

# Relative Security

Andrei Popescu      Jamie Wright

February 6, 2026

## 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 soundness in the infinitary case is a substantial application of Isabelle’s corecursion and coinduction infrastructure.

## Contents

<b>1</b>	<b>Finitary Relative Security</b>	<b>2</b>
1.1	Finite-trace versions of leakage models and attacker models . . . . .	2
1.2	Locales for increasingly concrete notions of finitary relative security . . . . .	3
<b>2</b>	<b>Relative Security</b>	<b>8</b>
2.1	Leakage models and attacker models . . . . .	8
2.2	Locales for increasingly concrete notions of relative security . . . . .	9
<b>3</b>	<b>Unwinding Proof Method for Finitary Relative Security</b>	<b>13</b>
3.1	The types and operators underlying unwinding: status, matching operators, etc. . . . .	13
3.2	The definition of unwinding . . . . .	18
3.3	The soundness of unwinding . . . . .	19
3.4	Compositional unwinding . . . . .	29
<b>4</b>	<b>Unwinding Proof Method for Relative Security</b>	<b>39</b>
4.1	The types and operators underlying unwinding: status, matching operators, etc. . . . .	40

4.2	The definition of unwinding . . . . .	45
4.3	The soundness of unwinding . . . . .	46
4.4	Compositional unwinding . . . . .	203
<b>5</b>	<b>Relative Security Unwinding Incompleteness example</b>	<b>212</b>
<b>6</b>	<b>Relative Security Unwinding Incompleteness example</b>	<b>218</b>

## 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  $\Rightarrow$  bool and validTrans :: 'state  $\times$  'state  $\Rightarrow$  bool and final ::
'state  $\Rightarrow$  bool
+
fixes leakVia :: 'state list  $\Rightarrow$  'state list  $\Rightarrow$  'leak  $\Rightarrow$  bool
```

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

```
fun leakVia :: 'state list  $\Rightarrow$  'state list  $\Rightarrow$  'secret list  $\times$  'secret list  $\Rightarrow$  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* *leakViaV*  
+  
*Opt*: *Leakage-Mod-fin* *istateO* *validTransO* *finalO* *leakViaO*  
**for** *validTransV* :: 'stateV × 'stateV ⇒ bool  
**and** *istateV* :: 'stateV ⇒ bool **and** *finalV* :: 'stateV ⇒ bool  
**and** *leakViaV* :: 'stateV list ⇒ 'stateV list ⇒ 'leak ⇒ bool  
  
**and** *validTransO* :: 'stateO × 'stateO ⇒ bool  
**and** *istateO* :: 'stateO ⇒ bool **and** *finalO* :: 'stateO ⇒ bool  
**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 ⇒ 'actV list  
**and** *OV* :: 'stateV trace ⇒ 'obsV list  
  
**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 corrState :: 'stateV ⇒ 'stateO ⇒ 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 ⇒ bool and validTrans :: 'state × 'state ⇒ bool and final ::
'state ⇒ bool
+
fixes
  isSec :: 'state ⇒ bool and getSec :: 'state ⇒ 'secret
and
  isInt :: 'state ⇒ bool and getInt :: 'state ⇒ 'act × 'obs
assumes final-not-isInt: ∧s. final s ⇒ ¬ isInt s
and final-not-isSec: ∧s. final s ⇒ ¬ isSec s
begin

```

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

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

**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* ∧ *isIntO s2* → *getActO s1* = *getActO s2*)

→

(∃ *sv1 trv1 sv2 trv2*.

*istateV sv1* ∧ *istateV sv2* ∧ *corrState sv1 s1* ∧ *corrState sv2 s2* ∧

$Van.validFromS\ sv1\ trv1 \wedge completedFromV\ sv1\ trv1 \wedge$   
 $Van.validFromS\ sv2\ trv2 \wedge completedFromV\ sv2\ trv2 \wedge$   
 $Van.S\ trv1 = Opt.S\ tr1 \wedge Van.S\ trv2 = Opt.S\ tr2 \wedge$   
 $Van.A\ trv1 = Van.A\ trv2 \wedge Van.O\ trv1 \neq Van.O\ trv2))$   
**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*  $\equiv$   
 $(isSecV\ trnO = isSecO\ trnA) \wedge (isSecV\ trnO \longrightarrow getSecV\ trnO = getSecO\ trnA)$

**lemma** *eqSec-S-Cons'*:  
 $eqSec\ trnO\ trnA \implies$   
 $(Van.S\ (trnO \# trO') = Opt.S\ (trnA \# trA')) \implies Van.S\ trO' = Opt.S\ trA'$   
**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]*:  
 $eqSec\ trnO\ trnA \implies trO' = [] \longleftrightarrow trA' = [] \implies$   
 $(Van.S\ (trnO \# trO') = Opt.S\ (trnA \# trA')) \longleftrightarrow (Van.S\ trO' = Opt.S\ trA')$   
**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/Trivialia
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 leakVia :: '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 leakVia :: 'state llist  $\Rightarrow$  'state llist  $\Rightarrow$  'secret llist  $\times$  'secret llist  $\Rightarrow$  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 < Leakage-Mod
where leakVia = leakVia
by standard

```

## 2.2 Locales for increasingly concrete notions of relative security

```

locale Relative-Security'' =
  Van: Leakage-Mod istateV validTransV finalV leakViaV
+
  Opt: Leakage-Mod istateO validTransO finalO leakViaO
for validTransV :: 'stateV  $\times$  'stateV  $\Rightarrow$  bool
and istateV :: 'stateV  $\Rightarrow$  bool and finalV :: 'stateV  $\Rightarrow$  bool
and leakViaV :: 'stateV llist  $\Rightarrow$  'stateV llist  $\Rightarrow$  'leak  $\Rightarrow$  bool

and validTransO :: 'stateO  $\times$  'stateO  $\Rightarrow$  bool
and istateO :: 'stateO  $\Rightarrow$  bool and finalO :: 'stateO  $\Rightarrow$  bool
and leakViaO :: 'stateO llist  $\Rightarrow$  'stateO llist  $\Rightarrow$  'leak  $\Rightarrow$  bool

and corrState :: 'stateV  $\Rightarrow$  'stateO  $\Rightarrow$  bool
begin

```

**definition** *lrsecure* :: bool **where**

```

lrsecure  $\equiv$   $\forall$  l s1 tr1 s2 tr2.
  istateO s1  $\wedge$  Opt.linvalidFromS s1 tr1  $\wedge$  Opt.lcompletedFrom s1 tr1  $\wedge$ 
  istateO s2  $\wedge$  Opt.linvalidFromS s2 tr2  $\wedge$  Opt.lcompletedFrom s2 tr2  $\wedge$ 
  leakViaO tr1 tr2 l
   $\longrightarrow$ 
  ( $\exists$  sv1 trv1 sv2 trv2.
    istateV sv1  $\wedge$  istateV sv2  $\wedge$  corrState sv1 s1  $\wedge$  corrState sv2 s2  $\wedge$ 
    Van.linvalidFromS sv1 trv1  $\wedge$  Van.lcompletedFrom sv1 trv1  $\wedge$ 
    Van.linvalidFromS sv2 trv2  $\wedge$  Van.lcompletedFrom sv2 trv2  $\wedge$ 
    leakViaV 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 × '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 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 corrState :: 'stateV ⇒ 'stateO ⇒ bool

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

```

```

context Relative-Security'
begin

```

**lemma** lrsecure-def2:

```

lrsecure ↔
(∀ s1 tr1 s2 tr2.
  istateO s1 ∧ Opt.lvalidFromS s1 tr1 ∧ Opt.lcompletedFrom s1 tr1 ∧
  istateO s2 ∧ Opt.lvalidFromS s2 tr2 ∧ Opt.lcompletedFrom 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.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**  $lA :: 'state \text{ ltrace} \Rightarrow 'act \text{ llist}$  **where**  
 $lA \text{ tr} \equiv \text{lfiltermap } isInt \text{ getAct } (lbutlast \text{ tr})$

**sublocale**  $lA: LfiltermapBL \text{ isInt } \text{getAct } lA$   
**apply** *standard unfolding*  $lA\text{-def}$  ..

**lemma**  $lA: lcompletedFrom \text{ s } \text{tr} \Longrightarrow lA \text{ tr} = \text{lmap } \text{getAct } (\text{lfilter } isInt \text{ tr})$   
**apply**(*cases lfinite tr*)  
**subgoal unfolding**  $lA.lmap\text{-lfilter } lbutlast\text{-def}$   
**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**  $lA.lmap\text{-lfilter } lbutlast\text{-def}$  **by** *auto* .

**definition**  $lO :: 'state \text{ ltrace} \Rightarrow 'obs \text{ llist}$  **where**  
 $lO \text{ tr} \equiv \text{lfiltermap } isInt \text{ getObs } (lbutlast \text{ tr})$

**sublocale**  $lO: LfiltermapBL \text{ isInt } \text{getObs } lO$   
**apply** *standard unfolding*  $lO\text{-def}$  ..

**lemma**  $lO: lcompletedFrom \text{ s } \text{tr} \Longrightarrow lO \text{ tr} = \text{lmap } \text{getObs } (\text{lfilter } isInt \text{ tr})$   
**apply**(*cases lfinite tr*)  
**subgoal unfolding**  $lO.lmap\text{-lfilter } lbutlast\text{-def}$   
**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**  $lO.lmap\text{-lfilter } lbutlast\text{-def}$  **by** *auto* .

**definition**  $lS :: 'state \text{ llist} \Rightarrow 'secret \text{ llist}$  **where**  
 $lS \text{ tr} \equiv \text{lfiltermap } isSec \text{ getSec } (lbutlast \text{ tr})$

**sublocale**  $lS: LfiltermapBL \text{ isSec } \text{getSec } lS$   
**apply** *standard unfolding*  $lS\text{-def}$  ..

**lemma**  $lS: lcompletedFrom \text{ s } \text{tr} \Longrightarrow lS \text{ tr} = \text{lmap } \text{getSec } (\text{lfilter } isSec \text{ tr})$   
**apply**(*cases lfinite tr*)  
**subgoal unfolding**  $lS.lmap\text{-lfilter } lbutlast\text{-def}$   
**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**  $lS.lmap\text{-lfilter } lbutlast\text{-def}$  **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**  $lcompletedFromV :: 'stateV \Rightarrow 'stateV\ list \Rightarrow bool$  **where**  $lcompletedFromV \equiv Van.lcompletedFrom$   
**abbreviation**  $lcompletedFromO :: 'stateO \Rightarrow 'stateO\ list \Rightarrow bool$  **where**  $lcompletedFromO \equiv Opt.lcompletedFrom$

**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 newStat :: status  $\Rightarrow$  bool  $\times$  'a  $\Rightarrow$  bool  $\times$  'a  $\Rightarrow$  status where
  newStat Eq (True,a) (True,a') = (if a = a' then Eq else Diff)
| newStat stat - - = stat
```

```
definition sstatO' statO sv1 sv2 = newStat statO (isIntV sv1, getObsV sv1)
(isIntV sv2, getObsV sv2)
```

```
definition sstatA' statA s1 s2 = newStat 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  $\Delta s1 s1' s2 statA sv1 sv2 statO \equiv$ 
```

```
   $\exists sv1'. validTransV (sv1,sv1') \wedge$ 
```

```
   $\Delta \infty s1' s2 statA sv1' sv2 statO$ 
```

```
definition match1-12  $\Delta s1 s1' s2 statA sv1 sv2 statO \equiv$ 
```

```
   $\exists sv1' sv2'.$ 
```

```
  let statO' = sstatO' statO sv1 sv2 in
```

```
  validTransV (sv1,sv1')  $\wedge$ 
```

```
  validTransV (sv2,sv2')  $\wedge$ 
```

$$\Delta \infty s1' s2 \text{ statA } sv1' sv2' \text{ statO}'$$

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

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

**lemma** *match1-1-mono*:

$\Delta \leq \Delta' \implies \text{match1-1 } \Delta s1 s1' s2 \text{ statA } sv1 sv2 \text{ statO} \implies \text{match1-1 } \Delta' s1 s1' s2$   
 $\text{statA } sv1 sv2 \text{ statO}$

**unfolding** *le-fun-def match1-1-def* by *auto*

**lemma** *match1-12-mono*:

$\Delta \leq \Delta' \implies \text{match1-12 } \Delta s1 s1' s2 \text{ statA } sv1 sv2 \text{ statO} \implies \text{match1-12 } \Delta' s1 s1'$   
 $s2 \text{ statA } sv1 sv2 \text{ statO}$

