFromO s1′ ltr1′ lnever isIntO ltr1′ **and**
ω3: *ω3* Δ *w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′*
statOO
and *trn′*: *trn′* = (*if trv1* = [] *then L else R*)
and *lltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*,
sv1′, *sv2′*, *statOO*))
and *lltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*, *s2*, *ltr2*, *statA*,
sv1′, *sv2′*, *statOO*))
using *lltrv1-lltrv2-lnever-L*[*OF unw* Δ *r ltr1 isi4 current*]
unfolding *ltrv1 ltrv2* **by** *blast*
define *ltrv1′* **where** *ltrv1′*: *ltrv1′* ≡ *lltrv1* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*,
s2, *ltr2*, *statA*, *sv1′*, *sv2′*, *statOO*)
have *lltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1′*
unfolding *lltrv1 ltrv1′* **..**
define *ltrv2′* **where** *ltrv2′*: *ltrv2′* ≡ *lltrv2* (*trn′*, *w1′*, *w2′*, *s1′*, *ltr1′*,
s2, *ltr2*, *statA*, *sv1′*, *sv2′*, *statOO*)
have *lltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2′*
unfolding *lltrv2 ltrv2′* **..**

show *?thesis*
apply(*rule TwoFuncPred.sameFM-selectlappend*)
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w1*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - w2*])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - ltrv2'*])
apply(*intro conjI*)
subgoal unfolding *nL* **.. subgoal unfolding** *nR* **..**
subgoal using *lltrv1* **.**
subgoal using *lltrv2* **.**
subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def* **by** (*simp add*: *Van.A.map-filter*)

subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1'*]) **apply**(*rule exI*[*of - ltr1'*])
apply(*rule exI*[*of - s2*]) **apply**(*rule exI*[*of - ltr2*])
apply(*rule exI*[*of - statA*])
apply(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv1'* **..**
subgoal unfolding *ltrv2'* **..**
subgoal using *ω3* **unfolding** *ω3-def* **by** *simp*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*1*) *snd-conv*)
subgoal by *fact*
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω3* **unfolding** *ω3-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto*
subgoal using *ωω*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal using *ωω* **using** *ltr2*(*1*) **by** *fastforce*
subgoal by *fact*
subgoal by *fact* **. .**
next
case *False* **note** *current* = *current False*
obtain *w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO* **where**
χχ: *χχ s1 ltr1 tr1 s1' s1'' ltr1'* **and**

```
χ3': χ3' Δ w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2
trv2 sv2'' statOO
      and lltrv1: ltrv1 =
    lappend (llist-of trv1) (lltrv1 (R,w1',w2',s1'',s1'' $ ltr1',s2,ltr2,statA,sv1'',sv2'',statOO))

      and lltrv2: ltrv2 =
    lappend (llist-of trv2) (lltrv2 (R,w1',w2',s1'',s1'' $ ltr1',s2,ltr2,statA,sv1'',sv2'',statOO))
      using lltrv1-lltrv2-not-lnever-L[OF unw Δ r ltr1 isi4 current]
      unfolding ltrv1 ltrv2 by blast
        define ltrv1' where ltrv1': ltrv1' ≡ lltrv1 (R,w1',w2',s1'',s1'' $
ltr1',s2,ltr2,statA,sv1'',sv2'',statOO)
      have lltrv1: ltrv1 = lappend (llist-of trv1) ltrv1'
      unfolding lltrv1 ltrv1' ..
        define ltrv2' where ltrv2': ltrv2' ≡ lltrv2 (R,w1',w2',s1'',s1'' $
ltr1',s2,ltr2,statA,sv1'',sv2'',statOO)
      have lltrv2: ltrv2 = lappend (llist-of trv2) ltrv2'
      unfolding lltrv2 ltrv2' ..

      show ?thesis apply(rule TwoFuncPred.sameFM-selectlappend)
      apply(rule exI[of - trv1]) apply(rule exI[of - w1']) apply(rule exI[of -
w1])
      apply(rule exI[of - trv2]) apply(rule exI[of - w2']) apply(rule exI[of -
w2])
      apply(rule exI[of - ltrv1']) apply(rule exI[of - ltrv2'])
      apply(intro conjI)
        subgoal unfolding nL .. subgoal unfolding nR ..
        subgoal using lltrv1 .
        subgoal using lltrv2 .
        subgoal using χ3' unfolding χ3'-def by auto
        subgoal using χ3' unfolding χ3'-def by auto
      subgoal using χ3' unfolding χ3'-def by (simp add: Van.A.map-filter)

        subgoal apply(rule disjI1)
        apply(rule exI[of - R])  apply(rule exI[of - w1']) apply(rule exI[of -
w2'])
        apply(rule exI[of - s1'']) apply(rule exI[of - s1'' $ ltr1'])
        apply(rule exI[of - s2]) apply(rule exI[of - ltr2])
      apply(rule exI[of - statA]) apply(rule exI[of - sv1'']) apply(rule exI[of
- sv2''])
        apply(rule exI[of - statOO])
        apply(intro conjI)
          subgoal ..subgoal ..
          subgoal unfolding ltrv1' ..
          subgoal unfolding ltrv2' ..
          subgoal using χ3' unfolding χ3'-def by simp
          subgoal using χ3' unfolding χ3'-def
       by (metis Simple-Transition-System.reach-validFromS-reach χχ χχ-def

          append-is-Nil-conv last-snoc not-Cons-self2 r(1))
```

subgoal by *fact*
subgoal using $\chi 3'$ **unfolding** $\chi 3'$*-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*)
snoc-eq-iff-butlast)
subgoal using $\chi 3'$ **unfolding** $\chi 3'$*-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*)
snoc-eq-iff-butlast)
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **by** *auto*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def*
using *llist-all-lappend-llist-of ltr1*(*3*) **by** *blast*
subgoal using $\chi\chi$ **unfolding** $\chi\chi$*-def* **using** *ltr2*(*1*) **by** *fastforce*
subgoal by *fact*
subgoal by *fact* **. .**
qed
next
case *R* **note** *current* = *current R*
show *?thesis*
proof(*cases lnever isSecO ltr2*)
case *True* **note** *current* = *current True*
obtain *trn' w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO* **where**
$\omega\omega$: *ltr2* = *s2* $ *ltr2' validTransO* (*s2*, *s2'*) *Opt.lvalidFromS s2' ltr2'*
lcompletedFromO s2' ltr2' lnever isIntO ltr2' **and**
$\omega 4$: $\omega 4$ Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2'*
statOO
and *trn'*: *trn'* = (*if trv2* = [] *then R else L*)
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*,
sv1', *sv2'*, *statOO*))
and *ltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*, *ltr2'*, *statA*,
sv1', *sv2'*, *statOO*))
using *lltrv1-lltrv2-lnever-R*[*OF unw* Δ *r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]
unfolding *ltrv1 ltrv2* **by** *blast*
define *ltrv1'* **where** *ltrv1'*: *ltrv1'* ≡ *lltrv1* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*,
ltr2', *statA*, *sv1'*, *sv2'*, *statOO*)
have *lltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1'*
unfolding *ltrv1 ltrv1'* **..**
define *ltrv2'* **where** *ltrv2'*: *ltrv2'* ≡ *lltrv2* (*trn'*, *w1'*, *w2'*, *s1*, *ltr1*, *s2'*,
ltr2', *statA*, *sv1'*, *sv2'*, *statOO*)
have *lltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2'*
unfolding *ltrv2 ltrv2'* **..**
show *?thesis*
apply(*rule TwoFuncPred.sameFM-selectlappend*)
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of -*
w1])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of -*
w2])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - ltrv2'*])

apply(*intro conjI*)
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal using *lltrv1* **.**
subgoal using *lltrv2* **.**
subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal using *ω4* **unfolding** *ω4-def* **by** (*simp add*: *Van.A.map-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - trn'*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - s2'*]) **apply**(*rule exI*[*of - ltr2'*])
apply(*rule exI*[*of - statA*]) **apply**(*rule exI*[*of - sv1'*]) **apply**(*rule exI*[*of - sv2'*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv1'* **..**
subgoal unfolding *ltrv2'* **..**
subgoal using *ω4* **unfolding** *ω4-def* **by** *simp*
subgoal by *fact*
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Opt.reach.Step ωω*(*2*) *fst-conv r*(*2*) *snd-conv*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *ω4* **unfolding** *ω4-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal by *fact*
subgoal using *ωω* **by** *auto*
subgoal using *ωω* **by** *auto* **. .**
next
case *False* **note** *current* = *current False*
obtain *w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO* **where**
χχ: *χχ s2 ltr2 tr2 s2' s2'' ltr2'* **and**
χ4': *χ4' Δ w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
and *ltrv1*: *ltrv1* =
lappend (*llist-of trv1*) (*lltrv1* (*L*, *w1'*, *w2'*, *s1*, *ltr1*, *s2''*, *s2''* $ *ltr2'*, *statA*, *sv1''*, *sv2''*, *statOO*))
and *ltrv2*: *ltrv2* =
lappend (*llist-of trv2*) (*lltrv2* (*L*, *w1'*, *w2'*, *s1*, *ltr1*, *s2''*, *s2''* $ *ltr2'*, *statA*, *sv1''*, *sv2''*, *statOO*))
using *lltrv1-lltrv2-not-lnever-R*[*OF unw Δ r ltr2*(*1*,*2*) *isi3 ltr2*(*3*) *current*]

unfolding *ltrv1 ltrv2* **by** *blast*
define *ltrv1'* **where** *ltrv1'*: *ltrv1'* ≡ *lltrv1* (*L*, *w1'*, *w2'*, *s1*, *ltr1*, *s2''*, *s2''* $ *ltr2'*, *statA*, *sv1''*, *sv2''*, *statOO*)
have *lltrv1*: *ltrv1* = *lappend* (*llist-of trv1*) *ltrv1'*
unfolding *ltrv1 ltrv1'* **..**
define *ltrv2'* **where** *ltrv2'*: *ltrv2'* ≡ *lltrv2* (*L*, *w1'*, *w2'*, *s1*, *ltr1*, *s2''*, *s2''* $ *ltr2'*, *statA*, *sv1''*, *sv2''*, *statOO*)
have *lltrv2*: *ltrv2* = *lappend* (*llist-of trv2*) *ltrv2'*
unfolding *ltrv2 ltrv2'* **..**

show *?thesis*
apply(*rule TwoFuncPred.sameFM-selectlappend*)
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w1*])
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - w2*])
apply(*rule exI*[*of - ltrv1'*]) **apply**(*rule exI*[*of - ltrv2'*])
apply(*intro conjI*)
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal using *lltrv1* **.**
subgoal using *lltrv2* **.**
subgoal using *χ4'* **unfolding** *χ4'-def* **by** *simp*
subgoal using *χ4'* **unfolding** *χ4'-def* **by** *simp*
subgoal using *χ4'* **unfolding** *χ4'-def* **by** (*simp add*: *Van.A.map-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - L*]) **apply**(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1*]) **apply**(*rule exI*[*of - ltr1*])
apply(*rule exI*[*of - s2''*]) **apply**(*rule exI*[*of - s2''* $ *ltr2'*])
apply(*rule exI*[*of - statA*])
apply(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - sv2''*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrv1'* **..**
subgoal unfolding *ltrv2'* **..**
subgoal using *χ4'* **unfolding** *χ4'-def* **by** *simp*
subgoal by *fact*
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach χχ χχ-def*

append-is-Nil-conv last-snoc not-Cons-self2 r(*2*))
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*3*) *snoc-eq-iff-butlast*)
subgoal using *χ4'* **unfolding** *χ4'-def*
by (*metis Simple-Transition-System.reach-validFromS-reach r*(*4*) *snoc-eq-iff-butlast*)
subgoal by *fact*

```
            subgoal by fact
            subgoal by fact
            subgoal using χχ unfolding χχ-def by auto
            subgoal using χχ unfolding χχ-def by auto
            subgoal using χχ unfolding χχ-def
            using llist-all-lappend-llist-of ltr2(3) by blast . .
         qed
        qed
       qed
      qed
     qed
    }
    thus ?thesis unfolding Van.lA[OF cltrv1] Van.lA[OF cltrv2]
    apply− apply(rule TwoFuncPred.sameFM-lmap-lfilter)
    using assms by blast
qed
```

```
fun isN :: ('stateO,'stateV) tuple34 ⇒ bool
where
isN (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) ⟷ ltr1 = [[]] ∨ ltr2 = [[]]

fun H-T ::
('stateO,'stateV)tuple34 ⇒
 ('stateO,'stateV)tuple12 ×
 ('stateO,'stateV)tuple34
where
H-T (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
 (let (tr1,s1',s1'',ltr1',tr2,s2',s2'',ltr2') =
      (SOME k. case k of (tr1,s1',s1'',ltr1',tr2,s2',s2'',ltr2') ⇒
         φφ s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2')
  in let (w1',w2',trv1,sv1'',trv2,sv2'',statAA,statOO) =
           (SOME k'. case k' of (w1',w2',trv1,sv1'',trv2,sv2'',statAA,statOO) ⇒
             φ' Δ w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO
sv1 trv1 sv1'' sv2 trv2 sv2'' statOO)
  in ((trv1,sv1'',trv2,sv2'',statAA,statOO),
      (w1',w2',s1'',s1'' $ ltr1',s2'',s2'' $ ltr2',statAA,sv1'',sv2'',statOO))
 )

declare H-T.simps[simp del]

definition H ≡ fst o H-T
definition T ≡ snd o H-T

fun Econd where Econd (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) = (lnever
```

isIntO ltr1)

fun *E* **where** *E* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = *f* (*L*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*

definition *F* :: ('*stateO*,'*stateV*)*tuple34* ⇒ ('*stateO*,'*stateV*)*tuple12 llist*
where *F* ≡ *ccorec-llist isN H Econd E T*

lemma *F-LNil*:
ltr1 = [[]] ∨ *ltr2* = [[]] ⟹ *F* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[]]
unfolding *F-def* **apply**(*subst llist-ccorec*(*1*)) **by** *auto*

lemma *F-lnever*:
assumes *ltr1* ≠ [[]] *ltr2* ≠ [[]] *lnever isIntO ltr1*
shows *F* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = *f* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
using *assms* **unfolding** *F-def* **apply**(*subst llist-ccorec*(*2*))
  subgoal unfolding *E.simps lnull-def* **apply**(*rule f-not-LNil*) **by** *auto*
  subgoal using *assms* **by** *auto*
  subgoal unfolding *Econd.simps* **by** *auto* **.**

lemma *F-not-lnever*:
assumes *ltr1* ≠ [[]] *ltr2* ≠ [[]] ¬ *lnever isIntO ltr1*
shows *F* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
  LCons (*H* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) (*F* (*T* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)))
using *assms* **unfolding** *F-def* **apply**(*subst llist-ccorec*(*2*))
  subgoal unfolding *E.simps lnull-def* **apply**(*rule f-not-LNil*) **by** *auto*
  subgoal using *assms* **by** *auto*
  subgoal unfolding *Econd.simps* **by** *auto* **.**

definition *ltrv1* :: ('*stateO*,'*stateV*)*tuple34* ⇒ '*stateV llist* **where**
ltrv1 tp = *lconcat* (*lmap* (λ(*trv1*,*sv1''*,*trv2*,*sv2''*,*statAA*,*statOO*). *llist-of trv1*) (*F tp*))

definition *firstHolds1* :: ('*stateO*,'*stateV*)*tuple34* ⇒ *nat* **where**
firstHolds1 tp = *firstNC* (*lmap* (λ(*trv1*,*sv1''*,*trv2*,*sv2''*,*statAA*,*statOO*). *trv1*) (*F tp*))

definition *ltrv2* :: ('*stateO*,'*stateV*)*tuple34* ⇒ '*stateV llist* **where**
ltrv2 tp = *lconcat* (*lmap* (λ(*trv1*,*sv1''*,*trv2*,*sv2''*,*statAA*,*statOO*). *llist-of trv2*) (*F tp*))

definition *firstHolds2* :: ('*stateO*,'*stateV*)*tuple34* ⇒ *nat* **where**
firstHolds2 tp = *firstNC* (*lmap* (λ(*trv1*,*sv1''*,*trv2*,*sv2''*,*statAA*,*statOO*). *trv2*) (*F tp*))

lemma *ltrv1-ne-imp*:
assumes *ltrv1 tp* ≠ [[]]
shows ∃ *trv1 sv1'' trv2 sv2'' statAA statOO*. (*trv1*,*sv1''*,*trv2*,*sv2''*,*statAA*,*statOO*) ∈ *lset* (*F tp*) ∧
trv1 ≠ []
using *assms* **unfolding** *ltrv1-def* **unfolding** *lconcat-eq-LNil-iff* **by** *force*

lemma *ltrv2-ne-imp*:
assumes *ltrv2 tp* ≠ [[]]
shows ∃ *trv1 sv1'' trv2 sv2'' statAA statOO*. (*trv1*,*sv1''*,*trv2*,*sv2''*,*statAA*,*statOO*) ∈ *lset* (*F tp*) ∧
trv2 ≠ []
using *assms* **unfolding** *ltrv2-def* **unfolding** *lconcat-eq-LNil-iff* **by** *force*

lemma *ltrv1-LNil*[*simp*]:
ltr1 = [[]] ∨ *ltr2* = [[]] ⟹ *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[]]
unfolding *ltrv1-def F-LNil* **by** *simp*
lemma *ltrv2-LNil*[*simp*]:
ltr1 = [[]] ∨ *ltr2* = [[]] ⟹ *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[]]
unfolding *ltrv2-def F-LNil* **by** *simp*

lemma *ltrv1-lnever*:
assumes *ltr1* ≠ [[]] *ltr2* ≠ [[]] *lnever isIntO ltr1*
shows *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = *lltrv1* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
unfolding *ltrv1-def F-lnever*[*OF assms*] *lltrv1-def* **..**

lemma *ltrv2-lnever*:
assumes *ltr1* ≠ [[]] *ltr2* ≠ [[]] *lnever isIntO ltr1*
shows *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = *lltrv2* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
unfolding *ltrv2-def F-lnever*[*OF assms*] *lltrv2-def* **..**

lemma *H-T-not-lnever*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*

and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l*: ¬ *lnever isIntO ltr1 Opt.lA ltr1* = *Opt.lA ltr2*
shows ∃ *w1′ w2′ tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′ trv1 sv1″ trv2 sv2″ statAA statOO*.
φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′ ∧
φ′ Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1″ s2 tr2 s2′ s2″ statAA statO sv1 trv1 sv1″ sv2 trv2 sv2″ statOO* ∧
H-T (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
((*trv1*,*sv1″*,*trv2*,*sv2″*,*statAA*,*statOO*),
(*w1′*,*w2′*, *s1″*,*s1″* $ *ltr1′*,*s2″*,*s2″* $ *ltr2′*,*statAA*,*sv1″*,*sv2″*,*statOO*))
proof−
obtain *tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′*
where *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′*
using *isIntO-φφ*[*OF ltr1 ltr2 l*(*2*,*1*)]
by *auto*

define *tp* **where**
tp = (*SOME k*. *case k of* (*tr1*,*s1′*,*s1″*,*ltr1′*,*tr2*,*s2′*,*s2″*,*ltr2′*) ⇒
φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′)

have *0*: *case tp of* (*tr1*,*s1′*,*s1″*,*ltr1′*,*tr2*,*s2′*,*s2″*,*ltr2′*) ⇒
φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′
using *φφ* **unfolding** *tp-def* **apply**− **apply**(*rule someI-ex*)
apply(*rule exI*[*of* - (*tr1*,*s1′*,*s1″*,*ltr1′*,*tr2*,*s2′*,*s2″*,*ltr2′*)]) **by** *auto*

obtain *tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′* **where**
tp: *tp* = (*tr1*,*s1′*,*s1″*,*ltr1′*,*tr2*,*s2′*,*s2″*,*ltr2′*) **by**(*cases tp*, *auto*)

have *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′*
using *0* **unfolding** *tp* **by** *auto*

obtain *w1′ w2′ trv1 sv1″ trv2 sv2″ statAA statOO*
where *φ′*: *φ′* Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1″ s2 tr2 s2′ s2″ statAA statO sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
using *unwindCond-ex-φ′*[*OF unw* Δ *r stat*, *of tr1 s1′ s1″ tr2 s2′ s2″*]
using *φφ* **unfolding** *φφ-def* **by** *auto*

define *tp′* **where**
tp′ = (*SOME k′*. *case k′ of* (*w1′*,*w2′*,*trv1*,*sv1″*,*trv2*,*sv2″*,*statAA*,*statOO*) ⇒
φ′ Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1″ s2 tr2 s2′ s2″ statAA statO sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*)

have *1*: *case tp′ of* (*w1′*,*w2′*,*trv1*,*sv1″*,*trv2*,*sv2″*,*statAA*,*statOO*) ⇒
φ′ Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1″ s2 tr2 s2′ s2″ statAA statO sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
using *φ′* **unfolding** *tp′-def* **apply**− **apply**(*rule someI-ex*)
apply(*rule exI*[*of* - (*w1′*,*w2′*,*trv1*,*sv1″*,*trv2*,*sv2″*,*statAA*,*statOO*)]) **by** *auto*

obtain *w1′ w2′ trv1 sv1′′ trv2 sv2′′ statAA statOO* **where**
tp′: *tp′* = (*w1′*,*w2′*,*trv1*,*sv1′′*,*trv2*,*sv2′′*,*statAA*,*statOO*) **by**(*cases tp′*, *auto*)

have *φ′*: *φ′* Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1′′ s2 tr2 s2′ s2′′ statAA statO sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
using *1* **unfolding** *tp′* **by** *auto*

show *?thesis*
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - tr1*]) **apply**(*rule exI*[*of - s1′*]) **apply**(*rule exI*[*of - s1′′*]) **apply**(*rule exI*[*of - ltr1′*])
apply(*rule exI*[*of - tr2*]) **apply**(*rule exI*[*of - s2′*]) **apply**(*rule exI*[*of - s2′′*]) **apply**(*rule exI*[*of - ltr2′*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1′′*]) **apply**(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2′′*])
apply(*rule exI*[*of - statAA*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal using *φφ* **.**
subgoal using *φ′* **.**
subgoal unfolding *H-T.simps*
unfolding *tp-def*[*symmetric*] *tp* **apply** *simp*
unfolding *tp′-def*[*symmetric*] *tp′* **by** *simp* **.**
qed

lemma *ltrv1-ltrv2-not-lnever*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l*: ¬ *lnever isIntO ltr1 Opt.lA ltr1* = *Opt.lA ltr2*
shows ∃ *w1′ w2′ tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statAA statOO*.
φφ s1 ltr1 s2 ltr2 tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′ ∧
φ′ Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1′′ s2 tr2 s2′ s2′′ statAA statO sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO* ∧
ltrv1 (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv1*) (*ltrv1* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*)) ∧
ltrv2 (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv2*) (*ltrv2* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*))
proof−
have *ltr1NE*: *ltr1* ≠ [[]] **using** *l*(*1*) **by** *auto*
hence *ltr2NE*: *ltr2* ≠ [[]] **using** *l*(*2*)
using *Opt.lcompletedFrom-def ltr2*(*2*) **by** *blast*
show *?thesis*
using *H-T-not-lnever*[*OF assms*] **apply**(*elim exE*)

subgoal for *w1′ w2′ tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statAA statOO*
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - tr1*]) **apply**(*rule exI*[*of - s1′*]) **apply**(*rule exI*[*of - s1′′*]) **apply**(*rule exI*[*of - ltr1′*])
apply(*rule exI*[*of - tr2*]) **apply**(*rule exI*[*of - s2′*]) **apply**(*rule exI*[*of - s2′′*]) **apply**(*rule exI*[*of - ltr2′*])
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - sv1′′*]) **apply**(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - sv2′′*])
apply(*rule exI*[*of - statAA*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal by *simp*
subgoal by *simp*
subgoal unfolding *ltrv1-def* **apply**(*subst F-not-lnever*[*OF ltr1NE ltr2NE l*(*1*)])
unfolding *H-def T-def* **by** *simp*
subgoal unfolding *ltrv2-def* **apply**(*subst F-not-lnever*[*OF ltr1NE ltr2NE l*(*1*)])
unfolding *H-def T-def* **by** *simp* **. .**
qed

lemma *lvalidFromS-ltrv1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *ltr14*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lvalidFromS sv1* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
proof−
{**fix** *n1 sv1 ltrr1*
assume ∃ *w1 w2 s1 ltr1 s2 ltr2 statA sv2 statO*.
ltrr1 = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
n1 = *w1* ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧
Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧
Opt.lA ltr1 = *Opt.lA ltr2*
hence *Van.llvalidFromS n1 sv1 ltrr1*
proof(*coinduct rule*: *Van.llvalidFromS.coinduct*[*of* λ*n1 sv1 ltrr1*.
∃ *w1 w2 s1 ltr1 s2 ltr2 statA sv2 statO*.
ltrr1 = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
n1 = *w1* ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧

Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧
Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧
Opt.lA ltr1 = *Opt.lA ltr2*])
case (*llvalidFromS n1 sv1 ltrr1*)
then obtain *w1 w2 s1 ltr1 s2 ltr2 statA sv2 statO*
where *ltrr1*: *ltrr1* = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *n1*: *n1* = *w1*
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
by *auto*

have *current*: *ltr1* ≠ [[]] *ltr2* ≠ [[]]
using *ltr1*(*2*) *ltr2*(*2*) **unfolding** *Opt.lcompletedFrom-def* **by** *auto*

show *?case* **proof**(*cases lnever isIntO ltr1*)
case *True* **note** *current* = *current True*
hence *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34* *ltr2*(*2*))
note *ln34* = *True this*
have *ltrr1*: *ltrr1* = *lltrv1* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
unfolding *ltrr1 ltrv1-lnever*[*OF current*] **by** *simp*
show *?thesis* **apply**(*rule Van.llvalidFromS-selectlvalidFromS*)
unfolding *ltrr1* **apply** *simp*
apply(*rule lvalidFromS-lltrv1*)
using *ln34* Δ *llvalidFromS ln34*(*2*) *ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) *r*(*1*) *r*(*2*) *r*(*4*) *unw* **by** *auto*
next
case *False* **note** *ln3* = *False*
hence *ln4*: ¬ *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34* *ltr2*(*2*))

have *ltr1* ≠ [[*s1*]] **using** *ln3 ltr1*
using *Opt.final-not-isInt* **by** *auto*
hence *llength ltr1* > *Suc 0*
by (*metis Opt.lcompletedFrom-singl Suc-ile-eq current*(*1*) *enat-0-iff*(*1*)
linorder-not-less llength-LNil llist-eq-cong ltr1(*1*) *ltr1*(*2*) *nle-le not-less-zero*)
hence ¬ *finalO s1*
by (*metis Opt.final-def Opt.lvalidFromS-Cons-iff current*(*1*) *eSuc-enat enat-0*

linorder-neq-iff llength-LCons llength-LNil llist.exhaust-sel ltr1(*1*))
hence *nf12*: ¬ *finalV sv1* ∧ ¬ *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *force*

obtain *w1' w2' tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statAA statOO*
where *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'*
and *φ'*: *φ'* Δ *w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
and *ltrr1*: *ltrr1* =
lappend (*llist-of trv1*) (*ltrv1* (*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2''*,*s2''* $ *ltr2'*,*statAA*,*sv1''*,*sv2''*,*statOO*))

using *ltrv1-ltrv2-not-lnever*[*OF unw* Δ *r stat ltr1 ltr2 ln3 A34*]
unfolding *ltrr1* **by** *blast*
define *ltrr1'* **where** *ltrr1'*: *ltrr1'* = *ltrv1* (*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2''*,*s2''* $ *ltr2'*,*statAA*,*sv1''*,*sv2''*,*statOO*)
have *ltrr1*: *ltrr1* = *lappend* (*llist-of trv1*) *ltrr1'*
unfolding *ltrr1 ltrr1'* **..**
have *ne*: *trv1* ≠ [] ∨ (*trv1* = [] ∧ *w1'* < *w1*)
using *φ'* **unfolding** *φ'-def ltrr1* **by** *simp*

show *?thesis* **using** *ne* **proof**(*elim disjE conjE*)
assume *trv1*: *trv1* ≠ []
show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of - sv1*]) **apply**(*rule exI*[*of - trv1*])
apply(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - w1'*])
apply(*rule exI*[*of - ltrr1'*]) **apply**(*rule exI*[*of - w1*])
apply(*intro conjI*)
subgoal unfolding *n1* **.. subgoal ..**
subgoal unfolding *ltrr1* **..**
subgoal using *φ'* **unfolding** *φ'-def*
by (*metis Van.validS-append1 Van.validFromS-def append-is-Nil-conv hd-append2*)
subgoal by *fact*
subgoal using *φ'* **unfolding** *φ'-def*
by (*metis Simple-Transition-System.validFromS-def Van.validS-validTrans append-is-Nil-conv list.sel*(*1*) *not-Cons-self2 trv1*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1''*]) **apply**(*rule exI*[*of - s1''* $ *ltr1'*])
apply(*rule exI*[*of - s2''*]) **apply**(*rule exI*[*of - s2''* $ *ltr2'*])
apply(*rule exI*[*of - statAA*]) **apply**(*rule exI*[*of - sv2''*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal unfolding *ltrr1'* **..**
subgoal ..
subgoal using *φ'* **unfolding** *φ'-def* **by** *auto*
subgoal using *r*(*1*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def*

by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r(3)* φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc trv1*)
subgoal using *r(4)* φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using φ' **unfolding** φ'*-def* **by** *auto*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp* **. .**
next
assume *trv1[simp]*: *trv1* = [] **and** *MM'*: *w1'* < *w1*
hence *sv1''[simp]*: *sv1''* = *sv1* **using** φ' **unfolding** φ'*-def* **by** *simp*
show *?thesis*
apply(*rule Van.llvalidFromS-selectDelay*)
apply(*rule exI[of - w1']*) **apply**(*rule exI[of - w1]*)
apply(*rule exI[of - sv1'']*) **apply**(*rule exI[of - ltrr1']*)
apply(*intro conjI*)
subgoal unfolding *n1* **.. subgoal by** *simp*
subgoal unfolding *ltrr1* **by** *simp* **subgoal by** *fact*
subgoal apply(*rule disjI1*)
apply(*rule exI[of - w1']*) **apply**(*rule exI[of - w2']*)
apply(*rule exI[of - s1'']*) **apply**(*rule exI[of - s1'' $ ltr1']*)
apply(*rule exI[of - s2'']*) **apply**(*rule exI[of - s2'' $ ltr2']*)
apply(*rule exI[of - statAA]*) **apply**(*rule exI[of - sv2'']*) **apply**(*rule exI[of - statOO]*)
apply(*intro conjI*)
subgoal unfolding *ltrr1'* **..**
subgoal ..
subgoal using φ' **unfolding** φ'*-def* **by** *auto*
subgoal using *r(1)* $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r(2)* $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal unfolding *sv1''* **by** *fact*
subgoal using *r(4)* φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using φ' **unfolding** φ'*-def* **by** *auto*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp* **. .**
qed

qed
qed
}
thus *?thesis* **apply**−**apply**(*rule Van.llvalidFromS-imp-lvalidFromS*)
using *assms* **by** *blast*
qed

lemma *lvalidFromS-ltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *ltr14*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lvalidFromS sv2* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
proof−
{**fix** *n2 sv2 ltrr2*
assume ∃ *w1 w2 s1 ltr1 s2 ltr2 statA sv1 statO*.
ltrr2 = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
n2 = *w2* ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧
Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧
Opt.lA ltr1 = *Opt.lA ltr2*
hence *Van.llvalidFromS n2 sv2 ltrr2*
proof(*coinduct rule*: *Van.llvalidFromS.coinduct*[*of* λ*n2 sv2 ltrr2*.
∃ *w1 w2 s1 ltr1 s2 ltr2 statA sv1 statO*.
ltrr2 = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
n2 = *w2* ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧
Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧
Opt.lA ltr1 = *Opt.lA ltr2*])
case (*llvalidFromS n2 sv2 ltrr2*)
then obtain *w1 w2 s1 ltr1 s2 ltr2 statA sv1 statO*
where *ltrr2*: *ltrr2* = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *n2*: *n2* = *w2*
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
by *auto*

have *current*: *ltr1* ≠ [[]] *ltr2* ≠ [[]]
using *ltr1*(*2*) *ltr2*(*2*) **unfolding** *Opt.lcompletedFrom-def* **by** *auto*

show *?case* **proof**(*cases lnever isIntO ltr1*)
case *True* **note** *current* = *current True*
hence *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34* *ltr2*(*2*))
note *ln34* = *True this*
have *ltrr2*: *ltrr2* = *lltrv2* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
unfolding *ltrr2 ltrv2-lnever*[*OF current*] **by** *simp*
show *?thesis* **apply**(*rule Van.llvalidFromS-selectlvalidFromS*)
unfolding *ltrr2* **apply** *simp*
apply(*rule lvalidFromS-lltrv2*)
using *ln34* Δ *llvalidFromS ln34*(*2*) *ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) *r*(*1*) *r*(*2*) *r*(*3*) *unw* **by** *auto*
next
case *False* **note** *ln3* = *False*
hence *ln4*: ¬ *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34* *ltr2*(*2*))

have *ltr2* ≠ [[*s2*]] **using** *ln4 ltr2*
using *Opt.final-not-isInt* **by** *auto*
hence *llength ltr2* > *Suc 0*
by (*metis Opt.lcompletedFrom-singl Suc-ile-eq current*(*2*) *enat-0-iff*(*2*)
linorder-not-less llength-LNil llist-eq-cong ltr2(*1*) *ltr2*(*2*) *nle-le not-iless0*)
hence ¬ *finalO s2*
by (*metis Opt.final-def Opt.lvalidFromS-Cons-iff current*(*2*) *eSuc-enat enat-0*

linorder-neq-iff llength-LCons llength-LNil llist.exhaust-sel ltr2(*1*))
hence *nf12*: ¬ *finalV sv1* ∧ ¬ *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *force*

obtain *w1′ w2′ tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statAA statOO*
where *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′*
and *φ′*: *φ′* Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1′′ s2 tr2 s2′ s2′′ statAA statO sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
and *ltrr2*: *ltrr2* =
lappend (*llist-of trv2*) (*ltrv2* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*))

using *ltrv1-ltrv2-not-lnever*[*OF unw* Δ *r stat ltr1 ltr2 ln3 A34*]
unfolding *ltrr2* **by** *blast*
define *ltrr2′* **where** *ltrr2′*: *ltrr2′* = *ltrv2* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*)
have *ltrr2*: *ltrr2* = *lappend* (*llist-of trv2*) *ltrr2′*

unfolding *ltrr2 ltrr2'* **..**
have *ne*: *trv2* ≠ [] ∨ (*trv2* = [] ∧ *w2'* < *w2*)
using φ' **unfolding** *φ'-def ltrr2* **by** *simp*

show *?thesis* **using** *ne* **proof**(*elim disjE conjE*)
assume *trv2*: *trv2* ≠ []
show *?thesis*
apply(*rule Van.llvalidFromS-selectlappend*)
apply(*rule exI*[*of - sv2*]) **apply**(*rule exI*[*of - trv2*])
apply(*rule exI*[*of - sv2''*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - ltrr2'*]) **apply**(*rule exI*[*of - w2*])
apply(*intro conjI*)
subgoal unfolding *n2* **.. subgoal ..**
subgoal unfolding *ltrr2* **..**
subgoal using φ' **unfolding** *φ'-def*
by (*metis Van.validS-append1 Van.validFromS-def append-is-Nil-conv hd-append2*)
subgoal by *fact*
subgoal using φ' **unfolding** *φ'-def*
by (*metis Simple-Transition-System.validFromS-def Van.validS-validTrans append-is-Nil-conv list.distinct*(*1*) *list.sel*(*1*) *trv2*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1''*]) **apply**(*rule exI*[*of - s1'' $ ltr1'*])
apply(*rule exI*[*of - s2''*]) **apply**(*rule exI*[*of - s2'' $ ltr2'*])
apply(*rule exI*[*of - statAA*]) **apply**(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal unfolding *ltrr2'* **..**
subgoal ..
subgoal using φ' **unfolding** *φ'-def* **by** *auto*
subgoal using *r*(*1*) φφ **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) φφ **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) φ' **unfolding** *φ'-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) φ' **unfolding** *φ'-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using φ' **unfolding** *φ'-def* **by** *auto*
subgoal using φφ **unfolding** *φφ-def* **by** *simp*
subgoal using φφ **unfolding** *φφ-def* **by** *simp*
subgoal using φφ **unfolding** *φφ-def* **by** *simp*
subgoal using φφ **unfolding** *φφ-def* **by** *simp*
subgoal using φφ **unfolding** *φφ-def* **by** *simp* **. .**
next
assume *trv2*[*simp*]: *trv2* = [] **and** *MM'*: *w2'* < *w2*

hence *sv2''*[*simp*]: *sv2''* = *sv2* **using** φ' **unfolding** *φ'-def* **by** *simp*
show *?thesis*
apply(*rule Van.llvalidFromS-selectDelay*)
apply(*rule exI*[*of - w2'*]) **apply**(*rule exI*[*of - w2*])
apply(*rule exI*[*of - sv2''*]) **apply**(*rule exI*[*of - ltrr2'*])
apply(*intro conjI*)
subgoal unfolding *n2* **..** **subgoal by** *simp*
subgoal unfolding *ltrr2* **by** *simp* **subgoal by** *fact*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1''*]) **apply**(*rule exI*[*of - s1'' $ ltr1'*])
apply(*rule exI*[*of - s2''*]) **apply**(*rule exI*[*of - s2'' $ ltr2'*])
apply(*rule exI*[*of - statAA*]) **apply**(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal unfolding *ltrr2'* **..**
subgoal ..
subgoal using φ' **unfolding** *φ'-def* **by** *auto*
subgoal using *r(1)* φφ **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r(2)* φφ **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r(3)* φ' **unfolding** *φ'-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal unfolding *sv2''* **by** *fact*
subgoal using φ' **unfolding** *φ'-def* **by** *auto*
subgoal using φφ **unfolding** *φφ-def* **by** *simp*
subgoal using φφ **unfolding** *φφ-def* **by** *simp*
subgoal using φφ **unfolding** *φφ-def* **by** *simp*
subgoal using φφ **unfolding** *φφ-def* **by** *simp*
subgoal using φφ **unfolding** *φφ-def* **by** *simp* **. .**
qed
qed
qed
}
thus *?thesis* **apply**−**apply**(*rule Van.llvalidFromS-imp-lvalidFromS*)
using *assms* **by** *blast*
qed

lemma *lcompletedFrom-ltrv1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*

and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lcompletedFrom sv1* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
proof−
{**fix** *ltrr1* **assume** *ltrr1*: *ltrr1* = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *lfin*: *lfinite ltrr1*
hence *list-of ltrr1* ≠ [] ∧ *finalV* (*last* (*list-of ltrr1*))
using *assms*(*2*−) **proof**(*induct length* (*list-of ltrr1*) *w1*
arbitrary: *w2 ltrr1 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*
rule: *less2-induct′*)
case (*less w1 ltrr1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*)
hence *ltrr1*: *ltrr1* = *ltrv1* (*w1*,*w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
and *lfin*: *lfinite ltrr1*
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 lcompletedFromO s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 lcompletedFromO s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
by *auto*