**unfolding** *le-fun-def match1-12-def* by *fastforce*

**lemma** *match1-mono*:

**assumes**  $\Delta \leq \Delta'$

**shows** *match1*  $\Delta s1 s2 \text{ statA } sv1 sv2 \text{ statO} \implies \text{match1 } \Delta' s1 s2 \text{ statA } sv1 sv2$   
 $\text{statO}$

**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*  $\Delta s1 s2 s2' \text{ statA } sv1 sv2 \text{ statO} \equiv$

$$\exists sv2'. \text{validTransV } (sv2, sv2') \wedge$$

$$\Delta \infty s1 s2' \text{ statA } sv1 sv2' \text{ statO}$$

**definition** *match2-12*  $\Delta s1 s2 s2' \text{ statA } sv1 sv2 \text{ statO} \equiv$

$$\exists sv1' sv2'.$$

$$\text{let } \text{statO}' = \text{sstatO}' \text{ statO } sv1 sv2 \text{ in}$$

$$\text{validTransV } (sv1, sv1') \wedge$$

$$\text{validTransV } (sv2, sv2') \wedge$$

$$\Delta \infty s1 s2' \text{ statA } sv1' sv2' \text{ statO}'$$

**definition** *match2*  $\Delta s1 s2 \text{ statA } sv1 sv2 \text{ statO} \equiv$

$$\neg \text{isIntO } s2 \longrightarrow$$

$$(\forall s2'. \text{validTransO } (s2, s2'))$$

$\longrightarrow$   
 $(\neg \text{isSecO } s2 \wedge \Delta \infty s1 \ s2' \ \text{statA } sv1 \ sv2 \ \text{statO}) \vee$   
 $(\text{eqSec } sv2 \ s2 \wedge \neg \text{isIntV } sv2 \wedge \text{match2-1 } \Delta \ s1 \ s2 \ s2' \ \text{statA } sv1 \ sv2 \ \text{statO}) \vee$   
 $(\neg \text{isSecV } sv1 \wedge \text{eqSec } sv2 \ s2 \wedge \text{Van.eqAct } sv1 \ sv2 \wedge \text{match2-12 } \Delta \ s1 \ s2 \ s2' \ \text{statA } sv1 \ sv2 \ \text{statO}))$

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

**lemma** *match2-1-mono*:

$\Delta \leq \Delta' \implies \text{match2-1 } \Delta \ s1 \ s1' \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO} \implies \text{match2-1 } \Delta' \ s1 \ s1' \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO}$

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

**lemma** *match2-12-mono*:

$\Delta \leq \Delta' \implies \text{match2-12 } \Delta \ s1 \ s1' \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO} \implies \text{match2-12 } \Delta' \ s1 \ s1' \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO}$

**unfolding** *le-fun-def match2-12-def* **by** *fastforce*

**lemma** *match2-mono*:

**assumes**  $\Delta \leq \Delta'$

**shows**  $\text{match2 } \Delta \ s1 \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO} \implies \text{match2 } \Delta' \ s1 \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO}$

**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*  $\Delta \ s1' \ s2' \ \text{statA}' \ sv1 \ sv2 \ \text{statO} \equiv$

$\exists sv1'. \text{validTransV } (sv1, sv1') \wedge$   
 $\Delta \infty s1' \ s2' \ \text{statA}' \ sv1' \ sv2 \ \text{statO}$

**definition** *match12-2*  $\Delta \ s1' \ s2' \ \text{statA}' \ sv1 \ sv2 \ \text{statO} \equiv$

$\exists sv2'. \text{validTransV } (sv2, sv2') \wedge$   
 $\Delta \infty s1' \ s2' \ \text{statA}' \ sv1 \ sv2' \ \text{statO}$

**definition** *match12-12*  $\Delta \ s1' \ s2' \ \text{statA}' \ sv1 \ sv2 \ \text{statO} \equiv$

$\exists sv1' \ sv2'.$   
*let* *statO'* = *sstatO' statO sv1 sv2 in*  
 $\text{validTransV } (sv1, sv1') \wedge$   
 $\text{validTransV } (sv2, sv2') \wedge$   
 $(\text{statA}' = \text{Diff} \longrightarrow \text{statO}' = \text{Diff}) \wedge$   
 $\Delta \infty s1' \ s2' \ \text{statA}' \ sv1' \ sv2' \ \text{statO}'$

**definition** *match12*  $\Delta \ s1 \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO} \equiv$

$\forall s1' \ s2'.$

*let* *statA'* = *sstatA' statA s1 s2 in*

$validTransO (s1, s1') \wedge$   
 $validTransO (s2, s2') \wedge$   
 $Opt.eqAct s1 s2 \wedge$   
 $isIntO s1 \wedge isIntO s2$   
 $\longrightarrow$   
 $(\neg isSecO s1 \wedge \neg isSecO s2 \wedge (statA = statA' \vee statO = Diff)) \wedge \Delta \infty s1' s2'$   
 $statA' sv1 sv2 statO)$   
 $\vee$   
 $(\neg isSecO s2 \wedge eqSec sv1 s1 \wedge \neg isIntV sv1 \wedge$   
 $(statA = statA' \vee statO = Diff) \wedge$   
 $match12-1 \Delta s1' s2' statA' sv1 sv2 statO)$   
 $\vee$   
 $(\neg isSecO s1 \wedge eqSec sv2 s2 \wedge \neg isIntV sv2 \wedge$   
 $(statA = statA' \vee statO = Diff) \wedge$   
 $match12-2 \Delta s1' s2' statA' sv1 sv2 statO)$   
 $\vee$   
 $(eqSec sv1 s1 \wedge eqSec sv2 s2 \wedge Van.eqAct sv1 sv2 \wedge$   
 $match12-12 \Delta s1' s2' statA' sv1 sv2 statO)$

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

**lemma**  $match12-simpleI$ :

**assumes**  $\bigwedge s1' s2' statA'$ .

$statA' = sstatA' statA s1 s2 \implies$

$validTransO (s1, s1') \implies$

$validTransO (s2, s2') \implies$

$Opt.eqAct s1 s2 \implies$

$(\neg isSecO s1 \wedge \neg isSecO s2 \wedge (statA = statA' \vee statO = Diff)) \wedge \Delta \infty s1' s2'$   
 $statA' sv1 sv2 statO)$

$\vee$

$(eqSec sv1 s1 \wedge eqSec sv2 s2 \wedge Van.eqAct sv1 sv2 \wedge$

$match12-12 \Delta s1' s2' statA' sv1 sv2 statO)$

**shows**  $match12 \Delta s1 s2 statA sv1 sv2 statO$

**using**  $assms unfolding match12-def Let-def$  **by**  $blast$

**lemma**  $match12-1-mono$ :

$\Delta \leq \Delta' \implies match12-1 \Delta s1' s2' statA' sv1 sv2 statO \implies match12-1 \Delta' s1' s2'$   
 $statA' sv1 sv2 statO$

**unfolding**  $le-fun-def match12-1-def$  **by**  $auto$

**lemma**  $match12-2-mono$ :

$\Delta \leq \Delta' \implies match12-2 \Delta s1 s2' statA' sv1 sv2 statO \implies match12-2 \Delta' s1 s2'$   
 $statA' sv1 sv2 statO$

**unfolding**  $le-fun-def match12-2-def$  **by**  $auto$

**lemma**  $match12-12-mono$ :

$\Delta \leq \Delta' \implies match12-12 \Delta s1' s2' statA' sv1 sv2 statO \implies match12-12 \Delta' s1'$   
 $s2' statA' sv1 sv2 statO$

**unfolding** *le-fun-def match12-12-def* by *fastforce*

**lemma** *match12-mono*:

**assumes**  $\Delta \leq \Delta'$

**shows**  $\text{match12 } \Delta \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \implies \text{match12 } \Delta' \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$

**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** *react*  $\Delta \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \equiv$

*match1*  $\Delta \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$

$\wedge$

*match2*  $\Delta \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$

$\wedge$

*match12*  $\Delta \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$

**lemmas** *react-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**  $\text{react } \Delta \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \implies \text{react } \Delta' \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$

**unfolding** *react-def* **using** *match1-mono[OF assms]* *match2-mono[OF assms]* *match12-mono[OF assms]* **by** *auto*

**definition** *move-1*  $\Delta \ w \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \equiv$

$\exists sv1'. \text{validTransV } (sv1, sv1') \wedge$

$\Delta \ w \ s1 \ s2 \ \text{statA} \ sv1' \ sv2 \ \text{statO}$

**definition** *move-2*  $\Delta \ w \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \equiv$

$\exists sv2'. \text{validTransV } (sv2, sv2') \wedge$

$\Delta \ w \ s1 \ s2 \ \text{statA} \ sv1 \ sv2' \ \text{statO}$

**definition** *move-12*  $\Delta \ w \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \equiv$

$\exists sv1' \ sv2'.$

*let* *statO'* = *sstatO' statO sv1 sv2 in*

*validTransV*  $(sv1, sv1') \wedge \text{validTransV } (sv2, sv2') \wedge$

$\Delta \ w \ s1 \ s2 \ \text{statA} \ sv1' \ sv2' \ \text{statO}'$

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

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

**lemma** *move-1-mono*:

$\Delta \leq \Delta' \implies move-1 \Delta meas s1 s2 statA sv1 sv2 statO \implies move-1 \Delta' meas s1 s2 statA sv1 sv2 statO$

**unfolding** *le-fun-def move-1-def* **by** *auto*

**lemma** *move-2-mono*:

$\Delta \leq \Delta' \implies move-2 \Delta meas s1 s2 statA sv1 sv2 statO \implies move-2 \Delta' meas s1 s2 statA sv1 sv2 statO$

**unfolding** *le-fun-def move-2-def* **by** *auto*

**lemma** *move-12-mono*:

$\Delta \leq \Delta' \implies move-12 \Delta meas s1 s2 statA sv1 sv2 statO \implies move-12 \Delta' meas s1 s2 statA sv1 sv2 statO$

**unfolding** *le-fun-def move-12-def* **by** *fastforce*

**lemma** *proact-mono*:

**assumes**  $\Delta \leq \Delta'$

**shows** *proact*  $\Delta meas s1 s2 statA sv1 sv2 statO \implies proact \Delta' meas s1 s2 statA sv1 sv2 statO$

**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* ::

$(enat \implies 'stateO \implies 'stateO \implies status \implies 'stateV \implies 'stateV \implies status \implies bool) \implies bool$

**where**

*unwindCond*  $\Delta \equiv \forall w s1 s2 statA sv1 sv2 statO.$

$reachO s1 \wedge reachO s2 \wedge reachV sv1 \wedge reachV sv2 \wedge$

$\Delta w s1 s2 statA sv1 sv2 statO$

$\longrightarrow$

$(finalO s1 \longleftrightarrow finalO s2) \wedge (finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO s2)$

$\wedge$

$(statA = Eq \longrightarrow (isIntO s1 \longleftrightarrow isIntO s2))$

$\wedge$

$((\exists v < w. proact \Delta v s1 s2 statA sv1 sv2 statO)$

$\vee$

$react \Delta s1 s2 statA sv1 sv2 statO$   
 $)$

**lemma** *unwindCond-simpleI*:

**assumes**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$

$reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$

$\Delta w s1 s2 statA sv1 sv2 statO$

$\implies$

$(finalO s1 \longleftrightarrow finalO s2) \wedge (finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO s2)$

**and**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$

$reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$

$\Delta w s1 s2 statA sv1 sv2 statO \implies statA = Eq$

$\implies$

$isIntO s1 \longleftrightarrow isIntO s2$

**and**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$

$reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$

$\Delta w s1 s2 statA sv1 sv2 statO$

$\implies$

$react \Delta s1 s2 statA sv1 sv2 statO$

**shows** *unwindCond*  $\Delta$

**using** *assms unfolding unwindCond-def by auto*

### 3.3 The soundness of unwinding

**definition**  $\psi s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2 \equiv$

$trv1 \neq [] \wedge trv2 \neq [] \wedge$

$Van.validFromS sv1 trv1 \wedge$

$Van.validFromS sv2 trv2 \wedge$

$(finalV (lastt sv1 trv1) \longleftrightarrow finalO (lastt s1 tr1)) \wedge (finalV (lastt sv2 trv2) \longleftrightarrow finalO (lastt s2 tr2)) \wedge$