have *current*: *ltr1* ≠ [[]] *ltr2* ≠ [[]]
using *ltr1*(*2*) *ltr2*(*2*) **unfolding** *Opt.lcompletedFrom-def* **by** *auto*

show *?case* **proof**(*cases lnever isIntO ltr1*)
case *True* **note** *ln3* = *True* **note** *current* = *current True*
hence *ln4*: *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34 ltr2*(*2*))
note *ln34* = *True this*
have *ltrr1*: *ltrr1* = *lltrv1* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
unfolding *ltrr1 ltrv1-lnever*[*OF current*] **by** *simp*
show *?thesis*
using *lcompletedFrom-lltrv1*[*OF unw* Δ *r ltr1 ln3 ltr2 ln4*, *of L*]
using *lfin*[*unfolded ltrr1*]
unfolding *Van.lcompletedFrom-def ltrr1*[*symmetric*]
using *llist-of-list-of* **by** *fastforce*
next
case *False* **note** *ln3* = *False*
hence *ln4*: ¬ *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34 ltr2*(*2*))

have *ltr1* ≠ [[*s1*]] **using** *ln3 ltr1*
using *Opt.final-not-isInt* **by** *auto*
hence *llength ltr1* > *Suc 0*
by (*metis Opt.lcompletedFrom-singl Suc-ile-eq current*(*1*) *enat-0-iff*(*1*) *linorder-not-less llength-LNil llist-eq-cong ltr1*(*1*) *ltr1*(*2*) *nle-le not-less-zero*)

hence ¬ *finalO s1*
by (*metis Opt.final-def Opt.lvalidFromS-Cons-iff current*(*1*) *eSuc-enat enat-0*

linorder-neq-iff llength-LCons llength-LNil llist.exhaust-sel ltr1(*1*))
hence *nf12*: ¬ *finalV sv1* ∧ ¬ *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *force*

obtain *w1′ w2′ tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statAA statOO*
where *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′*
and *φ′*: *φ′* Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1′′ s2 tr2 s2′ s2′′ statAA statO sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
and *ltrr1*: *ltrr1* =
lappend (*llist-of trv1*) (*ltrv1* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*))

using *ltrv1-ltrv2-not-lnever*[*OF unw* Δ *r stat ltr1 ltr2 ln3 A34*]
unfolding *ltrr1* **by** *blast*
define *ltrr1′* **where** *ltrr1′*: *ltrr1′* = *ltrv1* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*)
have *ltrr1*: *ltrr1* = *lappend* (*llist-of trv1*) *ltrr1′*
unfolding *ltrr1 ltrr1′* **..**
have *ne*: *trv1* ≠ [] ∨ (*trv1* = [] ∧ *w1′* < *w1*)
using *φ′* **unfolding** *φ′-def ltrr1* **by** *simp*

have *lfin′*: *lfinite ltrr1′*
using *lfin ne* **unfolding** *ltrr1* **by** *simp*
have *len*: *length* (*list-of ltrr1′*) < *length* (*list-of ltrr1*) ∨
length (*list-of ltrr1′*) = *length* (*list-of ltrr1*) ∧ *w1′* < *w1*
using *ne lfin lfin′* **by** (*simp add*: *list-of-lappend ltrr1*)

have *0*: *list-of ltrr1′* ≠ [] ∧ *finalV* (*last* (*list-of ltrr1′*))
using *len* **proof**(*elim disjE conjE*)
assume *len*: *length* (*list-of ltrr1′*) < *length* (*list-of ltrr1*)
show *?thesis*
apply(*rule less*(*1*)[*OF - ltrr1′*])
subgoal by *fact* **subgoal by** *fact*
subgoal using *φ′* **unfolding** *φ′-def* **by** *simp*
subgoal using *r*(*1*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) *φ′* **unfolding** *φ′-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) *φ′* **unfolding** *φ′-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *φ′* **unfolding** *φ′-def* **by** *auto*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*

subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp* **.**
next
assume *len*: *length* (*list-of ltrr1'*) = *length* (*list-of ltrr1*) *w1'* < *w1*
show *?thesis*
apply(*rule less*(*2*)[*OF - - ltrr1'*])
subgoal by *fact* **subgoal unfolding** *len* **..**
subgoal by *fact*
subgoal using *φ'* **unfolding** *φ'-def* **by** *simp*
subgoal using *r*(*1*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) *φ'* **unfolding** *φ'-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) *φ'* **unfolding** *φ'-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *φ'* **unfolding** *φ'-def* **by** *auto*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp* **.**
qed
show *?thesis* **unfolding** *ltrr1* **using** *0*
by (*simp add*: *lfin' list-of-lappend*)
qed
qed
}
thus *?thesis* **unfolding** *Van.lcompletedFrom-def* **by** *auto*
qed

lemma *lcompletedFrom-ltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lcompletedFrom sv2* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
proof−
{**fix** *ltrr2* **assume** *ltrr2*: *ltrr2* = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *lfin*: *lfinite ltrr2*
hence *list-of ltrr2* ≠ [] ∧ *finalV* (*last* (*list-of ltrr2*))

using *assms*(*2*−) **proof**(*induct length* (*list-of ltrr2*) *w2*
arbitrary: *w1 ltrr2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*
rule: *less2-induct'*)
case (*less w2 ltrr2 w1 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*)
hence *ltrr2*: *ltrr2* = *ltrv2* (*w1*,*w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
and *lfin*: *lfinite ltrr2*
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 lcompletedFromO s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 lcompletedFromO s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
by *auto*

have *current*: *ltr1* ≠ [[]] *ltr2* ≠ [[]]
using *ltr1*(*2*) *ltr2*(*2*) **unfolding** *Opt.lcompletedFrom-def* **by** *auto*

show *?case* **proof**(*cases lnever isIntO ltr1*)
case *True* **note** *ln3* = *True* **note** *current* = *current True*
hence *ln4*: *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34* *ltr2*(*2*))
note *ln34* = *True this*
have *ltrr2*: *ltrr2* = *lltrv2* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
unfolding *ltrr2 ltrv2-lnever*[*OF current*] **by** *simp*
show *?thesis*
using *lcompletedFrom-lltrv2*[*OF unw* Δ *r ltr1 ln3 ltr2 ln4*, *of L*]
using *lfin*[*unfolded ltrr2*]
unfolding *Van.lcompletedFrom-def ltrr2*[*symmetric*]
using *llist-of-list-of* **by** *fastforce*
next
case *False* **note** *ln3* = *False*
hence *ln4*: ¬ *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34* *ltr2*(*2*))

have *ltr2* ≠ [[*s2*]] **using** *ln4 ltr2*
using *Opt.final-not-isInt* **by** *auto*
hence *llength ltr2* > *Suc 0*
by (*metis Opt.lcompletedFrom-singl Suc-ile-eq current*(*2*) *enat-0-iff*(*2*)
linorder-not-less llength-LNil llist-eq-cong ltr2(*1*) *ltr2*(*2*) *nle-le not-less-zero*)
hence ¬ *finalO s2*
by (*metis Opt.final-def Opt.lvalidFromS-Cons-iff current*(*2*) *eSuc-enat enat-0*

linorder-neq-iff llength-LCons llength-LNil llist.exhaust-sel ltr2(*1*))
hence *nf12*: ¬ *finalV sv1* ∧ ¬ *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *force*

```
    obtain w1' w2' tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statAA
statOO
     where φφ: φφ s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'
      and φ': φ' Δ w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA
statO sv1 trv1 sv1'' sv2 trv2 sv2'' statOO
     and ltrr2: ltrr2 =
     lappend (llist-of trv2) (ltrv2 (w1',w2',s1'',s1'' $ ltr1',s2'',s2'' $ ltr2',statAA,sv1'',sv2'',statOO))

     using ltrv1-ltrv2-not-lnever[OF unw Δ r stat ltr1 ltr2 ln3 A34]
     unfolding ltrr2 by blast
     define ltrr2' where ltrr2': ltrr2' = ltrv2 (w1',w2',s1'',s1'' $ ltr1',s2'',s2''
$ ltr2',statAA,sv1'',sv2'',statOO)
     have ltrr2: ltrr2 = lappend (llist-of trv2) ltrr2'
     unfolding ltrr2 ltrr2' ..
     have ne: trv2 ≠ [] ∨ (trv2 = [] ∧ w2' < w2)
     using φ' unfolding φ'-def ltrr2 by simp

     have lfin': lfinite ltrr2'
     using lfin ne unfolding ltrr2 by simp
     have len: length (list-of ltrr2') < length (list-of ltrr2) ∨
            length (list-of ltrr2') = length (list-of ltrr2) ∧ w2' < w2
     using ne lfin lfin' by (simp add: list-of-lappend ltrr2)

     have 0: list-of ltrr2' ≠ [] ∧ finalV (last (list-of ltrr2'))
     using len proof(elim disjE conjE)
      assume len: length (list-of ltrr2') < length (list-of ltrr2)
      show ?thesis
      apply(rule less(1)[OF - ltrr2'])
        subgoal by fact subgoal by fact
        subgoal using φ' unfolding φ'-def by simp
        subgoal using r(1) φφ unfolding φφ-def
            by (metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2)
        subgoal using r(2) φφ unfolding φφ-def
            by (metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2)
        subgoal using r(3) φ' unfolding φ'-def
        by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
        subgoal using r(4) φ' unfolding φ'-def
        by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
        subgoal using φ' unfolding φ'-def by auto
        subgoal using φφ unfolding φφ-def by simp
        subgoal using φφ unfolding φφ-def by simp
        subgoal using φφ unfolding φφ-def by simp
        subgoal using φφ unfolding φφ-def by simp
        subgoal using φφ unfolding φφ-def by simp .
     next
      assume len: length (list-of ltrr2') = length (list-of ltrr2) w2' < w2
      show ?thesis
```

apply(*rule less*(*2*)[*OF* - - *ltrr2'*])
subgoal by *fact* **subgoal unfolding** *len* **..**
subgoal by *fact*
subgoal using *φ'* **unfolding** *φ'-def* **by** *simp*
subgoal using *r*(*1*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) *φ'* **unfolding** *φ'-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) *φ'* **unfolding** *φ'-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *φ'* **unfolding** *φ'-def* **by** *auto*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp* **.**
qed
show *?thesis* **unfolding** *ltrr2* **using** *0*
by (*simp add*: *lfin' list-of-lappend*)
qed
qed
}
thus *?thesis* **unfolding** *Van.lcompletedFrom-def* **by** *auto*
qed

lemma *lS-ltrv1-ltr1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lS* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) = *Opt.lS ltr1*
proof−
have *cltrv1*: *Van.lcompletedFrom sv1* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-ltrv1*[*OF assms*] **.**
{fix *nL nR ltrr1 ltr1*
assume ∃ *w1 w2 s1 s2 ltr2 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w1* ∧
ltrr1 = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧

(statA = Diff ⟶ statO = Diff) ∧
Opt.lvalidFromS s1 ltr1 ∧ Opt.lcompletedFrom s1 ltr1 ∧
Opt.lvalidFromS s2 ltr2 ∧ Opt.lcompletedFrom s2 ltr2 ∧
Opt.lA ltr1 = Opt.lA ltr2
hence *TwoFuncPred.sameFM isSecV isSecO getSecV getSecO nL nR ltrr1 ltr1*
proof(*coinduct rule*: *TwoFuncPred.sameFM.coinduct*[*of λnL nR ltrr1 ltr1*.
∃ w1 w2 s1 s2 ltr2 statA sv1 sv2 statO.
nL = w1 ∧ nR = w1 ∧
ltrr1 = ltrv1 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) ∧
Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO ∧
reachO s1 ∧ reachO s2 ∧ reachV sv1 ∧ reachV sv2 ∧
(statA = Diff ⟶ statO = Diff) ∧
Opt.lvalidFromS s1 ltr1 ∧ Opt.lcompletedFrom s1 ltr1 ∧
Opt.lvalidFromS s2 ltr2 ∧ Opt.lcompletedFrom s2 ltr2 ∧
Opt.lA ltr1 = Opt.lA ltr2])
case (*2 nL nR ltrr1 ltr1*)
then obtain *w1 w2 s1 s2 ltr2 statA sv1 sv2 statO*
where *nL*: *nL = w1* **and** *nR*: *nR = w1*
and *ltrr1*: *ltrr1 = ltrv1 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)*
and Δ: *Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA = Diff ⟶ statO = Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1 = Opt.lA ltr2*
by *auto*

have *current*: *ltr1 ≠ [[]] ltr2 ≠ [[]]*
using *ltr1(2) ltr2(2)* **unfolding** *Opt.lcompletedFrom-def* **by** *auto*

show *?case* **proof**(*cases lnever isIntO ltr1*)
case *True* **note** *ln3 = True* **note** *current = current True*
hence *ln4*: *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1(2) A34 ltr2(2)*)
note *ln34 = True this*
have *ltrr1*: *ltrr1 = lltrv1 (L, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)*
unfolding *ltrr1 ltrv1-lnever[OF current]* **by** *simp*