$Van.S trv1 = Opt.S tr1 \wedge Van.S trv2 = Opt.S tr2 \wedge$

$Van.A trv1 = Van.A trv2 \wedge$

$(statO = Eq \wedge Opt.O tr1 \neq Opt.O tr2 \longrightarrow Van.O trv1 \neq Van.O trv2)$

**lemma**  *$\psi$ -completedFrom*:  $completedFromO s1 tr1 \implies completedFromO s2 tr2 \implies$

$\psi s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2$

$\implies completedFromV sv1 trv1 \wedge completedFromV sv2 trv2$

**unfolding**  *$\psi$ -def Opt.completedFrom-def Van.completedFrom-def lastt-def*  
**by** *presburger*

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

**lemma** *rsecure-strong*:

**assumes**

$\bigwedge s1 \ tr1 \ s2 \ tr2.$

$\text{istateO } s1 \wedge \text{Opt.validFromS } s1 \ tr1 \wedge \text{completedFromO } s1 \ tr1 \wedge$

$\text{istateO } s2 \wedge \text{Opt.validFromS } s2 \ tr2 \wedge \text{completedFromO } s2 \ tr2 \wedge$

$\text{Opt.A } tr1 = \text{Opt.A } tr2 \wedge (\text{isIntO } s1 \wedge \text{isIntO } s2 \implies \text{getActO } s1 = \text{getActO}$

$s2)$

$\implies$

$\exists sv1 \ trv1 \ sv2 \ trv2.$

$\text{istateV } sv1 \wedge \text{istateV } sv2 \wedge \text{corrState } sv1 \ s1 \wedge \text{corrState } sv2 \ s2 \wedge$

$\psi \ s1 \ tr1 \ s2 \ tr2 \ \text{Eq } sv1 \ trv1 \ sv2 \ trv2$

**shows** *rsecure*

**unfolding** *rsecure-def3* **apply** *clarify*

**subgoal for**  $s1 \ tr1 \ s2 \ tr2$

**using** *assms*[*of*  $s1 \ tr1 \ s2 \ tr2$ ]

**using**  $\psi$ -*completedFrom*  $\psi$ -*def* *completedFromO-lastt* **by** (*metis* (*full-types*))

.

**proposition** *unwindCond-ex- $\psi$* :

**assumes** *unwind*: *unwindCond*  $\Delta$

**and**  $\Delta$ :  $\Delta \ w \ s1 \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO}$  **and** *stat*:  $(\text{statA} = \text{Diff} \implies \text{statO} = \text{Diff})$

**and** *v*:  $\text{Opt.validFromS } s1 \ tr1 \ \text{Opt.completedFrom } s1 \ tr1 \ \text{Opt.validFromS } s2 \ tr2$   
 $\text{Opt.completedFrom } s2 \ tr2$

**and** *tr14*:  $\text{Opt.A } tr1 = \text{Opt.A } tr2$

**and** *r*:  $\text{reachO } s1 \ \text{reachO } s2 \ \text{reachV } sv1 \ \text{reachV } sv2$

**shows**  $\exists trv1 \ trv2. \ \psi \ s1 \ tr1 \ s2 \ tr2 \ \text{statO } sv1 \ trv1 \ sv2 \ trv2$

**using** *assms*(2-)

**proof**(*induction*  $\text{length } tr1 + \text{length } tr2 \ w$

*arbitrary*:  $s1 \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO } tr1 \ tr2$  *rule*: *less2-induct'*)

**case** (*less*  $w \ tr1 \ tr2 \ s1 \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO}$ )

**note** *ok* =  $\langle \text{statA} = \text{Diff} \implies \text{statO} = \text{Diff} \rangle$

**note**  $\Delta$  =  $\langle \Delta \ w \ s1 \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO} \rangle$

**note** *A34* =  $\langle \text{Opt.A } tr1 = \text{Opt.A } tr2 \rangle$

**note** *r34* = *less.prem*s(8,9) **note** *r12* = *less.prem*s(10,11)

**note**  $r = r34 \ r12$

**note**  $r3 = r34(1)$  **note**  $r4 = r34(2)$  **note**  $r1 = r12(1)$  **note**  $r2 = r12(2)$

**have** *i34*:  $\text{statA} = \text{Eq} \implies \text{isIntO } s1 = \text{isIntO } s2$

**and** *f34*:  $\text{finalO } s1 = \text{finalO } s2 \wedge \text{finalV } sv1 = \text{finalO } s1 \wedge \text{finalV } sv2 = \text{finalO}$   
 $s2$

**using**  $\Delta$  *unwind*[*unfolded* *unwindCond-def*] *r* **by** *auto*

```

have proact-match: ( $\exists v < w. \text{proact } \Delta v s1 s2 \text{ statA } sv1 sv2 \text{ statO}$ )  $\vee$  react  $\Delta s1$ 
s2 statA sv1 sv2 statO
  using  $\Delta$  unwind[unfolded unwindCond-def] r by auto
show ?case using proact-match proof safe
  fix v assume v:  $v < w$ 
  assume proact  $\Delta v s1 s2 \text{ statA } sv1 sv2 \text{ statO}$ 
  thus ?thesis unfolding proact-def proof safe
  assume sv1:  $\neg \text{isSecV } sv1 \neg \text{isIntV } sv1$  and move-1  $\Delta v s1 s2 \text{ statA } sv1 sv2$ 
statO
  then obtain sv1'
  where 0: validTransV (sv1, sv1')
  and  $\Delta$ :  $\Delta v s1 s2 \text{ statA } sv1' sv2 \text{ statO}$ 
  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  $\psi$ :  $\psi s1 tr1 s2 tr2 \text{ statO } sv1' trv1 sv2 trv2$ 
  using less(2)[OF v, of tr1 tr2 s1 s2 statA sv1' sv2 statO, simplified, OF  $\Delta$ 
ok - - - - r34 r1' r2]
  using A34 less.prem(3-6) by blast
  show ?thesis apply(rule exI[of - sv1 # trv1]) apply(rule exI[of - trv2])
  using  $\psi$  ok 0 sv1 unfolding  $\psi$ -def Van.completedFrom-def by auto
next
  assume sv2:  $\neg \text{isSecV } sv2 \neg \text{isIntV } sv2$  and move-2  $\Delta v s1 s2 \text{ statA } sv1 sv2$ 
statO
  then obtain sv2'
  where 0: validTransV (sv2, sv2')
  and  $\Delta$ :  $\Delta v s1 s2 \text{ statA } sv1 sv2' \text{ statO}$ 
  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  $\psi$ :  $\psi s1 tr1 s2 tr2 \text{ statO } sv1 trv1 sv2' trv2$ 
  using less(2)[OF v, of tr1 tr2 s1 s2 statA sv1 sv2' statO, simplified, OF  $\Delta$ 
ok - - - - r34 r1 r2']
  using A34 less.prem(3-6) by blast
  show ?thesis apply(rule exI[of - trv1]) apply(rule exI[of - sv2 # trv2])
  using  $\psi$  ok 0 sv2 unfolding  $\psi$ -def Van.completedFrom-def by auto
next
  assume sv12:  $\neg \text{isSecV } sv1 \neg \text{isSecV } sv2 \text{ Van.eqAct } sv1 sv2$ 
  and move-12  $\Delta v s1 s2 \text{ statA } sv1 sv2 \text{ statO}$ 
  then obtain sv1' sv2' statO'
  where 0: statO' = sstatO' statO sv1 sv2
  validTransV (sv1, sv1')  $\neg \text{isSecV } sv1$ 
  validTransV (sv2, sv2')  $\neg \text{isSecV } sv2$ 
  Van.eqAct sv1 sv2
  and  $\Delta$ :  $\Delta v s1 s2 \text{ statA } sv1' sv2' \text{ statO}'$ 
  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  $\longrightarrow$  statO' = Diff using ok 0 unfolding sstatO'-def
by (cases statO, auto)
  obtain trv1 trv2 where  $\psi$ :  $\psi s1 tr1 s2 tr2 \text{ statO}' sv1' trv1 sv2' trv2$ 

```

```

    using less(2)[OF v, of tr1 tr2 s1 s2 statA sv1' sv2' statO', simplified, OF Δ
ok' - - - - r34 r12']
    using A34 less.prem(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.prem(4)
    less.prem(6) by auto
qed
next
assume m: react 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.prem(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.prem(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.prem(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.prem(5) less.prem(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
   $trn4$ :  $validTransO (s2, s2') \vee (s2 = s2' \wedge tr2' = [])$  and
   $tr2'$ :  $Opt.validFromS s2' tr2'$  using  $\langle Opt.validFromS s2 tr2 \rangle$ 
unfolding  $tr2 s2_4$  using  $Opt.validFromS-Cons-iff$  by auto
have  $r3_4'$ :  $reachO s1' reachO s2'$ 
using  $r3 trn3 r_4 trn4$  by  $(metis Opt.reach.Step fst-conv snd-conv)+$ 
note  $r3' = r3_4'(1)$  note  $r_4' = r3_4'(2)$ 
define  $statA'$  where  $statA'$ :  $statA' = sstatA' statA s1 s2$ 
have  $\neg isIntO s1 \vee \neg isIntO s2 \vee (isIntO s1 \wedge isIntO s2)$ 
by auto
thus ?thesis
proof safe
  assume  $isAO3$ :  $\neg isIntO s1$ 
  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  $m$ :  $match1 \Delta s1 s2 statA sv1 sv2 statO$  using  $m$  unfolding react-def
by auto
  have  $(\neg isSecO s1 \wedge \Delta \infty s1' s2 statA sv1 sv2 statO) \vee$ 
     $(eqSec sv1 s1 \wedge \neg isIntV sv1 \wedge match1-1 \Delta s1 s1' s2 statA sv1 sv2$ 
 $statO) \vee$ 
     $(eqSec sv1 s1 \wedge \neg isSecV sv2 \wedge Van.eqAct sv1 sv2 \wedge match1-12 \Delta s1$ 
 $s1' s2 statA sv1 sv2 statO)$ 
  using  $m isAO3 trn3 ok$  unfolding match1-def by auto
  thus ?thesis
proof safe
  assume  $\neg isSecO s1$  and  $\Delta$ :  $\Delta \infty s1' s2 statA sv1 sv2 statO$ 
  hence  $S3$ :  $Opt.S tr1' = Opt.S tr1$  unfolding  $tr1$  by auto
  obtain  $trv1 trv2$  where  $\psi$ :  $\psi s1 tr1' s2 tr2 statO sv1 trv1 sv2 trv2$ 
  using  $less(1)[of tr1' tr2, OF - \Delta - - - - - r3' r_4 r12, unfolded O33',$ 
simplified]
  using  $less.premis tr1' ok A34' f3 f4$  unfolding  $tr1 Opt.completedFrom-def$ 
  by  $(auto split: if-splits simp: \psi-def lastt-def)$ 
  show ?thesis apply $(rule exI[of - trv1])$  apply $(rule exI[of - trv2])$ 
  using  $\psi O33' S3$  unfolding  $\psi-def$ 
  using completedFromO-lastt less.premis(4)
  by  $(auto simp add: tr1 tr1'NE)$ 
next
  assume  $trn13$ :  $eqSec sv1 s1$  and
   $Atrn1$ :  $\neg isIntV sv1$  and  $match1-1 \Delta s1 s1' s2 statA sv1 sv2 statO$ 
  then obtain  $sv1'$  where
   $trn1$ :  $validTransV (sv1, sv1')$  and
   $\Delta$ :  $\Delta \infty s1' s2 statA sv1' sv2 statO$ 
  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  $\psi$ :  $\psi s1 tr1' s2 tr2 statO sv1' trv1 sv2 trv2$ 
  using  $less(1)[of tr1' tr2, OF - \Delta - - - - - r3' r_4 r1' r2, unfolded O33',$ 

```

*simplified*]

**using** *less.prem*s  $tr1'$  *ok*  $A34'$   $f3$   $f4$  **unfolding**  $tr1$   $tr2$  *Opt.comple*tedFrom-def

**by** (*auto simp*:  $\psi$ -def lastt-def split: if-splits)

**show** ?thesis **apply**(rule *exI*[of -  $sv1$  #  $trv1$ ]) **apply**(rule *exI*[of -  $trv2$ ])

**using**  $\psi$   $O33'$  **unfolding**  $tr1$   $tr2$  *Van.comple*tedFrom-def

**using** *Van.validFromS-Cons*  $trn1$   $tr1'$  NE  $tr2'$  NE

**using** *isAO3 ok Atrn1 eqSec-S-Cons*  $trn13$

**unfolding**  $\psi$ -def **using** *completedFromO-lastt less.prem*s(4)  $tr1$  **by** *auto*

**next**

**assume**  $sv2$ :  $\neg$  *isSecV*  $sv2$  **and**  $trn13$ : *eqSec*  $sv1$   $s1$  **and**

$Atrn12$ : *Van.eqAct*  $sv1$   $sv2$  **and**  $match1-12$   $\Delta$   $s1$   $s1'$   $s2$  *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**

$\Delta$ :  $\Delta \infty$   $s1'$   $s2$  *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**  $\psi$ :  $\psi$   $s1'$   $tr1'$   $s2$   $tr2$  *statO'*  $sv1'$   $trv1$   $sv2'$   $trv2$

**using** *less(1)*[of  $tr1'$   $tr2$ , *OF* -  $\Delta$  - - - - -  $r3'$   $r4$   $r12'$ , *unfolded*  $O33'$ ,

*simplified*]

**using** *less.prem*s  $tr1'$  *ok*  $A34'$   $f3$   $f4$  **unfolding**  $tr1$   $tr2$  *Opt.comple*tedFrom-def

*statO'* *sstatO'*-def

**by** *auto presburger*+

**show** ?thesis **apply**(rule *exI*[of -  $sv1$  #  $trv1$ ]) **apply**(rule *exI*[of -  $sv2$  #

$trv2$ ])

**using**  $\psi$   $O33'$   $tr1'$  NE  $tr2'$  NE  $sv2$

**using** *Van.validFromS-Cons*  $trn1$   $trn2$

**using** *isAO3 ok Atrn12 eqSec-S-Cons*  $trn13$   $f3$   $f34$   $s13$

**unfolding**  $\psi$ -def  $tr1$  *Van.comple*tedFrom-def *Van.eqAct-def* *statO'* *sstatO'*-def

**using** *Van.A.Cons-unfold*  $tr1'$   $trn3$  **by** *auto*

**qed**

**next**

**assume**  $isAO4$ :  $\neg$  *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**  $m$ :  $match2$   $\Delta$   $s1$   $s2$  *statA*  $sv1$   $sv2$  *statO* **using**  $m$  **unfolding** *react-def*

**by** *auto*

**have** ( $\neg$  *isSecO*  $s2$   $\wedge$   $\Delta \infty$   $s1$   $s2'$  *statA*  $sv1$   $sv2$  *statO*)  $\vee$

(*eqSec*  $sv2$   $s2$   $\wedge$   $\neg$  *isIntV*  $sv2$   $\wedge$   $match2-1$   $\Delta$   $s1$   $s2$   $s2'$  *statA*  $sv1$   $sv2$

*statO*)  $\vee$

( $\neg$  *isSecV*  $sv1$   $\wedge$  *eqSec*  $sv2$   $s2$   $\wedge$  *Van.eqAct*  $sv1$   $sv2$   $\wedge$   $match2-12$   $\Delta$   $s1$

$s2$   $s2'$  *statA*  $sv1$   $sv2$  *statO*)

**using**  $m$   $isAO4$   $trn4$  *ok*  $tr2'$  NE **unfolding**  $match2$ -def **by** *auto*

```

thus ?thesis
proof safe
  assume  $\neg$  isSecO s2 and  $\Delta: \Delta \infty s1 s2' statA sv1 sv2 statO$ 
  hence  $S_4: Opt.S tr2' = Opt.S tr2$  unfolding tr2 by auto
  obtain trv1 trv2 where  $\psi: \psi s1 tr1 s2' tr2' statO sv1 trv1 sv2 trv2$ 
  using less(1)[of tr1 tr2', OF -  $\Delta$  - - - - - r3 r4', simplified]
  using less.premis 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  $\psi$  O44' S4 unfolding  $\psi$ -def
  using completedFromO-lastt less.premis(6)
  unfolding Opt.completedFrom-def using tr2 tr2'NE by auto
next
  assume trn24: eqSec sv2 s2 and
  Atrn2:  $\neg$  isIntV sv2 and match2-1  $\Delta s1 s2 s2' statA sv1 sv2 statO$ 
  then obtain sv2' where trn2: validTransV (sv2,sv2') and
   $\Delta: \Delta \infty s1 s2' statA sv1 sv2' statO$ 
  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  $\psi: \psi s1 tr1 s2' tr2' statO sv1 trv1 sv2' trv2$ 
  using less(1)[of tr1 tr2', OF -  $\Delta$  - - - - - r3 r4' r1 r2', simplified]
  using less.premis 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  $\psi$  tr1'NE tr2'NE
  using Van.validFromS-Cons trn2
  using isAO4 ok Atrn2 eqSec-S-Cons trn24
  unfolding  $\psi$ -def tr1 tr2 s13 s24 Van.completedFrom-def lastt-def by auto
next
  assume sv1:  $\neg$  isSecV sv1 and trn24: eqSec sv2 s2 and
  Atrn12: Van.eqAct sv1 sv2 and match2-12  $\Delta s1 s2 s2' statA sv1 sv2$ 
statO
  then obtain sv1' sv2' statO' where
  statO': statO' = sstatO' statO sv1 sv2 and
  trn1: validTransV (sv1,sv1') and
  trn2: validTransV (sv2,sv2') and
   $\Delta: \Delta \infty s1 s2' statA sv1' sv2' statO'$ 
  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  $\psi: \psi s1 tr1 s2' tr2' statO' sv1' trv1 sv2' trv2$ 
  using less(1)[of tr1 tr2', OF -  $\Delta$  - - - - - r3 r4' r12', simplified]
  using less.premis 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  $\psi$  O44' tr1'NE tr2'NE sv1

```

```

using Van.validFromS-Cons trn1 trn2
using isAO4 ok Atrn12 eqSec-S-Cons trn24
unfolding  $\psi$ -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  $\Delta$  s1 s2 statA sv1 sv2 statO using m unfolding statA'
react-def by auto
have trn34: getObsO s1 = getObsO s2  $\vee$  statA' = Diff
using isAO34 unfolding statA' sstatA'-def by (cases statA,auto)
have ( $\neg$  isSecO s1  $\wedge$   $\neg$  isSecO s2  $\wedge$  (statA = statA'  $\vee$  statO = Diff)  $\wedge$   $\Delta$ 
 $\infty$  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  $\Delta$  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  $\Delta$  s1' s2' statA' sv1 sv2 statO)
 $\vee$ 
(eqSec sv1 s1  $\wedge$  eqSec sv2 s2  $\wedge$  Van.eqAct sv1 sv2  $\wedge$ 
match12-12  $\Delta$  s1' s2' statA' sv1 sv2 statO)
(is ?K1  $\vee$  ?K2  $\vee$  ?K3  $\vee$  ?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  $\Delta$ :  $\Delta \infty$  s1' s2' statA' sv1 sv2 statO by simp
have ok': (statA' = Diff  $\longrightarrow$  statO = Diff)
using ok K1 unfolding statA' using isAO34 by auto
obtain trv1 trv2 where  $\psi$ :  $\psi$  s1' tr1' s2' tr2' statO sv1 trv1 sv2 trv2
using less(1)[of tr1' tr2', OF -  $\Delta$  - - - - - r34' r12, simplified]
using less.premis 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  $\psi$  trn34 O33' O44' K1 ok unfolding  $\psi$ -def tr1 tr2
using completedFromO-lastt less.premis(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.premis 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.premis 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  $\longrightarrow$  statO' = Diff and  $\Delta: \Delta \infty s1' s2' statA'$ 
    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  $\psi: \psi s1' tr1' s2' tr2' statO' sv1' trv1 sv2' trv2$ 
    using less(1)[of tr1' tr2', OF -  $\Delta$  - - - - - r34' r12', simplified]
    using less.premis 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  $\psi$  O33' O44' isAO34 tr1'NE tr2'NE unfolding  $\psi$ -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.premis(4) less.premis(6) list.inject
    s13
    s24 status.simps(1) tr1 tr1' tr2 tr2' trn3 trn4 newStat.simps(1) new-
    Stat.simps(3))
    qed
    qed
    qed
    qed
    qed