have *clltrv1*: *Van.lcompletedFrom sv1 (lltrv1 (L,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))*
using *lcompletedFrom-lltrv1[OF unw Δ r ltr1 ln3 ltr2 ln4]* **.**

show *?thesis* **apply**(*rule TwoFuncPred.sameFM-selectlmap-lfilter*)
unfolding *ltrr1* **apply** *simp*
using *lS-lltrv1-ltr1[OF unw Δ r ltr1 ln3 ltr2 ln4, of L]*
unfolding *Van.lS[OF clltrv1] Opt.lS[OF ltr1(2)]* **.**
next
case *False* **note** *ln3 = False*

hence *ln4*: ¬ *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34 ltr2*(*2*))

have *ltr1* ≠ [[*s1*]] **using** *ln3 ltr1*
using *Opt.final-not-isInt* **by** *auto*
hence *llength ltr1* > *Suc 0*
by (*metis Opt.lcompletedFrom-singl Suc-ile-eq current*(*1*) *enat-0-iff*(*1*) *linorder-not-less llength-LNil llist-eq-cong ltr1*(*1*) *ltr1*(*2*) *nle-le not-less-zero*)
hence ¬ *finalO s1*
by (*metis Opt.final-def Opt.lvalidFromS-Cons-iff current*(*1*) *eSuc-enat enat-0*

linorder-neq-iff llength-LCons llength-LNil llist.exhaust-sel ltr1(*1*))
hence *nf12*: ¬ *finalV sv1* ∧ ¬ *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *force*

obtain *w1′ w2′ tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statAA statOO*
where *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′*
and *φ′*: *φ′* Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1′′ s2 tr2 s2′ s2′′ statAA statO sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
and *ltrr1*: *ltrr1* =
lappend (*llist-of trv1*) (*ltrv1* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*))

using *ltrv1-ltrv2-not-lnever*[*OF unw* Δ *r stat ltr1 ltr2 ln3 A34*]
unfolding *ltrr1* **by** *blast*
define *ltrr1′* **where** *ltrr1′*: *ltrr1′* = *ltrv1* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*)
have *ltrr1*: *ltrr1* = *lappend* (*llist-of trv1*) *ltrr1′*
unfolding *ltrr1 ltrr1′* **..**
have *ne1*: *trv1* ≠ [] ∨ *w1′* < *w1*
using *φ′* **unfolding** *φ′-def ltrr1* **by** *simp*

show *?thesis*
apply(*rule TwoFuncPred.sameFM-selectlappend*)
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w1*])

apply(*rule exI*[*of - tr1 ## s1′*]) **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w1*])
apply(*rule exI*[*of - ltrr1′*]) **apply**(*rule exI*[*of - s1′′* $ *ltr1′*])
apply(*intro conjI*)
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal using *ltrr1* **.**
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal by *fact*
subgoal by *simp*
subgoal using *φ′* **unfolding** *φ′-def* **unfolding** *Van.S.map-filter Opt.S.map-filter* **by** *simp*
subgoal apply(*rule disjI1*)

apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1''*])
apply(*rule exI*[*of - s2''*]) **apply**(*rule exI*[*of - s2'' $ ltr2'*])
apply(*rule exI*[*of - statAA*])
apply(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - sv2''*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrr1'* **..**
subgoal using φ' **unfolding** φ'*-def* **by** *simp*
subgoal using *r*(*1*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using φ' **unfolding** φ'*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp* **. .**
qed
qed
}
thus *?thesis* **apply**−
unfolding *Van.lS*[*OF cltrv1*] *Opt.lS*[*OF ltr1*(*2*)] **apply**(*rule TwoFuncPred.sameFM-lmap-lfilter*)
using *assms* **by** *blast*
qed

lemma *lS-ltrv2-ltr2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lS* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) = *Opt.lS ltr2*
proof−
have *cltrv2*: *Van.lcompletedFrom sv2* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-ltrv2*[*OF assms*] **.**
{fix *nL nR ltrr2 ltr2*
assume ∃ *w1 w2 s1 s2 ltr1 statA sv1 sv2 statO*.
nL = *w2* ∧ *nR* = *w2* ∧

```
    ltrr2 = ltrv2 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) ∧
    Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO ∧
    reachO s1 ∧ reachO s2 ∧ reachV sv1 ∧ reachV sv2 ∧
    (statA = Diff ⟶ statO = Diff) ∧
    Opt.lvalidFromS s1 ltr1 ∧ Opt.lcompletedFrom s1 ltr1 ∧
    Opt.lvalidFromS s2 ltr2 ∧ Opt.lcompletedFrom s2 ltr2 ∧
    Opt.lA ltr1 = Opt.lA ltr2
  hence TwoFuncPred.sameFM isSecV isSecO getSecV getSecO nL nR ltrr2 ltr2
  proof(coinduct rule: TwoFuncPred.sameFM.coinduct[of λnL nR ltrr2 ltr2.
    ∃ w1 w2 s1 s2 ltr1 statA sv1 sv2 statO.
    nL = w2 ∧ nR = w2 ∧
    ltrr2 = ltrv2 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) ∧
    Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO ∧
    reachO s1 ∧ reachO s2 ∧ reachV sv1 ∧ reachV sv2 ∧
    (statA = Diff ⟶ statO = Diff) ∧
    Opt.lvalidFromS s1 ltr1 ∧ Opt.lcompletedFrom s1 ltr1 ∧
    Opt.lvalidFromS s2 ltr2 ∧ Opt.lcompletedFrom s2 ltr2 ∧
    Opt.lA ltr1 = Opt.lA ltr2])
   case (2 nL nR ltrr2 ltr2)
   then obtain w1 w2 s1 s2 ltr1 statA sv1 sv2 statO
   where nL: nL = w2 and nR: nR = w2
   and ltrr2: ltrr2 = ltrv2 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)
   and Δ: Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO
   and r: reachO s1 reachO s2 reachV sv1 reachV sv2
   and stat: statA = Diff ⟶ statO = Diff
   and ltr1: Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1
   and ltr2: Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2
   and A34: Opt.lA ltr1 = Opt.lA ltr2
   by auto

   have current: ltr1 ≠ [[]] ltr2 ≠ [[]]
   using ltr1(2) ltr2(2) unfolding Opt.lcompletedFrom-def by auto

   show ?case proof(cases lnever isIntO ltr1)
     case True note ln3 = True note current = current True
     hence ln4: lnever isIntO ltr2
      by (metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1(2) A34
ltr2(2))
     note ln34 = True this
      have ltrr2: ltrr2 = lltrv2 (L, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2,
statO)
     unfolding ltrr2 ltrv2-lnever[OF current] by simp

   have clltrv2: Van.lcompletedFrom sv2 (lltrv2 (L,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))
     using lcompletedFrom-lltrv2[OF unw Δ r ltr1 ln3 ltr2 ln4] .

     show ?thesis apply(rule TwoFuncPred.sameFM-selectlmap-lfilter)
     unfolding ltrr2 apply simp
     using lS-lltrv2-ltr2[OF unw Δ r ltr1 ln3 ltr2 ln4, of L]
```

unfolding *Van.lS*[*OF clltrv2*] *Opt.lS*[*OF ltr2*(*2*)] **.**
next
case *False* **note** *ln3* = *False*
hence *ln4*: ¬ *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34 ltr2*(*2*))

have *ltr2* ≠ [[*s2*]] **using** *ln4 ltr2*
using *Opt.final-not-isInt* **by** *auto*
hence *llength ltr2* > *Suc 0*
by (*metis Opt.lcompletedFrom-singl Suc-ile-eq current*(*2*) *enat-0-iff*(*2*) *linorder-not-less llength-LNil llist-eq-cong ltr2*(*1*) *ltr2*(*2*) *nle-le not-less-zero*)
hence ¬ *finalO s2*
by (*metis Opt.final-def Opt.lvalidFromS-Cons-iff current*(*2*) *eSuc-enat enat-0*

linorder-neq-iff llength-LCons llength-LNil llist.exhaust-sel ltr2(*1*))
hence *nf12*: ¬ *finalV sv1* ∧ ¬ *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *force*

obtain *w1′ w2′ tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′ trv1 sv1″ trv2 sv2″ statAA statOO*
where *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′*
and *φ′*: *φ′* Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1″ s2 tr2 s2′ s2″ statAA statO sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
and *ltrr2*: *ltrr2* =
lappend (*llist-of trv2*) (*ltrv2* (*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2″*,*s2″* $ *ltr2′*,*statAA*,*sv1″*,*sv2″*,*statOO*))

using *ltrv1-ltrv2-not-lnever*[*OF unw* Δ *r stat ltr1 ltr2 ln3 A34*]
unfolding *ltrr2* **by** *blast*
define *ltrr2′* **where** *ltrr2′*: *ltrr2′* = *ltrv2* (*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2″*,*s2″* $ *ltr2′*,*statAA*,*sv1″*,*sv2″*,*statOO*)
have *ltrr1*: *ltrr2* = *lappend* (*llist-of trv2*) *ltrr2′*
unfolding *ltrr2 ltrr2′* **..**
have *ne2*: *trv2* ≠ [] ∨ *w2′* < *w2*
using *φ′* **unfolding** *φ′-def ltrr2* **by** *simp*

show *?thesis*
apply(*rule TwoFuncPred.sameFM-selectlappend*)
apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - w2′*]) **apply**(*rule exI*[*of - w2*])

apply(*rule exI*[*of - tr2 ## s2′*]) **apply**(*rule exI*[*of - w2′*]) **apply**(*rule exI*[*of - w2*])
apply(*rule exI*[*of - ltrr2′*]) **apply**(*rule exI*[*of - s2″* $ *ltr2′*])
apply(*intro conjI*)
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal using *ltrr1* **.**
subgoal using *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal by *fact*
subgoal by *simp*

subgoal using φ' **unfolding** φ'*-def* **unfolding** *Van.S.map-filter Opt.S.map-filter* **by** *simp*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1''*])
apply(*rule exI*[*of - s2''*]) **apply**(*rule exI*[*of - s1'' $ ltr1'*])
apply(*rule exI*[*of - statAA*])
apply(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - sv2''*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrr2'* **..**
subgoal using φ' **unfolding** φ'*-def* **by** *simp*
subgoal using *r*(*1*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using φ' **unfolding** φ'*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp* **. .**
qed
qed
}
thus *?thesis* **apply**−
unfolding *Van.lS*[*OF cltrv2*] *Opt.lS*[*OF ltr2*(*2*)] **apply**(*rule TwoFuncPred.sameFM-lmap-lfilter*)
using *assms* **by** *blast*
qed

lemma *lA-ltrv1-ltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* $\longrightarrow$ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lA* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) =
Van.lA (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))

proof−
have *cltrv1*: *Van.lcompletedFrom sv1* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-ltrv1*[*OF assms*] **.**
have *cltrv2*: *Van.lcompletedFrom sv2* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-ltrv2*[*OF assms*] **.**
{fix *nL nR ltrr1 ltrr2*
assume ∃ *w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w2* ∧
ltrr1 = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
ltrr2 = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧
Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧
Opt.lA ltr1 = *Opt.lA ltr2*
hence *TwoFuncPred.sameFM isIntV isIntV getActV getActV nL nR ltrr1 ltrr2*
proof(*coinduct rule*: *TwoFuncPred.sameFM.coinduct*[*of* λ*nL nR ltrr1 ltrr2*.
∃ *w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*.
nL = *w1* ∧ *nR* = *w2* ∧
ltrr1 = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
ltrr2 = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧
Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧
Opt.lA ltr1 = *Opt.lA ltr2*])
case (*2 nL nR ltrr1 ltrr2*)
then obtain *w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO*
where *nL*: *nL* = *w1* **and** *nR*: *nR* = *w2*
and *ltrr1*: *ltrr1* = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *ltrr2*: *ltrr2* = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
by *auto*

have *current*: *ltr1* ≠ [[]] *ltr2* ≠ [[]]
using *ltr1*(*2*) *ltr2*(*2*) **unfolding** *Opt.lcompletedFrom-def* **by** *auto*

show *?case* **proof**(*cases lnever isIntO ltr1*)
case *True* **note** *ln3* = *True* **note** *current* = *current True*
hence *ln4*: *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34*
ltr2(*2*))

note *ln34* = *True this*
have *ltrr1*: *ltrr1* = *lltrv1* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
unfolding *ltrr1 ltrv1-lnever*[*OF current*] **by** *simp*
have *ltrr2*: *ltrr2* = *lltrv2* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
unfolding *ltrr2 ltrv2-lnever*[*OF current*] **by** *simp*

have *clltrv1*: *Van.lcompletedFrom sv1* (*lltrv1* (*L*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-lltrv1*[*OF unw* Δ *r ltr1 ln3 ltr2 ln4*] **.**
have *clltrv2*: *Van.lcompletedFrom sv2* (*lltrv2* (*L*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-lltrv2*[*OF unw* Δ *r ltr1 ln3 ltr2 ln4*] **.**

show *?thesis* **apply**(*rule TwoFuncPred.sameFM-selectlmap-lfilter*)
unfolding *ltrr1 ltrr2* **apply** *simp*
using *lA-lltrv1-lltrv2*[*OF unw* Δ *r ltr1 ln3 ltr2 ln4*, *of L*]
unfolding *Van.lA*[*OF clltrv1*] *Van.lA*[*OF clltrv2*] **.**
next
case *False* **note** *ln3* = *False*
hence *ln4*: ¬ *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34 ltr2*(*2*))

have *ltr1* ≠ [[*s1*]] **using** *ln3 ltr1*
using *Opt.final-not-isInt* **by** *auto*
hence *llength ltr1* > *Suc 0*
by (*metis Opt.lcompletedFrom-singl Suc-ile-eq current*(*1*) *enat-0-iff*(*1*) *linorder-not-less llength-LNil llist-eq-cong ltr1*(*1*) *ltr1*(*2*) *nle-le not-less-zero*)
hence ¬ *finalO s1*
by (*metis Opt.final-def Opt.lvalidFromS-Cons-iff current*(*1*) *eSuc-enat enat-0*

linorder-neq-iff llength-LCons llength-LNil llist.exhaust-sel ltr1(*1*))
hence *nf12*: ¬ *finalV sv1* ∧ ¬ *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *force*

obtain *w1′ w2′ tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′ trv1 sv1″ trv2 sv2″ statAA statOO*
where *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′*
and *φ′*: *φ′* Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1″ s2 tr2 s2′ s2″ statAA statO sv1 trv1 sv1″ sv2 trv2 sv2″ statOO*
and *ltrr1*: *ltrr1* =
lappend (*llist-of trv1*) (*ltrv1* (*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2″*,*s2″* $ *ltr2′*,*statAA*,*sv1″*,*sv2″*,*statOO*))

and *ltrr2*: *ltrr2* =
lappend (*llist-of trv2*) (*ltrv2* (*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2″*,*s2″* $ *ltr2′*,*statAA*,*sv1″*,*sv2″*,*statOO*))

using *ltrv1-ltrv2-not-lnever*[*OF unw* Δ *r stat ltr1 ltr2 ln3 A34*]
unfolding *ltrr1 ltrr2* **by** *blast*
define *ltrr1′* **where** *ltrr1′*: *ltrr1′* = *ltrv1* (*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2″*,*s2″*

$ *ltr2′*,*statAA*,*sv1″*,*sv2″*,*statOO*)
have *ltrr1*: *ltrr1* = *lappend* (*llist-of trv1*) *ltrr1′*
unfolding *ltrr1 ltrr1′* **..**
have *ne1*: *trv1* ≠ [] ∨ *w1′* < *w1*
using *φ′* **unfolding** *φ′-def ltrr1* **by** *simp*
define *ltrr2′* **where** *ltrr2′*: *ltrr2′* = *ltrv2* (*w1′*,*w2′*,*s1″*,*s1″* $ *ltr1′*,*s2″*,*s2″*
$ *ltr2′*,*statAA*,*sv1″*,*sv2″*,*statOO*)
have *ltrr2*: *ltrr2* = *lappend* (*llist-of trv2*) *ltrr2′*
unfolding *ltrr2 ltrr2′* **..**
have *ne2*: *trv2* ≠ [] ∨ *w2′* < *w2*
using *φ′* **unfolding** *φ′-def ltrr1* **by** *simp*

show *?thesis*
apply(*rule TwoFuncPred.sameFM-selectlappend*)
apply(*rule exI*[*of - trv1*]) **apply**(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w1*])

apply(*rule exI*[*of - trv2*]) **apply**(*rule exI*[*of - w2′*]) **apply**(*rule exI*[*of - w2*])

apply(*rule exI*[*of - ltrr1′*]) **apply**(*rule exI*[*of - ltrr2′*])
apply(*intro conjI*)
subgoal unfolding *nL* **..** **subgoal unfolding** *nR* **..**
subgoal using *ltrr1* **.**
subgoal using *ltrr2* **.**
subgoal by *fact* **subgoal by** *fact*
subgoal using *φ′* **unfolding** *φ′-def* **unfolding** *Van.A.map-filter* **by** *simp*
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - w1′*]) **apply**(*rule exI*[*of - w2′*])
apply(*rule exI*[*of - s1″*]) **apply**(*rule exI*[*of - s1″* $ *ltr1′*])
apply(*rule exI*[*of - s2″*]) **apply**(*rule exI*[*of - s2″* $ *ltr2′*])
apply(*rule exI*[*of - statAA*])
apply(*rule exI*[*of - sv1″*]) **apply**(*rule exI*[*of - sv2″*])
apply(*rule exI*[*of - statOO*])
apply(*intro conjI*)
subgoal .. subgoal ..
subgoal unfolding *ltrr1′* **..**
subgoal unfolding *ltrr2′* **..**
subgoal using *φ′* **unfolding** *φ′-def* **by** *simp*
subgoal using *r*(*1*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) *φ′* **unfolding** *φ′-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) *φ′* **unfolding** *φ′-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *φ′* **unfolding** *φ′-def* **by** *simp*
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def* **by** *simp*

subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def* **by** *simp*
subgoal using *r*(*2*) *φφ* **unfolding** *φφ-def* **by** *simp* **. .**
qed
qed
}
thus *?thesis* **apply**−
unfolding *Van.lA*[*OF cltrv1*] *Van.lA*[*OF cltrv2*] **apply**(*rule TwoFuncPred.sameFM-lmap-lfilter*)
using *assms* **by** *blast*
qed

lemma *lO-ltrv1-ltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
and *O12*: *Van.lO* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) =
Van.lO (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
and *stO*: *statO* = *Eq*
shows *Opt.lO ltr1* = *Opt.lO ltr2*
proof−
have *cltrv1*: *Van.lcompletedFrom sv1* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-ltrv1*[*OF assms*(*1*−*12*)] **.**
have *cltrv2*: *Van.lcompletedFrom sv2* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-ltrv2*[*OF assms*(*1*−*12*)] **.**
{fix *nL nR ltr1 ltr2*
assume ∃ *ltrr1 ltrr2 w1 w2 s1 s2 statA sv1 sv2 statO*.
ltrr1 = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
ltrr2 = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧
Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧
Opt.lA ltr1 = *Opt.lA ltr2* ∧
Van.lO ltrr1 = *Van.lO ltrr2* ∧
statO = *Eq*
hence *TwoFuncPred.sameFM isIntO isIntO getObsO getObsO nL nR ltr1 ltr2*
proof(*coinduct rule*: *TwoFuncPred.sameFM.coinduct*[*of* λ*nL nR ltr1 ltr2*.
∃ *ltrr1 ltrr2 w1 w2 s1 s2 statA sv1 sv2 statO*.
ltrr1 = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
ltrr2 = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ∧
Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO* ∧

reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
(*statA* = *Diff* ⟶ *statO* = *Diff*) ∧
Opt.lvalidFromS s1 ltr1 ∧ *Opt.lcompletedFrom s1 ltr1* ∧
Opt.lvalidFromS s2 ltr2 ∧ *Opt.lcompletedFrom s2 ltr2* ∧
Opt.lA ltr1 = *Opt.lA ltr2* ∧
Van.lO ltrr1 = *Van.lO ltrr2* ∧
statO = *Eq*])
case (*2 nL nR ltr1 ltr2*)
then obtain *ltrr1 ltrr2 w1 w2 s1 s2 statA sv1 sv2 statO* **where**
ltrr11: *ltrr1* = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and *ltrr22*: *ltrr2* = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
and *O12*: *Van.lO ltrr1* = *Van.lO ltrr2*
and *stO*: *statO* = *Eq*
by *auto*

have *stA*: *statA* = *Eq* **using** *stat stO*
using *status.exhaust* **by** *blast*

have *current*: *ltr1* ≠ [[]] *ltr2* ≠ [[]]
using *ltr1*(*2*) *ltr2*(*2*) **unfolding** *Opt.lcompletedFrom-def* **by** *auto*

show *?case* **proof**(*cases lnever isIntO ltr1*)
case *True* **note** *ln3* = *True* **note** *current* = *current True*
hence *ln4*: *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34*
ltr2(*2*))
note *ln34* = *True this*
have *ltrr1*: *ltrr1* = *lltrv1* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*,
statO)
unfolding *ltrr11 ltrv1-lnever*[*OF current*] **by** *simp*
have *ltrr2*: *ltrr2* = *lltrv2* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*,
statO)
unfolding *ltrr22 ltrv2-lnever*[*OF current*] **by** *simp*

have *clltrv1*: *Van.lcompletedFrom sv1* (*lltrv1* (*L*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-lltrv1*[*OF unw* Δ *r ltr1 ln3 ltr2 ln4*] **.**
have *clltrv2*: *Van.lcompletedFrom sv2* (*lltrv2* (*L*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
using *lcompletedFrom-lltrv2*[*OF unw* Δ *r ltr1 ln3 ltr2 ln4*] **.**

show *?thesis* **apply**(*rule TwoFuncPred.sameFM-selectlmap-lfilter*)
unfolding *Opt.lO*[*OF ltr1*(*2*)] *Opt.lO*[*OF ltr2*(*2*)]
by (*metis ln3 ln4 lnever-LNil-lfilter′*)

next
case *False* **note** *ln3* = *False*
hence *ln4*: ¬ *lnever isIntO ltr2*
by (*metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1*(*2*) *A34 ltr2*(*2*))

have *ltr1* ≠ [[*s1*]] **using** *ln3 ltr1*
using *Opt.final-not-isInt* **by** *auto*
hence *llength ltr1* > *Suc 0*
by (*metis Opt.lcompletedFrom-singl Suc-ile-eq current*(*1*) *enat-0-iff*(*1*) *linorder-not-less llength-LNil llist-eq-cong ltr1*(*1*) *ltr1*(*2*) *nle-le not-less-zero*)
hence ¬ *finalO s1*
by (*metis Opt.final-def Opt.lvalidFromS-Cons-iff current*(*1*) *eSuc-enat enat-0*

linorder-neq-iff llength-LCons llength-LNil llist.exhaust-sel ltr1(*1*))
hence *nf12*: ¬ *finalV sv1* ∧ ¬ *finalV sv2*
using Δ *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *unw unwindCond-def* **by** *force*

obtain *w1′ w2′ tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′ trv1 sv1′′ trv2 sv2′′ statAA statOO*
where *φφ*: *φφ s1 ltr1 s2 ltr2 tr1 s1′ s1′′ ltr1′ tr2 s2′ s2′′ ltr2′*
and *φ′*: *φ′* Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1′′ s2 tr2 s2′ s2′′ statAA statO sv1 trv1 sv1′′ sv2 trv2 sv2′′ statOO*
and *ltrr1*: *ltrr1* =
lappend (*llist-of trv1*) (*ltrv1* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*))

and *ltrr2*: *ltrr2* =
lappend (*llist-of trv2*) (*ltrv2* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*))

using *ltrv1-ltrv2-not-lnever*[*OF unw* Δ *r stat ltr1 ltr2 ln3 A34*]
unfolding *ltrr11 ltrr22* **by** *blast*
define *ltrr1′* **where** *ltrr1′*: *ltrr1′* = *ltrv1* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*)
have *ltrr1*: *ltrr1* = *lappend* (*llist-of trv1*) *ltrr1′*
unfolding *ltrr1 ltrr1′* **..**
have *ne1*: *trv1* ≠ [] ∨ *w1′* < *w1*
using *φ′* **unfolding** *φ′-def ltrr1* **by** *simp*
define *ltrr2′* **where** *ltrr2′*: *ltrr2′* = *ltrv2* (*w1′*,*w2′*,*s1′′*,*s1′′* $ *ltr1′*,*s2′′*,*s2′′* $ *ltr2′*,*statAA*,*sv1′′*,*sv2′′*,*statOO*)
have *ltrr2*: *ltrr2* = *lappend* (*llist-of trv2*) *ltrr2′*
unfolding *ltrr2 ltrr2′* **..**
have *ne2*: *trv2* ≠ [] ∨ *w2′* < *w2*
using *φ′* **unfolding** *φ′-def ltrr1* **by** *simp*

have *ltr1-eq*: *ltr1* = *lappend* (*llist-of* (*tr1* ## *s1′*)) (*s1′′* $ *ltr1′*)
and *ltr2-eq*: *ltr2* = *lappend* (*llist-of* (*tr2* ## *s2′*)) (*s2′′* $ *ltr2′*) **using** *φφ* **unfolding** *φφ-def* **by** *auto*

have *sst*: *statOO* = *Diff* ⟷ *Van.O* (*trv1* ## *sv1′′*) ≠ *Van.O* (*trv2* ##

sv2'')
statA = Eq ⟹ *statAA = Diff* ⟷ *Opt.O* ((*tr1 ## s1'*) *## s1''*) ≠ *Opt.O* ((*tr2 ## s2'*) *## s2''*)
statO = Diff ⟹ *statOO = Diff*
statAA = Diff ⟹ *statOO = Diff*
using *φ' stO* **unfolding** *φ'-def* **by** *auto*

have *Atrv12'*: *Van.A* (*trv1 ## sv1''*) = *Van.A* (*trv2 ## sv2''*)
using *φ'* **unfolding** *φ'-def* **by** *auto*