```

```

theorem unwind-rsecure:
  assumes init: initCond  $\Delta$ 
  and unwind: unwindCond  $\Delta$ 
  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  $\langle istateO s1 \wedge istateO s2 \rangle$ ] exE conjE)
  subgoal by clarify
  subgoal for sv1 sv2
  using unwind apply (drule-tac unwindCond-ex- $\psi$ , 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* ::

$(\text{enat} \Rightarrow 'stateO \Rightarrow 'stateO \Rightarrow \text{status} \Rightarrow 'stateV \Rightarrow 'stateV \Rightarrow \text{status} \Rightarrow \text{bool}) \Rightarrow$   
 $(\text{enat} \Rightarrow 'stateO \Rightarrow 'stateO \Rightarrow \text{status} \Rightarrow 'stateV \Rightarrow 'stateV \Rightarrow \text{status} \Rightarrow \text{bool})$   
 $\Rightarrow \text{bool}$

**where**

*unwindIntoCond*  $\Delta \Delta' \equiv \forall w s1 s2 \text{statA} sv1 sv2 \text{statO}.$

*reachO*  $s1 \wedge \text{reachO } s2 \wedge \text{reachV } sv1 \wedge \text{reachV } sv2 \wedge$

$\Delta w s1 s2 \text{statA} sv1 sv2 \text{statO} \longrightarrow$

$(\text{finalO } s1 \longleftrightarrow \text{finalO } s2) \wedge (\text{finalV } sv1 \longleftrightarrow \text{finalO } s1) \wedge (\text{finalV } sv2 \longleftrightarrow \text{finalO } s2)$

$\wedge$

$(\text{statA} = \text{Eq} \longrightarrow (\text{isIntO } s1 \longleftrightarrow \text{isIntO } s2))$

$\wedge$

$((\exists v < w. \text{proact } \Delta' v s1 s2 \text{statA} sv1 sv2 \text{statO})$

$\vee$

$\text{react } \Delta' s1 s2 \text{statA} sv1 sv2 \text{statO})$

**lemma** *unwindIntoCond-simpleI*:

**assumes**

$\bigwedge w s1 s2 \text{statA} sv1 sv2 \text{statO}.$

$\text{reachO } s1 \Longrightarrow \text{reachO } s2 \Longrightarrow \text{reachV } sv1 \Longrightarrow \text{reachV } sv2 \Longrightarrow$

$\Delta w s1 s2 \text{statA} sv1 sv2 \text{statO}$

$\Longrightarrow$

$(\text{finalO } s1 \longleftrightarrow \text{finalO } s2) \wedge (\text{finalV } sv1 \longleftrightarrow \text{finalO } s1) \wedge (\text{finalV } sv2 \longleftrightarrow \text{finalO } s2)$

**and**

$\bigwedge w s1 s2 \text{statA} sv1 sv2 \text{statO}.$

$\text{reachO } s1 \Longrightarrow \text{reachO } s2 \Longrightarrow \text{reachV } sv1 \Longrightarrow \text{reachV } sv2 \Longrightarrow$

$\Delta w s1 s2 \text{statA} sv1 sv2 \text{statO} \Longrightarrow$

$\text{statA} = \text{Eq}$

$\Longrightarrow$

$\text{isIntO } s1 \longleftrightarrow \text{isIntO } s2$

$\bigwedge w s1 s2 \text{statA} sv1 sv2 \text{statO}.$

$\text{reachO } s1 \Longrightarrow \text{reachO } s2 \Longrightarrow \text{reachV } sv1 \Longrightarrow \text{reachV } sv2 \Longrightarrow$

$\Delta w s1 s2 \text{statA} sv1 sv2 \text{statO}$

$\Longrightarrow$

$\text{react } \Delta' s1 s2 \text{statA} sv1 sv2 \text{statO}$

**shows** *unwindIntoCond*  $\Delta \Delta'$   
**using** *assms unfolding unwindIntoCond-def by auto*

**lemma** *unwindIntoCond-simpleI2*:

**assumes**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w s1 s2 statA sv1 sv2 statO$   
 $\implies$   
 $(finalO s1 \longleftrightarrow finalO s2) \wedge (finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO s2)$

**and**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w s1 s2 statA sv1 sv2 statO \implies$   
 $statA = Eq$   
 $\implies$   
 $isIntO s1 \longleftrightarrow isIntO s2$

**and**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w s1 s2 statA sv1 sv2 statO$   
 $\implies$   
 $(\exists v < w. proact \Delta' v s1 s2 statA sv1 sv2 statO)$

**shows** *unwindIntoCond*  $\Delta \Delta'$

**using** *assms unfolding unwindIntoCond-def by auto*

**lemma** *unwindIntoCond-simpleIB*:

**assumes**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w s1 s2 statA sv1 sv2 statO$   
 $\implies$   
 $(finalO s1 \longleftrightarrow finalO s2) \wedge (finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO s2)$

**and**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w s1 s2 statA sv1 sv2 statO \implies$   
 $statA = Eq$   
 $\implies$   
 $isIntO s1 \longleftrightarrow isIntO s2$

**and**

$\bigwedge w s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w s1 s2 statA sv1 sv2 statO$   
 $\implies$   
 $(\exists v < w. proact \Delta' v s1 s2 statA sv1 sv2 statO) \vee react \Delta' s1 s2 statA sv1 sv2 statO$

**shows** *unwindIntoCond*  $\Delta \Delta'$   
**using** *assms unfolding unwindIntoCond-def* **by** *auto*

**theorem** *distrib-unwind-rsecure*:  
**assumes**  $m: 0 < m$  **and**  $\text{next}: \bigwedge i. i < (m::\text{nat}) \implies \text{next } i \subseteq \{0..<m\}$   
**and** *init*: *initCond* ( $\Delta s 0$ )  
**and** *step*:  $\bigwedge i. i < m \implies$   
 $\text{unwindIntoCond } (\Delta s i) (\lambda w s1 s2 \text{ statA } sv1 sv2 \text{ statO}.$   
 $\quad \exists j \in \text{next } i. \Delta s j w s1 s2 \text{ statA } sv1 sv2 \text{ statO})$

**shows** *rsecure*  
**proof** –  
**define**  $\Delta$  **where**  $D: \Delta \equiv \lambda w s1 s2 \text{ statA } sv1 sv2 \text{ statO}. \exists i < m. \Delta s i w s1 s2$   
 $\text{statA } sv1 sv2 \text{ statO}$   
**have**  $i: \text{initCond } \Delta$   
**using** *init m unfolding initCond-def* **by** *meson*  
**have**  $c: \text{unwindCond } \Delta$  **unfolding** *unwindCond-def* **apply**(*intro allI impI allI*)  
**apply**(*subst (asm) D*) **apply** (*elim exE conjE*)  
**subgoal for**  $w s1 s2 \text{ statA } sv1 sv2 \text{ statO } i$   
**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 -*  
 $\text{statA}]$ )  
**apply**(*erule allE[of - sv1]*) **apply**(*erule allE[of - sv2]*) **apply**(*erule allE[of -*  
 $\text{statO}]$ )  
**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  $\lambda w s1 s2 \text{ statA } sv1 sv2 \text{ statO}. \exists j \in \text{next } i. \Delta s j w$*   
 $s1 s2 \text{ statA } sv1 sv2 \text{ statO}]$ )  
**subgoal** **unfolding** *le-fun-def D* **by** *simp (meson atLeastLessThan-iff next*  
 $\text{subsetD})$   
**subgoal** . . . .  
**subgoal** **apply**(*rule disjI2*)  
**apply**(*rule match-mono[of  $\lambda w s1 s2 \text{ statA } sv1 sv2 \text{ statO}. \exists j \in \text{next } i. \Delta s j w$*   
 $s1 s2 \text{ statA } sv1 sv2 \text{ statO}]$ )  
**subgoal** **unfolding** *le-fun-def D* **by** *simp (meson atLeastLessThan-iff next*  
 $\text{subsetD})$   
**subgoal** . . . .  
**show** *?thesis* **using** *unwind-rsecure[OF i c]* .  
**qed**

**corollary** *linear-unwind-rsecure*:  
**assumes** *init*: *initCond* ( $\Delta s 0$ )  
**and** *step*:  $(\bigwedge i. i < m \implies$   
 $\text{unwindIntoCond } (\Delta s i) (\lambda w s1 s2 \text{ statA } sv1 sv2 \text{ statO}.$   
 $\quad \Delta s i w s1 s2 \text{ statA } sv1 sv2 \text{ statO} \vee$   
 $\quad \Delta s (\text{Suc } i) w s1 s2 \text{ statA } sv1 sv2 \text{ statO}))$

**and** *finish*: *unwindIntoCond* ( $\Delta s\ m$ ) ( $\Delta s\ m$ )  
**shows** *rsecure*  
**apply**(*rule distrib-unwind-rsecure*[*of Suc m*  $\lambda i.$  *if*  $i < m$  *then*  $\{i, \text{Suc } i\}$  *else*  $\{i\}$   $\Delta s,$   
*OF - - init*])  
**using** *step finish*  
**by** (*auto simp: less-Suc-eq-le*)

**definition** *oor* **where**

$\text{oor } \Delta\ \Delta_2 \equiv \lambda w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}.$   
 $\Delta\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \vee \Delta_2\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$

**lemma** *oorI1*:

$\Delta\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies \text{oor } \Delta\ \Delta_2\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**unfolding** *oor-def* **by** *simp*

**lemma** *oorI2*:

$\Delta_2\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies \text{oor } \Delta\ \Delta_2\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**unfolding** *oor-def* **by** *simp*

**definition** *oor3* **where**

$\text{oor3 } \Delta\ \Delta_2\ \Delta_3 \equiv \lambda w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}.$   
 $\Delta\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \vee \Delta_2\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \vee$   
 $\Delta_3\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$

**lemma** *oor3I1*:

$\Delta\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies \text{oor3 } \Delta\ \Delta_2\ \Delta_3\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**unfolding** *oor3-def* **by** *simp*

**lemma** *oor3I2*:

$\Delta_2\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies \text{oor3 } \Delta\ \Delta_2\ \Delta_3\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**unfolding** *oor3-def* **by** *simp*

**lemma** *oor3I3*:

$\Delta_3\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies \text{oor3 } \Delta\ \Delta_2\ \Delta_3\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**unfolding** *oor3-def* **by** *simp*

**definition** *oor4* **where**

$\text{oor4 } \Delta\ \Delta_2\ \Delta_3\ \Delta_4 \equiv \lambda w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}.$   
 $\Delta\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \vee \Delta_2\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \vee$   
 $\Delta_3\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \vee \Delta_4\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$

**lemma** *oor4I1*:

$\Delta\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies \text{oor4 } \Delta\ \Delta_2\ \Delta_3\ \Delta_4\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**unfolding** *oor4-def* **by** *simp*

**lemma** *oor4I2*:

$\Delta_2\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies \text{oor4 } \Delta\ \Delta_2\ \Delta_3\ \Delta_4\ w\ s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$

**unfolding** *oor4-def* by *simp*

**lemma** *oor4I3*:

$\Delta_3 w s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w s1 s2 statA sv1 sv2 statO$

**unfolding** *oor4-def* by *simp*

**lemma** *oor4I4*:

$\Delta_4 w s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w s1 s2 statA sv1 sv2 statO$

**unfolding** *oor4-def* by *simp*

**definition** *oor5* where

$oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \equiv \lambda w s1 s2 statA sv1 sv2 statO.$

$\Delta w s1 s2 statA sv1 sv2 statO \vee \Delta_2 w s1 s2 statA sv1 sv2 statO \vee$

$\Delta_3 w s1 s2 statA sv1 sv2 statO \vee \Delta_4 w s1 s2 statA sv1 sv2 statO \vee$

$\Delta_5 w s1 s2 statA sv1 sv2 statO$

**lemma** *oor5I1*:

$\Delta w s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *oor5I2*:

$\Delta_2 w s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *oor5I3*:

$\Delta_3 w s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *oor5I4*:

$\Delta_4 w s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *oor5I5*:

$\Delta_5 w s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**definition** *oor6* where

$oor6 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 \equiv \lambda w s1 s2 statA sv1 sv2 statO.$

$\Delta w s1 s2 statA sv1 sv2 statO \vee \Delta_2 w s1 s2 statA sv1 sv2 statO \vee$

$\Delta_3 w s1 s2 statA sv1 sv2 statO \vee \Delta_4 w s1 s2 statA sv1 sv2 statO \vee$

$\Delta_5 w s1 s2 statA sv1 sv2 statO \vee \Delta_6 w s1 s2 statA sv1 sv2 statO$

**lemma** *oor6I1*:

$\Delta w s1 s2 statA sv1 sv2 statO \implies oor6 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 w s1 s2 statA sv1$

*sv2 statO*  
**unfolding** *oor6-def by simp*

**lemma** *oor6I2*:  
 $\Delta_2 w s1 s2 statA sv1 sv2 statO \implies oor6 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 w s1 s2 statA sv1 sv2 statO$   
**unfolding** *oor6-def by simp*

**lemma** *oor6I3*:  
 $\Delta_3 w s1 s2 statA sv1 sv2 statO \implies oor6 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 w s1 s2 statA sv1 sv2 statO$   
**unfolding** *oor6-def by simp*

**lemma** *oor6I4*:  
 $\Delta_4 w s1 s2 statA sv1 sv2 statO \implies oor6 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 w s1 s2 statA sv1 sv2 statO$   
**unfolding** *oor6-def by simp*

**lemma** *oor6I5*:  
 $\Delta_5 w s1 s2 statA sv1 sv2 statO \implies oor6 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 w s1 s2 statA sv1 sv2 statO$   
**unfolding** *oor6-def by simp*

**lemma** *oor6I6*:  
 $\Delta_6 w s1 s2 statA sv1 sv2 statO \implies oor6 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \Delta_6 w s1 s2 statA sv1 sv2 statO$   
**unfolding** *oor6-def by simp*

**lemma** *unwind-rsecure-foo*:  
  **assumes** *init*: *initCond*  $\Delta_0$   
    **and** *step0*: *unwindIntoCond*  $\Delta_0 \Delta_{NN}$   
    **and** *stepNN*: *unwindIntoCond*  $\Delta_{NN}$  (*oor5*  $\Delta_{NN} \Delta_{SN} \Delta_{NS} \Delta_{SS} \Delta_{nonspec}$ )  
    **and** *stepNS*: *unwindIntoCond*  $\Delta_{NS}$  (*oor4*  $\Delta_{NN} \Delta_{SN} \Delta_{NS} \Delta_{SS}$ )  
    **and** *stepSN*: *unwindIntoCond*  $\Delta_{SN}$  (*oor4*  $\Delta_{NN} \Delta_{SN} \Delta_{NS} \Delta_{SS}$ )  
    **and** *stepSS*: *unwindIntoCond*  $\Delta_{SS}$  (*oor4*  $\Delta_{NN} \Delta_{SN} \Delta_{NS} \Delta_{SS}$ )  
    **and** *stepNonspec*: *unwindIntoCond*  $\Delta_{nonspec} \Delta_{nonspec}$   
  **shows** *rsecure*

**proof** –  
  **define** *m* **where** *m*: *m*  $\equiv (6::nat)$   
  **define**  $\Delta s$  **where**  $\Delta s$ :  $\Delta s \equiv \lambda i::nat.$   
    **if** *i* = 0 **then**  $\Delta_0$   
    **else if** *i* = 1 **then**  $\Delta_{NN}$   
    **else if** *i* = 2 **then**  $\Delta_{SN}$   
    **else if** *i* = 3 **then**  $\Delta_{NS}$   
    **else if** *i* = 4 **then**  $\Delta_{SS}$   
    **else**  $\Delta_{nonspec}$

```

define nxt where nxt: nat  $\equiv \lambda i::nat.$ 
  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  $\Delta$ s])
  subgoal unfolding m by auto
  subgoal unfolding nxt m by auto
  subgoal using init unfolding  $\Delta$ s by auto
  subgoal
    unfolding m nxt  $\Delta$ s
    using step0 stepNN stepNS stepSN stepSS stepNonspec
    unfolding oor4-def oor5-def by auto .
qed

```

```

lemma isIntO-match1: isIntO s1  $\implies$  match1  $\Delta$  s1 s2 statA sv1 sv2 statO
unfolding match1-def by auto

```

```

lemma isIntO-match2: isIntO s2  $\implies$  match2  $\Delta$  s1 s2 statA sv1 sv2 statO
unfolding match2-def by auto

```