have *Δ'*: Δ ∞ *w1' w2' s1'' s2'' statAA sv1'' sv2'' statOO*
using *φ'* **unfolding** *φ'-def* **by** *auto*

have *vltrv1*: *Van.lvalidFromS sv1 ltrr1*
unfolding *ltrr11* **using** *lvalidFromS-ltrv1*
using *A34* Δ *ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *stat unw* **by** *blast*
have *cltrv1*: *Van.lcompletedFrom sv1 ltrr1*
unfolding *ltrr11* **using** *lcompletedFrom-ltrv1*
using *A34* Δ *ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *stat unw* **by** *blast*

have *vltrv2*: *Van.lvalidFromS sv2 ltrr2*
unfolding *ltrr22* **using** *lvalidFromS-ltrv2*
using *A34* Δ *ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *stat unw* **by** *blast*
have *cltrv2*: *Van.lcompletedFrom sv2 ltrr2*
unfolding *ltrr22* **using** *lcompletedFrom-ltrv2*
using *A34* Δ *ltr1*(*1*) *ltr1*(*2*) *ltr2*(*1*) *ltr2*(*2*) *r*(*1*) *r*(*2*) *r*(*3*) *r*(*4*) *stat unw* **by** *blast*

have *Oltrr1*: *Van.lO ltrr1* = *lmap getObsV* (*lfilter isIntV ltrr1*)
using *Van.lO*[*OF cltrv1*] **.**
have *Oltrr2*: *Van.lO ltrr2* = *lmap getObsV* (*lfilter isIntV ltrr2*)
using *Van.lO*[*OF cltrv2*] **.**

have *cltrv1'*: *Van.lcompletedFrom* (*lastt sv1 trv1*) *ltrr1'*
using *cltrv1* **unfolding** *ltrr1 Van.lcompletedFrom-def Van.final-def* **apply** *simp*
using *φ'* **unfolding** *φ'-def*
by (*metis Van.validS-validTrans Van.validFromS-def lappend-LNil2 last-appendR lfinite-llist-of list-of-lappend*
list-of-llist-of llist-of.simps(*1*) *snoc-eq-iff-butlast*)

have *cltrv2'*: *Van.lcompletedFrom* (*lastt sv2 trv2*) *ltrr2'*
using *cltrv2* **unfolding** *ltrr2 Van.lcompletedFrom-def Van.final-def* **apply** *simp*
using *φ'* **unfolding** *φ'-def*
by (*metis Van.validS-validTrans Van.validFromS-def lappend-LNil2 last-appendR*

lfinite-llist-of list-of-lappend
list-of-llist-of llist-of.simps(*1*) *snoc-eq-iff-butlast*)

have *Oltrr1'*: *Van.lO ltrr1'* = *lmap getObsV* (*lfilter isIntV ltrr1'*)
using *Van.lO*[*OF cltrv1'*] **.**
have *Oltrr2'*: *Van.lO ltrr2'* = *lmap getObsV* (*lfilter isIntV ltrr2'*)
using *Van.lO*[*OF cltrv2'*] **.**

have *Van.O* (*trv1* ## *sv1''*) = *Van.O* (*trv2* ## *sv2''*) ∧
Van.lO ltrr1' = *Van.lO ltrr2'*
using *Atrv12' O12 Oltrr1' Oltrr2'* **unfolding** *Oltrr1 Oltrr2* **unfolding** *ltrr1 ltrr2*
unfolding *Van.A.map-filter Van.O.map-filter Van.lO.lmap-lfilter* **apply** *simp*

unfolding *lmap-lappend-distrib* **apply** *simp* **apply**(*subst* (*asm*) *lappend-llist-of-inj*)

using *map-eq-imp-length-eq* **by** *auto*
hence *O12''*: *Van.O* (*trv1* ## *sv1''*) = *Van.O* (*trv2* ## *sv2''*)
and *O12'*: *Van.lO ltrr1'* = *Van.lO ltrr2'* **by** *auto*

have *stOO*: *statOO* = *Eq* **using** *O12'' sst* **by**(*cases statOO*, *auto*)

have *O34'*: *Opt.O* ((*tr1* ## *s1'*) ## *s1''*) = *Opt.O* ((*tr2* ## *s2'*) ## *s2''*)
using *stOO sst*(*2*) *sst*(*4*) *stA* **by** *blast*

hence *s14'*: *getObsO s1'* = *getObsO s2'*
using *φφ* **unfolding** *φφ-def Opt.O.map-filter* **by** (*simp add*: *never-Nil-filter*)

show *?thesis*
apply(*rule TwoFuncPred.sameFM-selectlappend*)
apply(*rule exI*[*of - tr1* ## *s1'*]) **apply**(*rule exI*[*of - undefined*]) **apply**(*rule exI*[*of - nL*])
apply(*rule exI*[*of - tr2* ## *s2'*]) **apply**(*rule exI*[*of - undefined*]) **apply**(*rule exI*[*of - nR*])
apply(*rule exI*[*of - s1''* $ *ltr1'*]) **apply**(*rule exI*[*of - s2''* $ *ltr2'*])
apply(*intro conjI*)
subgoal .. **subgoal ..**
subgoal by *fact* **subgoal by** *fact*
subgoal by *simp* **subgoal by** *simp*
subgoal using *φφ* **unfolding** *φφ-def* **unfolding** *Opt.O.map-filter*
by (*simp add*: *s14' never-Nil-filter*)
subgoal apply(*rule disjI1*)
apply(*rule exI*[*of - ltrr1'*]) **apply**(*rule exI*[*of - ltrr2'*])
apply(*rule exI*[*of - w1'*]) **apply**(*rule exI*[*of - w2'*])
apply(*rule exI*[*of - s1''*]) **apply**(*rule exI*[*of - s2''*])
apply(*rule exI*[*of - statAA*])
apply(*rule exI*[*of - sv1''*]) **apply**(*rule exI*[*of - sv2''*])
apply(*rule exI*[*of - statOO*])

apply(*intro conjI*)
subgoal unfolding *ltrr1′* **..**
subgoal unfolding *ltrr2′* **..**
subgoal using φ' **unfolding** φ'*-def* **by** *simp*
subgoal using *r*(*1*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def*
by (*metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc not-Cons-self2*)
subgoal using *r*(*3*) φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using *r*(*4*) φ' **unfolding** φ'*-def*
by (*metis Van.reach-validFromS-reach snoc-eq-iff-butlast*)
subgoal using φ' **unfolding** φ'*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal using *r*(*2*) $\varphi\varphi$ **unfolding** $\varphi\varphi$*-def* **by** *simp*
subgoal by *fact*
subgoal by *fact* **. .**
qed
qed
}
thus *?thesis*
unfolding *Opt.lO*[*OF ltr1*(*2*)] *Opt.lO*[*OF ltr2*(*2*)] **apply**(*rule TwoFuncPred.sameFM-lmap-lfilter*[**where** *wL* = *undefined* **and** *wR* = *undefined*])
using *assms* **by** *blast*
qed

end

theorem *unwind-lrsecure*:
assumes *init*: *initCond* Δ **and** *unwind*: *unwindCond* Δ
shows *lrsecure*
unfolding *lrsecure-def2* **proof** *clarify*
fix *s1 tr1 s2 tr2*
assume *3*: *istateO s1 Opt.lvalidFromS s1 tr1 lcompletedFromO s1 tr1*
and *4*: *istateO s2 Opt.lvalidFromS s2 tr2 lcompletedFromO s2 tr2*
and *A34*: *Opt.lA tr1* = *Opt.lA tr2* **and** *O34*: *Opt.lO tr1* ≠ *Opt.lO tr2*
obtain *sv1 sv2* **where**
isv12: *istateV sv1 istateV sv2* **and** *c12*: *corrState sv1 s1 corrState sv2 s2*
and Δ: Δ ∞ ∞ ∞ *s1 s2 Eq sv1 sv2 Eq*
using *init 3 4* **unfolding** *initCond-def* **by** *blast*
have *r*: *reachV sv1 reachV sv2 reachO s1 reachO s2*

by (*auto simp*: *Van.Istate isv12 Opt.Istate 3 4*)
note *all* = *3 4 A34 isv12* Δ *unwind r*
show $\exists$ *sv1 trv1 sv2 trv2*.
istateV sv1 $\wedge$ *istateV sv2* $\wedge$ *corrState sv1 s1* $\wedge$ *corrState sv2 s2* $\wedge$
Van.lvalidFromS sv1 trv1 $\wedge$ *lcompletedFromV sv1 trv1* $\wedge$ *Van.lvalidFromS sv2 trv2* $\wedge$
lcompletedFromV sv2 trv2 $\wedge$ *Van.lS trv1* = *Opt.lS tr1* $\wedge$ *Van.lS trv2* = *Opt.lS tr2* $\wedge$
Van.lA trv1 = *Van.lA trv2* $\wedge$ *Van.lO trv1* $\neq$ *Van.lO trv2*
apply(*rule exI*[*of - sv1*])
apply(*rule exI*[*of - ltrv1* Δ (∞, ∞, *s1*, *tr1*, *s2*, *tr2*, *Eq*, *sv1*, *sv2*, *Eq*)])
apply(*rule exI*[*of - sv2*])
apply(*rule exI*[*of - ltrv2* Δ (∞, ∞, *s1*, *tr1*, *s2*, *tr2*, *Eq*, *sv1*, *sv2*, *Eq*)])
apply(*intro conjI*)
subgoal by *fact* **subgoal by** *fact* **subgoal by** *fact* **subgoal by** *fact*
subgoal apply(*rule lvalidFromS-ltrv1*) **using** *all* **by** *auto*
subgoal apply(*rule lcompletedFrom-ltrv1*) **using** *all* **by** *auto*
subgoal apply(*rule lvalidFromS-ltrv2*) **using** *all* **by** *auto*
subgoal apply(*rule lcompletedFrom-ltrv2*) **using** *all* **by** *auto*
subgoal apply(*rule lS-ltrv1-ltr1*) **using** *all* **by** *auto*
subgoal apply(*rule lS-ltrv2-ltr2*) **using** *all* **by** *auto*
subgoal apply(*rule lA-ltrv1-ltrv2*) **using** *all* **by** *auto*
subgoal using *O34* **apply**− **apply**(*erule contrapos-nn*)
apply(*rule lO-ltrv1-ltrv2*) **using** *all* **by** *auto* **.**
qed

4.4 Compositional unwinding

We allow networks of unwinding relations that unwind into each other, which offer a compositional alternative to monolithic unwinding.

definition *unwindIntoCond* ::
(*enat* $\Rightarrow$ *enat* $\Rightarrow$ *enat* $\Rightarrow$ *'stateO* $\Rightarrow$ *'stateO* $\Rightarrow$ *status* $\Rightarrow$ *'stateV* $\Rightarrow$ *'stateV* $\Rightarrow$ *status* $\Rightarrow$ *bool*) $\Rightarrow$
(*enat* $\Rightarrow$ *enat* $\Rightarrow$ *enat* $\Rightarrow$ *'stateO* $\Rightarrow$ *'stateO* $\Rightarrow$ *status* $\Rightarrow$ *'stateV* $\Rightarrow$ *'stateV* $\Rightarrow$ *status* $\Rightarrow$ *bool*)
$\Rightarrow$ *bool*
where
unwindIntoCond Δ Δ' $\equiv$ $\forall$ *w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 $\wedge$ *reachO s2* $\wedge$ *reachV sv1* $\wedge$ *reachV sv2* $\wedge$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\longrightarrow$
(*finalO s1* $\longleftrightarrow$ *finalO s2*) $\wedge$ (*finalV sv1* $\longleftrightarrow$ *finalO s1*) $\wedge$ (*finalV sv2* $\longleftrightarrow$ *finalO s2*)
$\wedge$
(*statA* = *Eq* $\longrightarrow$ (*isIntO s1* $\longleftrightarrow$ *isIntO s2*))
$\wedge$
(($\exists$ *v*<*w*. *proact* Δ' *v w1 w2 s1 s2 statA sv1 sv2 statO*)
$\vee$
match Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*)

theorem *distrib-unwind-lrsecure*:
assumes *m*: *0* < *m* **and** *nxt*: ⋀*i*. *i* < (*m*::*nat*) ⟹ *nxt i* ⊆ {*0*..<*m*}
and *init*: *initCond* (Δ*s 0*)
and *step*: ⋀*i*. *i* < *m* ⟹
  unwindIntoCond (Δ*s i*) (λ*w w1 w2 s1 s2 statA sv1 sv2 statO*.
    ∃*j* ∈ *nxt i*. Δ*s j w w1 w2 s1 s2 statA sv1 sv2 statO*)
shows *lrsecure*
proof−
  define Δ **where** *D*: Δ ≡ λ*w w1 w2 s1 s2 statA sv1 sv2 statO*. ∃*i* < *m*. Δ*s i w w1 w2 s1 s2 statA sv1 sv2 statO*
  have *i*: *initCond* Δ **using** *init m* **unfolding** *initCond-def D* **by** *meson*
  have *c*: *unwindCond* Δ **unfolding** *unwindCond-def* **apply**(*intro allI impI allI*)
  apply(*subst* (*asm*) *D*) **apply** (*elim exE conjE*)
    subgoal for *w w1 w2 s1 s2 statA sv1 sv2 statO i*
      apply(*frule step*) **unfolding** *unwindIntoCond-def*
       apply(*erule allE*[*of - w*]) **apply**(*erule allE*[*of - w1*]) **apply**(*erule allE*[*of - w2*])
       apply(*erule allE*[*of - s1*]) **apply**(*erule allE*[*of - s2*]) **apply**(*erule allE*[*of - statA*])
      apply(*erule allE*[*of - sv1*]) **apply**(*erule allE*[*of - sv2*]) **apply**(*erule allE*[*of - statO*])
      apply *simp* **apply**(*elim conjE*)
      apply(*erule disjE*)
        subgoal apply(*rule disjI1*)
        subgoal apply(*elim exE conjE*) **subgoal for** *v*
        apply(*rule exI*[*of - v*], *simp*)
        apply(*rule proact-mono*[*of* λ*w w1 w2 s1 s2 statA sv1 sv2 statO*. ∃*j*∈*nxt i*. Δ*s j w w1 w2 s1 s2 statA sv1 sv2 statO*])
          subgoal unfolding *le-fun-def D* **by** *simp* (*meson atLeastLessThan-iff nxt subsetD*)
          subgoal . . . .
        subgoal apply(*rule disjI2*)
        apply(*rule match-mono*[*of* λ*w w1 w2 s1 s2 statA sv1 sv2 statO*. ∃*j*∈*nxt i*. Δ*s j w w1 w2 s1 s2 statA sv1 sv2 statO*])
          subgoal unfolding *le-fun-def D* **by** *simp* (*meson atLeastLessThan-iff nxt subsetD*)
          subgoal . . . .
  show *?thesis* **using** *unwind-lrsecure*[*OF i c*] **.**
qed

lemma *unwindIntoCond-simpleI*:
assumes
 ⋀*w w1 w2 s1 s2 statA sv1 sv2 statO*.
 reachO s1 ⟹ *reachO s2* ⟹ *reachV sv1* ⟹ *reachV sv2* ⟹
 Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
 ⟹
 (*finalO s1* ⟷ *finalO s2*) ∧ (*finalV sv1* ⟷ *finalO s1*) ∧ (*finalV sv2* ⟷ *finalO s2*)

and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
statA = *Eq*
$\Longrightarrow$
isIntO s1 $\longleftrightarrow$ *isIntO s2*
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
match Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
shows *unwindIntoCond* Δ Δ'
using *assms* **unfolding** *unwindIntoCond-def* **by** *auto*

lemma *unwindIntoCond-simpleI2*:
assumes
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
(*finalO s1* $\longleftrightarrow$ *finalO s2*) $\wedge$ (*finalV sv1* $\longleftrightarrow$ *finalO s1*) $\wedge$ (*finalV sv2* $\longleftrightarrow$ *finalO s2*)
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
statA = *Eq*
$\Longrightarrow$
isIntO s1 $\longleftrightarrow$ *isIntO s2*
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
($\exists$ *v*<*w. proact* Δ' *v w1 w2 s1 s2 statA sv1 sv2 statO*)
shows *unwindIntoCond* Δ Δ'
using *assms* **unfolding** *unwindIntoCond-def* **by** *auto*

lemma *unwindIntoCond-simpleIB*:
assumes
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
(*finalO s1* $\longleftrightarrow$ *finalO s2*) $\wedge$ (*finalV sv1* $\longleftrightarrow$ *finalO s1*) $\wedge$ (*finalV sv2* $\longleftrightarrow$ *finalO s2*)
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*

reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
statA = *Eq*
$\Longrightarrow$
isIntO s1 $\longleftrightarrow$ *isIntO s2*
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
($\exists$ *v<w. proact* Δ' *v w1 w2 s1 s2 statA sv1 sv2 statO*) $\vee$ *match* Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
shows *unwindIntoCond* Δ Δ'
using *assms* **unfolding** *unwindIntoCond-def* **by** *auto*

definition *oor* **where**
oor Δ Δ_2 $\equiv$ λ*w w1 w2 s1 s2 statA sv1 sv2 statO.*
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO*

lemma *oorI1*:
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor* Δ Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor-def* **by** *simp*

lemma *oorI2*:
Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor* Δ Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor-def* **by** *simp*

definition *oor3* **where**
oor3 Δ Δ_2 Δ_3 $\equiv$ λ*w w1 w2 s1 s2 statA sv1 sv2 statO.*
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO*

lemma *oor3I1*:
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor3-def* **by** *simp*

lemma *oor3I2*:
Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor3-def* **by** *simp*

lemma *oor3I3*:
Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO*

unfolding *oor3-def* **by** *simp*

definition *oor4* **where**
oor4 Δ Δ_2 Δ_3 Δ_4 $\equiv$ λ*w w1 w2 s1 s2 statA sv1 sv2 statO.*
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$ Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO*

lemma *oor4I1*:
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor4-def* **by** *simp*

lemma *oor4I2*:
Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor4-def* **by** *simp*

lemma *oor4I3*:
Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor4-def* **by** *simp*

lemma *oor4I4*:
Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor4-def* **by** *simp*

definition *oor5* **where**
oor5 Δ Δ_2 Δ_3 Δ_4 Δ_5 $\equiv$ λ*w w1 w2 s1 s2 statA sv1 sv2 statO.*
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$ Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\vee$
Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO*

lemma *oor5I1*:
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *oor5I2*:
Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *oor5I3*:
Δ_3 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *oor5I4*:
Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *oor5I5*:
Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *oor5-def* **by** *simp*

lemma *isIntO-match1*: *isIntO s1* $\Longrightarrow$ *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *match1-def* **by** *auto*

lemma *isIntO-match2*: *isIntO s2* $\Longrightarrow$ *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
unfolding *match2-def* **by** *auto*

lemma *isIntO-match*:
  assumes ‹*isIntO s1*› **and** ‹*isIntO s2*›
      and ‹*match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*›
    shows ‹*match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*›
  unfolding *match-def* **apply** (*intro conjI*)
  subgoal
    using *assms*(*1*) **by** (*rule isIntO-match1*)
  subgoal
    using *assms*(*2*) **by** (*rule isIntO-match2*)
  subgoal
    using *assms*(*3*) **by** *assumption*
  .

lemma *match1-1-oorI1*:
match1-1 Δ *w1 w2 s1 s1′ s2 statA sv1 sv2 statO* $\Longrightarrow$
 match1-1 (*oor* Δ Δ_2) *w1 w2 s1 s1′ s2 statA sv1 sv2 statO*
apply(*rule match1-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match1-1-oorI2*:
match1-1 Δ_2 *w1 w2 s1 s1′ s2 statA sv1 sv2 statO* $\Longrightarrow$
 match1-1 (*oor* Δ Δ_2) *w1 w2 s1 s1′ s2 statA sv1 sv2 statO*
apply(*rule match1-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match1-oorI1*:
match1 Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
 match1 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule match1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match1-oorI2*:
match1 Δ_2 *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match1 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule match1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match2-1-oorI1*:
match2-1 Δ *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\Longrightarrow$
match2-1 (*oor* Δ Δ_2) *w1 w2 s1 s2 s2' statA sv1 sv2 statO*
apply(*rule match2-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match2-1-oorI2*:
match2-1 Δ_2 *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\Longrightarrow$
match2-1 (*oor* Δ Δ_2) *w1 w2 s1 s2 s2' statA sv1 sv2 statO*
apply(*rule match2-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match2-oorI1*:
match2 Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match2 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule match2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match2-oorI2*:
match2 Δ_2 *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match2 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule match2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-oorI1*:
match12 Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match12 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule match12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-oorI2*:
match12 Δ_2 *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match12 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule match12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-1-oorI1*:
match12-1 Δ *w1 w2 s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-1 (*oor* Δ Δ_2) *w1 w2 s1' s2' statA' sv1 sv2 statO*
apply(*rule match12-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-1-oorI2*:
match12-1 Δ_2 *w1 w2 s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-1 (*oor* Δ Δ_2) *w1 w2 s1' s2' statA' sv1 sv2 statO*
apply(*rule match12-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-2-oorI1*:
match12-2 Δ *w1 w2 s2 s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-2 (*oor* Δ Δ_2) *w1 w2 s2 s2' statA' sv1 sv2 statO*
apply(*rule match12-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-2-oorI2*:
match12-2 Δ_2 *w1 w2 s2 s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-2 (*oor* Δ Δ_2) *w1 w2 s2 s2' statA' sv1 sv2 statO*
apply(*rule match12-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-12-oorI1*:
match12-12 Δ *w1 w2 s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-12 (*oor* Δ Δ_2) *w1 w2 s1' s2' statA' sv1 sv2 statO*
apply(*rule match12-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match12-12-oorI2*:
match12-12 Δ_2 *w1 w2 s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-12 (*oor* Δ Δ_2) *w1 w2 s1' s2' statA' sv1 sv2 statO*
apply(*rule match12-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match-oorI1*:
match Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule match-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *match-oorI2*:
match Δ_2 *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule match-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *proact-oorI1*:
proact Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
proact (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule proact-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *proact-oorI2*:
proact Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
proact (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule proact-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-1-oorI1*:
move-1 Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-1 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule move-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-1-oorI2*:
move-1 Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-1 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule move-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-2-oorI1*:
move-2 Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-2 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule move-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-2-oorI2*:
move-2 Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-2 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule move-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-12-oorI1*:
move-12 Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-12 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule move-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

lemma *move-12-oorI2*:
move-12 Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-12 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
apply(*rule move-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

end

context *Relative-Security-Determ*
begin

lemma *match1-1-defD*: *match1-1* Δ *w1 w2 s1 s1'* *s2 statA sv1 sv2 statO* $\longleftrightarrow$

$\neg$ *finalV sv1* $\wedge$ Δ ∞ *w1 w2 s1'* *s2 statA* (*nextO sv1*) *sv2 statO*
unfolding *match1-1-def validTrans-iff-next* **by** *simp*

lemma *match1-12-defD*: *match1-12* Δ *w1 w2 s1 s1'* *s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$
Δ ∞ *w1 w2 s1'* *s2 statA* (*nextO sv1*) (*nextO sv2*) (*sstatO'* *statO sv1 sv2*)
unfolding *match1-12-def validTrans-iff-next* **by** *simp*

lemmas *match1-defsD* = *match1-def match1-1-defD match1-12-defD*

lemma *match2-1-defD*: *match2-1* Δ *w1 w2 s1 s2 s2'* *statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ Δ ∞ *w1 w2 s1 s2'* *statA sv1* (*nextO sv2*) *statO*
unfolding *match2-1-def validTrans-iff-next* **by** *simp*

lemma *match2-12-defD*: *match2-12* Δ *w1 w2 s1 s2 s2′ statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$ Δ ∞ *w1 w2 s1 s2′ statA* (*nextO sv1*) (*nextO sv2*) (*sstatO′ statO sv1 sv2*)
unfolding *match2-12-def validTrans-iff-next* **by** *simp*

lemmas *match2-defsD* = *match2-def match2-1-defD match2-12-defD*

lemma *match12-1-defD*: *match12-1* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ Δ ∞ *w1 w2 s1′ s2′ statA′* (*nextO sv1*) *sv2 statO*
unfolding *match12-1-def validTrans-iff-next* **by** *simp*

lemma *match12-2-defD*: *match12-2* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ Δ ∞ *w1 w2 s1′ s2′ statA′ sv1* (*nextO sv2*) *statO*
unfolding *match12-2-def validTrans-iff-next* **by** *simp*

lemma *match12-12-defD*: *match12-12* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* $\longleftrightarrow$

(*let statO′* = *sstatO′ statO sv1 sv2 in*
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$
(*statA′* = *Diff* $\longrightarrow$ *statO′* = *Diff*) $\wedge$
Δ ∞ *w1 w2 s1′ s2′ statA′* (*nextO sv1*) (*nextO sv2*) *statO′*)
unfolding *match12-12-def validTrans-iff-next* **by** *simp*

lemmas *match12-defsD* = *match12-def match12-1-defD match12-2-defD match12-12-defD*

lemmas *match-deep-defsD* = *match1-defsD match2-defsD match12-defsD*

lemma *move-1-defD*: *move-1* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ Δ *w w1 w2 s1 s2 statA* (*nextO sv1*) *sv2 statO*
unfolding *move-1-def validTrans-iff-next* **by** *simp*

lemma *move-2-defD*: *move-2* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ Δ *w w1 w2 s1 s2 statA sv1* (*nextO sv2*) *statO*
unfolding *move-2-def validTrans-iff-next* **by** *simp*

lemma *move-12-defD*: *move-12* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\longleftrightarrow$
(*let statO′* = *sstatO′ statO sv1 sv2 in*
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$
Δ *w w1 w2 s1 s2 statA* (*nextO sv1*) (*nextO sv2*) *statO′*)
unfolding *move-12-def validTrans-iff-next* **by** *simp*

lemmas *proact-defsD* = *proact-def move-1-defD move-2-defD move-12-defD*

end

end

References

[1] A. P. Brijesh Dongol, Matt Griffin and J. Wright. Relative security: Formally modeling and (dis)proving resilience against semantic optimization vulnerabilities. In *37th IEEE Computer Security Foundations Symposium, CSF 2024*. To appear.