```

lemma match1-1-oorI1:
  match1-1  $\Delta$  s1 s1' s2 statA sv1 sv2 statO  $\implies$ 
  match1-1 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta_2$  s1 s1' s2 statA sv1 sv2 statO  $\implies$ 
  match1-1 (oor  $\Delta$   $\Delta_2$ ) s1 s1' s2 statA sv1 sv2 statO
apply(rule match1-1-mono) unfolding le-fun-def oor-def by auto

```

```

lemma match1-oorI1:
  match1  $\Delta$  s1 s2 statA sv1 sv2 statO  $\implies$ 
  match1 (oor  $\Delta$   $\Delta_2$ ) s1 s2 statA sv1 sv2 statO
apply(rule match1-mono) unfolding le-fun-def oor-def by auto

```

```

lemma match1-oorI2:
  match1  $\Delta_2$  s1 s2 statA sv1 sv2 statO  $\implies$ 
  match1 (oor  $\Delta$   $\Delta_2$ ) s1 s2 statA sv1 sv2 statO
apply(rule match1-mono) unfolding le-fun-def oor-def by auto

```

**lemma** *match2-1-oorI1*:  
 $match2-1 \Delta s1 s2 s2' statA sv1 sv2 statO \implies$   
 $match2-1 (oor \Delta \Delta_2) s1 s2 s2' statA sv1 sv2 statO$   
**apply**(rule *match2-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match2-1-oorI2*:  
 $match2-1 \Delta_2 s1 s2 s2' statA sv1 sv2 statO \implies$   
 $match2-1 (oor \Delta \Delta_2) s1 s2 s2' statA sv1 sv2 statO$   
**apply**(rule *match2-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match2-oorI1*:  
 $match2 \Delta s1 s2 statA sv1 sv2 statO \implies$   
 $match2 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
**apply**(rule *match2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match2-oorI2*:  
 $match2 \Delta_2 s1 s2 statA sv1 sv2 statO \implies$   
 $match2 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
**apply**(rule *match2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-oorI1*:  
 $match12 \Delta s1 s2 statA sv1 sv2 statO \implies$   
 $match12 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
**apply**(rule *match12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-oorI2*:  
 $match12 \Delta_2 s1 s2 statA sv1 sv2 statO \implies$   
 $match12 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
**apply**(rule *match12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-1-oorI1*:  
 $match12-1 \Delta s1' s2' statA' sv1 sv2 statO \implies$   
 $match12-1 (oor \Delta \Delta_2) s1' s2' statA' sv1 sv2 statO$   
**apply**(rule *match12-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-1-oorI2*:  
 $match12-1 \Delta_2 s1' s2' statA' sv1 sv2 statO \implies$   
 $match12-1 (oor \Delta \Delta_2) s1' s2' statA' sv1 sv2 statO$   
**apply**(rule *match12-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-2-oorI1*:  
 $match12-2 \Delta s2 s2' statA' sv1 sv2 statO \implies$   
 $match12-2 (oor \Delta \Delta_2) s2 s2' statA' sv1 sv2 statO$   
**apply**(rule *match12-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-2-oorI2*:  
 $match12-2 \Delta_2 s2 s2' statA' sv1 sv2 statO \implies$

*match12-2 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta$  s1' s2' statA' sv1 sv2 statO  $\implies$*   
*match12-12 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta_2$  s1' s2' statA' sv1 sv2 statO  $\implies$*   
*match12-12 (oor  $\Delta$   $\Delta_2$ ) s1' s2' statA' sv1 sv2 statO*  
**apply**(rule *match12-12-mono*) **unfolding** *le-fun-def oor-def by auto*

**lemma** *match-oorI1*:  
*react  $\Delta$  s1 s2 statA sv1 sv2 statO  $\implies$*   
*react (oor  $\Delta$   $\Delta_2$ ) s1 s2 statA sv1 sv2 statO*  
**apply**(rule *match-mono*) **unfolding** *le-fun-def oor-def by auto*

**lemma** *match-oorI2*:  
*react  $\Delta_2$  s1 s2 statA sv1 sv2 statO  $\implies$*   
*react (oor  $\Delta$   $\Delta_2$ ) s1 s2 statA sv1 sv2 statO*  
**apply**(rule *match-mono*) **unfolding** *le-fun-def oor-def by auto*

**lemma** *proact-oorI1*:  
*proact  $\Delta$  meas s1 s2 statA sv1 sv2 statO  $\implies$*   
*proact (oor  $\Delta$   $\Delta_2$ ) meas s1 s2 statA sv1 sv2 statO*  
**apply**(rule *proact-mono*) **unfolding** *le-fun-def oor-def by auto*

**lemma** *proact-oorI2*:  
*proact  $\Delta_2$  meas s1 s2 statA sv1 sv2 statO  $\implies$*   
*proact (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta$  meas s1 s2 statA sv1 sv2 statO  $\implies$*   
*move-1 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta_2$  meas s1 s2 statA sv1 sv2 statO  $\implies$*   
*move-1 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta$  meas s1 s2 statA sv1 sv2 statO  $\implies$*

*move-2* ( $\text{oor } \Delta \Delta_2$ ) *meas*  $s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**apply**(rule *move-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *move-2-oorI2*:  
*move-2*  $\Delta_2$  *meas*  $s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies$   
*move-2* ( $\text{oor } \Delta \Delta_2$ ) *meas*  $s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**apply**(rule *move-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *move-12-oorI1*:  
*move-12*  $\Delta$  *meas*  $s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies$   
*move-12* ( $\text{oor } \Delta \Delta_2$ ) *meas*  $s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**apply**(rule *move-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *move-12-oorI2*:  
*move-12*  $\Delta_2$  *meas*  $s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO} \implies$   
*move-12* ( $\text{oor } \Delta \Delta_2$ ) *meas*  $s1\ s2\ \text{statA}\ sv1\ sv2\ \text{statO}$   
**apply**(rule *move-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**end**

**context** *Relative-Security-Determ*  
**begin**

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

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

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

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

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

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

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

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

**lemma** *match12-12-defD*:  $match12-12 \Delta s1' s2' statA' sv1 sv2 statO \longleftrightarrow$   
 $(let statO' = sstatO' statO sv1 sv2 in$   
 $\neg finalV sv1 \wedge \neg finalV sv2 \wedge$   
 $(statA' = Diff \longrightarrow statO' = Diff) \wedge$   
 $\Delta \infty s1' s2' statA' (nextO sv1) (nextO sv2) statO')$   
**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*:  $move-1 \Delta w s1 s2 statA sv1 sv2 statO \longleftrightarrow$   
 $\neg finalV sv1 \wedge \Delta w s1 s2 statA (nextO sv1) sv2 statO$   
**unfolding** *move-1-def validTrans-iff-next* **by** *simp*

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

**lemma** *move-12-defD*:  $move-12 \Delta w s1 s2 statA sv1 sv2 statO \longleftrightarrow$   
 $(let statO' = sstatO' statO sv1 sv2 in$   
 $\neg finalV sv1 \wedge \neg finalV sv2 \wedge$   
 $\Delta w s1 s2 statA (nextO sv1) (nextO sv2) statO')$   
**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 newStat :: status  $\Rightarrow$  bool  $\times$  'a  $\Rightarrow$  bool  $\times$  'a  $\Rightarrow$  status where
  newStat Eq (True,a) (True,a') = (if a = a' then Eq else Diff)
| newStat stat - - = stat

```

```

definition sstatO' statO sv1 sv2 = newStat statO (isIntV sv1, getObsV sv1)
(isIntV sv2, getObsV sv2)

```

```

definition sstatA' statA s1 s2 = newStat statA (isIntO s1, getObsO s1) (isIntO
s2, getObsO s2)

```

```

lemma newStat-EqI:
  assumes  $\langle R = S \rangle$ 
  shows  $\langle \text{newStat } Eq (P, R) (Q, S) = Eq \rangle$ 
  apply (cases P)
  apply (metis assms newStat.simps(1) newStat.simps(4))
  by (cases Q) auto

```

```

lemma newStat-diff: newStat stat r r = Diff  $\implies$  stat = Diff
  by (metis newStat.elims newStat.simps(1))

```

```

definition initCond ::
  (enat  $\Rightarrow$  enat  $\Rightarrow$  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
       $\wedge \Delta \infty \infty \infty s1 s2 Eq sv1 sv2 Eq$ )

```

```

definition match1-1  $\Delta w1 w2 s1 s1' s2 statA sv1 sv2 statO \equiv$ 
   $\exists sv1'. \text{validTransV}(sv1, sv1') \wedge$ 
   $\Delta \infty w1 w2 s1' s2 statA sv1' sv2 statO$ 

```

**definition**  $match1-12 \Delta w1 w2 s1 s1' s2 statA sv1 sv2 statO \equiv$   
 $(\exists sv1' sv2'.$   
 $let statO' = sstatO' statO sv1 sv2 in$   
 $validTransV (sv1,sv1') \wedge$   
 $validTransV (sv2,sv2') \wedge$   
 $\Delta \infty w1 w2 s1' s2 statA sv1' sv2' statO')$

**definition**  $match1 \Delta w1 w2 s1 s2 statA sv1 sv2 statO \equiv$   
 $\neg isIntO s1 \longrightarrow$   
 $(\forall s1'. validTransO (s1,s1')$   
 $\longrightarrow$   
 $(\exists w1' < w1. \exists w2' < w2. \neg isSecO s1 \wedge \Delta \infty w1' w2' s1' s2 statA sv1 sv2$   
 $statO) \vee$   
 $(\exists w2' < w2. eqSec sv1 s1 \wedge \neg isIntV sv1 \wedge match1-1 \Delta \infty w2' s1 s1' s2$   
 $statA sv1 sv2 statO) \vee$   
 $(eqSec sv1 s1 \wedge \neg isSecV sv2 \wedge Van.eqAct sv1 sv2 \wedge match1-12 \Delta \infty \infty s1$   
 $s1' s2 statA sv1 sv2 statO))$

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

**lemma**  $match1-1-mono:$   
 $\Delta \leq \Delta' \implies match1-1 \Delta w1 w2 s1 s1' s2 statA sv1 sv2 statO \implies$   
 $match1-1 \Delta' w1 w2 s1 s1' s2 statA sv1 sv2 statO$   
**unfolding**  $le-fun-def match1-1-def$  **by**  $auto$

**lemma**  $match1-12-mono:$   
 $\Delta \leq \Delta' \implies match1-12 \Delta w1 w2 s1 s1' s2 statA sv1 sv2 statO \implies$   
 $match1-12 \Delta' w1 w2 s1 s1' s2 statA sv1 sv2 statO$   
**unfolding**  $le-fun-def match1-12-def$  **by**  $fastforce$

**lemma**  $match1-mono:$   
**assumes**  $\Delta \leq \Delta'$   
**shows**  $match1 \Delta w1 w2 s1 s2 statA sv1 sv2 statO \implies match1 \Delta' w1 w2 s1 s2$   
 $statA sv1 sv2 statO$   
**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**  $match2-1 \Delta w1 w2 s1 s2 s2' statA sv1 sv2 statO \equiv$   
 $\exists sv2'. validTransV (sv2,sv2') \wedge$   
 $\Delta \infty w1 w2 s1 s2' statA sv1 sv2' statO$

**definition**  $match2-12 \Delta w1 w2 s1 s2 s2' statA sv1 sv2 statO \equiv$   
 $\exists sv1' sv2'.$

$let\ statO' = sstatO'\ statO\ sv1\ sv2\ in$   
 $validTransV\ (sv1,sv1') \wedge$   
 $validTransV\ (sv2,sv2') \wedge$   
 $\Delta \infty w1\ w2\ s1\ s2'\ statA\ sv1'\ sv2'\ statO'$

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

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

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

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

**lemma**  $match2-mono:$   
**assumes**  $\Delta \leq \Delta'$   
**shows**  $match2\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \implies match2\ \Delta'\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**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**  $match12-1\ \Delta\ w1\ w2\ s1'\ s2'\ statA'\ sv1\ sv2\ statO \equiv$   
 $\exists\ sv1'.\ validTransV\ (sv1,sv1') \wedge$   
 $\Delta \infty w1\ w2\ s1'\ s2'\ statA'\ sv1'\ sv2\ statO$

**definition**  $match12-2\ \Delta\ w1\ w2\ s1'\ s2'\ statA'\ sv1\ sv2\ statO \equiv$   
 $\exists\ sv2'.\ validTransV\ (sv2,sv2') \wedge$   
 $\Delta \infty w1\ w2\ s1'\ s2'\ statA'\ sv1\ sv2'\ statO$

**definition**  $match12-12\ \Delta\ w1\ w2\ s1'\ s2'\ statA'\ sv1\ sv2\ statO \equiv$

$\exists sv1' sv2'$ .  
*let*  $statO' = sstatO' statO sv1 sv2$  *in*  
 $validTransV (sv1, sv1') \wedge$   
 $validTransV (sv2, sv2') \wedge$   
 $(statA' = Diff \longrightarrow statO' = Diff) \wedge$   
 $\Delta \infty w1 w2 s1' s2' statA' sv1' sv2' statO'$

**definition**  $match12 \Delta w1 w2 s1 s2 statA sv1 sv2 statO \equiv$   
 $\forall s1' s2'$ .

*let*  $statA' = sstatA' statA s1 s2$  *in*  
 $validTransO (s1, s1') \wedge$   
 $validTransO (s2, s2') \wedge$   
 $Opt.eqAct s1 s2 \wedge$   
 $isIntO s1 \wedge isIntO s2$   
 $\longrightarrow$   
 $(\exists w1' < w1. \exists w2' < w2. \neg isSecO s1 \wedge \neg isSecO s2 \wedge (statA = statA' \vee statO = Diff)) \wedge$   
 $\Delta \infty w1' w2' s1' s2' statA' sv1 sv2 statO)$   
 $\vee$   
 $(\exists w2' < w2. \neg isSecO s2 \wedge eqSec sv1 s1 \wedge \neg isIntV sv1 \wedge$   
 $(statA = statA' \vee statO = Diff) \wedge$   
 $match12-1 \Delta \infty w2' s1' s2' statA' sv1 sv2 statO)$   
 $\vee$   
 $(\exists w1' < w1. \neg isSecO s1 \wedge eqSec sv2 s2 \wedge \neg isIntV sv2 \wedge$   
 $(statA = statA' \vee statO = Diff) \wedge$   
 $match12-2 \Delta w1' \infty s1' s2' statA' sv1 sv2 statO)$   
 $\vee$   
 $(eqSec sv1 s1 \wedge eqSec sv2 s2 \wedge Van.eqAct sv1 sv2 \wedge$   
 $match12-12 \Delta \infty \infty s1' s2' statA' sv1 sv2 statO)$

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

**lemma**  $match12-simpleI$ :

**assumes**  $\bigwedge s1' s2' statA'$ .

$statA' = sstatA' statA s1 s2 \implies$   
 $validTransO (s1, s1') \implies$   
 $validTransO (s2, s2') \implies$   
 $Opt.eqAct s1 s2 \implies$   
 $(\exists w1' < w1. \exists w2' < w2. \neg isSecO s1 \wedge \neg isSecO s2 \wedge (statA = statA' \vee statO = Diff)) \wedge$   
 $\Delta \infty w1' w2' s1' s2' statA' sv1 sv2 statO)$   
 $\vee$   
 $(eqSec sv1 s1 \wedge eqSec sv2 s2 \wedge Van.eqAct sv1 sv2 \wedge$   
 $match12-12 \Delta \infty \infty s1' s2' statA' sv1 sv2 statO)$

**shows**  $match12 \Delta w1 w2 s1 s2 statA sv1 sv2 statO$

**using** *assms unfolding match12-def Let-def by blast*

**lemma**  $match12-1-mono$ :

$\Delta \leq \Delta' \implies \text{match12-1 } \Delta \ w1 \ w2 \ s1' \ s2' \ \text{statA}' \ \text{sv1} \ \text{sv2} \ \text{statO} \implies \text{match12-1 } \Delta' \ w1 \ w2 \ s1' \ s2' \ \text{statA}' \ \text{sv1} \ \text{sv2} \ \text{statO}$

**unfolding** *le-fun-def match12-1-def* **by** *auto*

**lemma** *match12-2-mono*:

$\Delta \leq \Delta' \implies \text{match12-2 } \Delta \ w1 \ w2 \ s1 \ s2' \ \text{statA}' \ \text{sv1} \ \text{sv2} \ \text{statO} \implies \text{match12-2 } \Delta' \ w1 \ w2 \ s1 \ s2' \ \text{statA}' \ \text{sv1} \ \text{sv2} \ \text{statO}$

**unfolding** *le-fun-def match12-2-def* **by** *auto*

**lemma** *match12-12-mono*:

$\Delta \leq \Delta' \implies \text{match12-12 } \Delta \ w1 \ w2 \ s1' \ s2' \ \text{statA}' \ \text{sv1} \ \text{sv2} \ \text{statO} \implies \text{match12-12 } \Delta' \ w1 \ w2 \ s1' \ s2' \ \text{statA}' \ \text{sv1} \ \text{sv2} \ \text{statO}$

**unfolding** *le-fun-def match12-12-def* **by** *fastforce*

**lemma** *match12-mono*:

**assumes**  $\Delta \leq \Delta'$

**shows**  $\text{match12 } \Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO} \implies \text{match12 } \Delta' \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO}$

**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* - -  $s1' \ s2' - \text{sv1} \ \text{sv2} \ \text{statO}$ ]

*match12-2-mono*[*OF* *assms*, *of* - -  $s1' \ s2' - \text{sv1} \ \text{sv2} \ \text{statO}$ ]

*match12-12-mono*[*OF* *assms*, *of* - -  $s1' \ s2' - \text{sv1} \ \text{sv2} \ \text{statO}$ ]

*assms*[*unfolded* *le-fun-def*, *rule-format*, *of* - - -  $s1' \ s2'$

*sstatA' statA s1 s2 sv1 sv2 statO*]

**apply** *simp* **by** *blast* .

**definition** *react*  $\Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO} \equiv$

*match1*  $\Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO}$

$\wedge$

*match2*  $\Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO}$

$\wedge$

*match12*  $\Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO}$

**lemmas** *react-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**  $\text{react } \Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO} \implies \text{react } \Delta' \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO}$

**unfolding** *react-def* **using** *match1-mono*[*OF* *assms*] *match2-mono*[*OF* *assms*] *match12-mono*[*OF* *assms*] **by** *auto*

**definition** *move-1*  $\Delta \ w \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ \text{sv1} \ \text{sv2} \ \text{statO} \equiv$

$\exists sv1'. \text{validTransV } (sv1, sv1') \wedge$   
 $\Delta w w1 w2 s1 s2 \text{statA } sv1' sv2 \text{statO}$

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

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

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

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

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

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

**lemma** *move-12-mono*:  
 $\Delta \leq \Delta' \implies \text{move-12 } \Delta w w1 w2 s1 s2 \text{statA } sv1 sv2 \text{statO} \implies \text{move-12 } \Delta' w w1$   
 $w2 s1 s2 \text{statA } sv1 sv2 \text{statO}$   
**unfolding** *le-fun-def move-12-def* **by** *fastforce*

**lemma** *proact-mono*:  
**assumes**  $\Delta \leq \Delta'$   
**shows** *proact*  $\Delta w w1 w2 s1 s2 \text{statA } sv1 sv2 \text{statO} \implies \text{proact } \Delta' w w1 w2 s1 s2$   
 $\text{statA } sv1 sv2 \text{statO}$   
**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* ::  
 $(\text{enat} \Rightarrow \text{enat} \Rightarrow \text{enat} \Rightarrow 'stateO \Rightarrow 'stateO \Rightarrow \text{status} \Rightarrow 'stateV \Rightarrow 'stateV \Rightarrow$

$status \Rightarrow bool) \Rightarrow bool$

**where**

$unwindCond \Delta \equiv \forall w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \wedge reachO s2 \wedge reachV sv1 \wedge reachV sv2 \wedge$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
 $\longrightarrow$   
 $(finalO s1 \longleftrightarrow finalO s2) \wedge (finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO$   
 $s2)$   
 $\wedge$   
 $(statA = Eq \longrightarrow (isIntO s1 \longleftrightarrow isIntO s2))$   
 $\wedge$   
 $((\exists v < w. proact \Delta v w1 w2 s1 s2 statA sv1 sv2 statO)$   
 $\vee$   
 $react \Delta w1 w2 s1 s2 statA sv1 sv2 statO$   
 $)$

**lemma** *unwindCond-simpleI*:

**assumes**

$\wedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \Longrightarrow reachO s2 \Longrightarrow reachV sv1 \Longrightarrow reachV sv2 \Longrightarrow$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
 $\Longrightarrow$   
 $(finalO s1 \longleftrightarrow finalO s2) \wedge (finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO$   
 $s2)$

**and**

$\wedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \Longrightarrow reachO s2 \Longrightarrow reachV sv1 \Longrightarrow reachV sv2 \Longrightarrow$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \Longrightarrow statA = Eq$   
 $\Longrightarrow$   
 $isIntO s1 \longleftrightarrow isIntO s2$

**and**

$\wedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \Longrightarrow reachO s2 \Longrightarrow reachV sv1 \Longrightarrow reachV sv2 \Longrightarrow$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
 $\Longrightarrow$   
 $react \Delta w1 w2 s1 s2 statA sv1 sv2 statO$

**shows** *unwindCond*  $\Delta$

**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**  $\psi s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2 \equiv$   
 $trv1 \neq \square \wedge trv2 \neq \square \wedge$   
 $Van.validFromS sv1 trv1 \wedge$

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

**lemma**  $\psi$ -completedFrom:  $completedFromO\ s1\ tr1 \implies completedFromO\ s2\ tr2 \implies$

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

**lemma**  $completedFromO$ -lastt:  $completedFromO\ s1\ tr1 \implies finalO\ (lastt\ s1\ tr1)$

**unfolding**  $Opt.completedFrom$ -def  $lastt$ -def **by** *auto*

**lemma** *rsecure-strong*:

**assumes**

$\bigwedge s1\ tr1\ s2\ tr2.$

$istateO\ s1 \wedge Opt.validFromS\ s1\ tr1 \wedge completedFromO\ s1\ tr1 \wedge$

$istateO\ s2 \wedge Opt.validFromS\ s2\ tr2 \wedge completedFromO\ s2\ tr2 \wedge$

$Opt.A\ tr1 = Opt.A\ tr2$

$\implies$

$\exists sv1\ trv1\ sv2\ trv2.$

$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$

$\psi\ s1\ tr1\ s2\ tr2\ Eq\ sv1\ trv1\ sv2\ trv2$

**shows** *rsecure*

**unfolding** *rsecure-def2* **apply** *safe*

**subgoal for**  $s1\ tr1\ s2\ tr2$

**using** *assms*[of  $s1\ tr1\ s2\ tr2$ ]

**using**  $\psi$ -completedFrom  $\psi$ -def  $completedFromO$ -lastt **apply** *clarsimp* **by** *metis* .

**proposition** *unwindCond-ex- $\psi$* :

**assumes** *unwind*:  $unwindCond\ \Delta$

**and**  $\Delta$ :  $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$  **and** *stat*:  $(statA = Diff \longrightarrow statO = Diff)$

**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**  $\exists trv1\ trv2. \psi\ s1\ tr1\ s2\ tr2\ statO\ sv1\ trv1\ sv2\ trv2$

**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.prem(8,9) note r12 = less.prem(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) ∨ react
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.prem(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.prem(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  $0: statO' = sstatO' statO sv1 sv2$ 
 $validTransV (sv1,sv1') \neg isSecV sv1$ 
 $validTransV (sv2,sv2') \neg isSecV sv2$ 
 $Van.eqAct sv1 sv2$ 
and  $\Delta: \Delta v w1 w2 s1 s2 statA sv1' sv2' statO'$ 
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  $ok': statA = Diff \longrightarrow statO' = Diff$  using  $ok 0$  unfolding sstatO'-def
by (cases statO, auto)
obtain  $trv1 trv2$  where  $\psi: \psi s1 tr1 s2 tr2 statO' sv1' trv1 sv2' trv2$ 
using less(2)[OF v, of tr1 tr2 w1 w2 s1 s2 statA sv1' sv2' statO', simplified,
OF  $\Delta ok' - - - - r34 r12'$ ]
using A34 less.prem(3-6) by blast
show ?thesis apply(rule exI[of - sv1 # trv1]) apply(rule exI[of - sv2 #
trv2])
using  $\psi ok' 0 sv12$  unfolding  $\psi$ -def sstatO'-def Van.completedFrom-def
using Van.A.Cons-unfold Van.eqAct-def completedFromO-lastt less.prem(4)
less.prem(6) by auto
qed
next
assume  $m: react \Delta w1 w2 s1 s2 statA sv1 sv2 statO$ 
show ?thesis
proof(cases length tr1  $\leq$  Suc 0)
case True note  $tr1 = True$ 
hence  $tr1 = [] \vee tr1 = [s1]$ 
by (metis Simple-Transition-System.validFromS-Cons-iff Suc-length-conv le-Suc-eq
le-zero-eq length-0-conv less.prem(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  $tr2: tr2 = [] \vee tr2 = [s2]$ 
by (metis Opt.final-def Simple-Transition-System.validFromS-Cons-iff less.prem(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.prem(4)
by (auto simp add: lastt-def  $\psi$ -def)
next
case False
then obtain  $s13 tr1'$  where  $tr1: tr1 = s13 \# tr1'$  and  $tr1'NE: tr1' \neq []$ 
by (cases tr1, auto)
have  $s13[simp]: s13 = s1$  using  $\langle Opt.validFromS s1 tr1 \rangle$ 
by (simp add: Opt.validFromS-Cons-iff tr1)
obtain  $s1'$  where
 $trn3: validTransO (s1, s1')$  and
 $tr1': Opt.validFromS s1' tr1'$  using  $\langle Opt.validFromS s1 tr1 \rangle$ 
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**  $f3$ :  $\neg\ finalO\ s1$  **using** *Opt.final-def trn3* **by** *blast*  
**hence**  $f4$ :  $\neg\ finalO\ s2$  **using**  $f3'$  **by** *blast*  
**hence**  $tr2$ :  $\neg\ length\ tr2 \leq\ Suc\ 0$   
**by** (*metis Opt.completed-Cons Simple-Transition-System.validFromS-Cons-iff*)  
  
*bot-nat-0.extremum completedFromO-def length-Cons less.prem5 less.prem6*  
*neq-Nil-conv not-less-eq*)  
  
**then obtain**  $s24\ tr2'$  **where**  $tr2$ :  $tr2 = s24 \# tr2'$  **and**  $tr2'NE$ :  $tr2' \neq []$   
**by** (*cases tr2, auto*)  
**have**  $s24[simp]$ :  $s24 = s2$  **using**  $\langle Opt.validFromS\ s2\ tr2 \rangle$   
**by** (*simp add: Opt.validFromS-Cons-iff tr2*)  
**obtain**  $s2'$  **where**  
 $trn4$ :  $validTransO\ (s2, s2') \vee (s2 = s2' \wedge tr2' = [])$  **and**  
 $tr2'$ :  $Opt.validFromS\ s2'\ tr2'$  **using**  $\langle Opt.validFromS\ s2\ tr2 \rangle$   
**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**  $\neg\ isIntO\ s1 \vee \neg\ isIntO\ s2 \vee (isIntO\ s1 \wedge isIntO\ s2)$   
**by** *auto*  
**thus** *?thesis*  
**proof** *safe*  
**assume**  $isAO3$ :  $\neg\ isIntO\ s1$   
**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**  $m$ :  $match1\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$  **using**  $m$  **unfolding**  
*react-def* **by** *auto*  
**have**  $(\exists\ w1' < w1. \exists\ w2' < w2. \neg\ isSecO\ s1 \wedge \Delta \infty\ w1'\ w2'\ s1'\ s2\ statA\ sv1\ sv2\ statO) \vee$   
 $(\exists\ w2' < w2. eqSec\ sv1\ s1 \wedge \neg\ isIntV\ sv1 \wedge match1-1\ \Delta \infty\ w2'\ s1\ s1'$   
 $s2\ statA\ sv1\ sv2\ statO) \vee$   
 $(eqSec\ sv1\ s1 \wedge \neg\ isSecV\ sv2 \wedge Van.eqAct\ sv1\ sv2 \wedge match1-12\ \Delta \infty$   
 $\infty\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO)$   
**using**  $m\ isAO3\ trn3\ ok$  **unfolding** *match1-def* **by** *auto*  
**thus** *?thesis*  
**proof** *safe*  
**fix**  $w1'\ w2'$   
**assume**  $\neg\ isSecO\ s1$  **and**  $\Delta$ :  $\Delta \infty\ w1'\ w2'\ s1'\ s2\ statA\ sv1\ sv2\ statO$   
**hence**  $S3$ :  $Opt.S\ tr1' = Opt.S\ tr1$  **unfolding**  $tr1$  **by** *auto*  
**obtain**  $trv1\ trv2$  **where**  $\psi$ :  $\psi\ s1\ tr1'\ s2\ tr2\ statO\ sv1\ trv1\ sv2\ trv2$   
**using**  $less(1)[of\ tr1'\ tr2, OF - \Delta - - - - - r3'\ r4\ r12, unfolded\ O33',$   
*simplified]*  
**using**  $less.prem5\ tr1'\ ok\ A34'\ f3\ f4$  **unfolding**  $tr1$  *Opt.completedFrom-def*

```

    by (auto split: if-splits simp:  $\psi$ -def lastt-def)
    show ?thesis apply(rule exI[of - trv1]) apply(rule exI[of - trv2])
    using  $\psi$  O33' S3 unfolding  $\psi$ -def
    using completedFromO-lastt less.premis(4)
    by (auto simp add: tr1 tr1'NE)
next
  fix w2'
  assume trn13: eqSec sv1 s1 and
  Atrn1:  $\neg$  isIntV sv1 and match1-1  $\Delta \infty w2' s1 s1' s2$  statA sv1 sv2 statO
  then obtain sv1' where
  trn1: validTransV (sv1,sv1') and
   $\Delta: \Delta \infty \infty w2' s1' s2$  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  $\psi: \psi s1 tr1' s2 tr2$  statO sv1' trv1 sv2 trv2
  using less(1)[of tr1' tr2, OF -  $\Delta$  - - - - - r3' r4 r1' r2, unfolded O33',
simplified]
  using less.premis tr1' ok A34' f3 f4 unfolding tr1 tr2 Opt.completedFrom-def

  by (auto simp:  $\psi$ -def lastt-def split: if-splits)
  show ?thesis apply(rule exI[of - sv1 # trv1]) apply(rule exI[of - trv2])
  using  $\psi$  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  $\psi$ -def using completedFromO-lastt less.premis(4) tr1 by auto

next
  assume sv2:  $\neg$  isSecV sv2 and trn13: eqSec sv1 s1 and
  Atrn12: Van.eqAct sv1 sv2 and match1-12  $\Delta \infty \infty s1 s1' s2$  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
   $\Delta: \Delta \infty \infty \infty s1' s2$  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  $\psi: \psi s1' tr1' s2 tr2$  statO' sv1' trv1 sv2' trv2
  using less(1)[of tr1' tr2, OF -  $\Delta$  - - - - - r3' r4 r12', unfolded O33',
simplified]
  using less.premis 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  $\psi$  O33' tr1'NE tr2'NE sv2
  using Van.validFromS-Cons trn1 trn2

```

```

    using isAO3 ok Atrn12 eqSec-S-Cons trn13 f3 f34 s13
  unfolding  $\psi$ -def tr1 Van.completedFrom-def Van.eqAct-def statO' sstatO'-def
    using Van.A.Cons-unfold tr1' trn3 by auto
  qed
next
  assume isAO4:  $\neg$  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:  $match2\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$  using m unfolding
react-def by auto
  have ( $\exists w1' < w1. \exists w2' < w2. \neg isSecO\ s2 \wedge \Delta \infty w1'\ w2'\ s1\ s2'\ statA\ sv1\ sv2\ statO$ )  $\vee$ 
    ( $\exists w1' < w1. eqSec\ sv2\ s2 \wedge \neg isIntV\ sv2 \wedge match2-1\ \Delta\ w1' \infty s1\ s2\ s2'\ statA\ sv1\ sv2\ statO$ )  $\vee$ 
    ( $\neg isSecV\ sv1 \wedge eqSec\ sv2\ s2 \wedge Van.eqAct\ sv1\ sv2 \wedge match2-12\ \Delta \infty s1\ s2\ s2'\ statA\ sv1\ sv2\ statO$ )
  using m isAO4 trn4 ok tr2'NE unfolding match2-def by auto
  thus ?thesis
  proof safe
    fix w1' w2'
    assume  $\neg isSecO\ s2$  and  $\Delta: \Delta \infty w1'\ w2'\ s1\ s2'\ statA\ sv1\ sv2\ statO$ 
    hence S4:  $Opt.S\ tr2' = Opt.S\ tr2$  unfolding tr2 by auto
    obtain trv1 trv2 where  $\psi: \psi\ s1\ tr1\ s2'\ tr2'\ statO\ sv1\ trv1\ sv2\ trv2$ 
    using less(1)[of tr1 tr2', OF -  $\Delta$  - - - - - r3 r4', simplified]
    using less.premis 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  $\psi\ O44'\ S4$  unfolding  $\psi$ -def
    using completedFromO-lastt less.premis(6)
    unfolding Opt.completedFrom-def using tr2 tr2'NE by auto
  next
    fix w1'
    assume trn24:  $eqSec\ sv2\ s2$  and
  Atrn2:  $\neg isIntV\ sv2$  and  $match2-1\ \Delta\ w1' \infty s1\ s2\ s2'\ statA\ sv1\ sv2\ statO$ 
    then obtain sv2' where  $trn2: validTransV\ (sv2, sv2')$  and
     $\Delta: \Delta \infty w1' \infty s1\ s2'\ statA\ sv1\ sv2'\ statO$ 
    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  $\psi: \psi\ s1\ tr1\ s2'\ tr2'\ statO\ sv1\ trv1\ sv2'\ trv2$ 
    using less(1)[of tr1 tr2', OF -  $\Delta$  - - - - - r3 r4' r1 r2', simplified]
    using less.premis 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  $\psi\ tr1'NE\ tr2'NE$ 
    using Van.validFromS-Cons trn2
    using isAO4 ok Atrn2 eqSec-S-Cons trn24

```

**unfolding**  $\psi$ -def tr1 tr2 s13 s24 Van.completedFrom-def lastt-def **by** auto  
**next**  
**assume** sv1:  $\neg$  isSecV sv1 **and** trn24: eqSec sv2 s2 **and**  
Atrn12: Van.eqAct sv1 sv2 **and** match2-12  $\Delta \infty \infty$  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**  
 $\Delta$ :  $\Delta \infty \infty \infty$  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**  $\psi$ :  $\psi$  s1 tr1 s2' tr2' statO' sv1' trv1 sv2' trv2  
**using** less(1)[of tr1 tr2', OF -  $\Delta$  - - - - - r3 r4' r12', simplified]  
**using** less.premis 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**  $\psi$  O44' tr1'NE tr2'NE sv1  
**using** Van.validFromS-Cons trn1 trn2  
**using** isAO4 ok Atrn12 eqSec-S-Cons trn24  
**unfolding**  $\psi$ -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  $\Delta$  w1 w2 s1 s2 statA sv1 sv2 statO **using** m **unfolding**  
statA' react-def **by** auto  
**have** trn34: getObsO s1 = getObsO s2  $\vee$  statA' = Diff  
**using** isAO34 **unfolding** statA' sstatA'-def **by** (cases statA,auto)  
**have** ( $\exists$  w1' < w1.  $\exists$  w2' < w2.  $\neg$  isSecO s1  $\wedge$   $\neg$  isSecO s2  $\wedge$  (statA = statA'  
 $\vee$  statO = Diff)  $\wedge$   
 $\Delta \infty$  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  $\Delta \infty$  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  $\Delta$  w1'  $\infty$  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.premis 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.premis(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.premis 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.premis(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.premis 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.premis 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
newStat.simps(4) newStat-diff)
qed
qed
qed

```

qed  
qed

**lemma** *unwindCond-final*:

$unwindCond \Delta \implies reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $(finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO s2)$

**unfolding** *unwindCond-def*

**unfolding** *proact-def react-def match1-def match1-1-def*

**by** *auto*

**definition**  $\varphi \Delta w w1 w2 w1' w2' statA s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2$   
 $trv2 statOO \equiv$

$trv1 \neq [] \wedge trv2 \neq [] \wedge$   
 $(length\ trv1 > Suc\ 0 \vee w1' \leq w1) \wedge (length\ trv2 > Suc\ 0 \vee w2' \leq w2) \wedge$   
 $Van.validFromS\ sv1\ trv1 \wedge$   
 $Van.validFromS\ sv2\ trv2 \wedge$   
 $Van.S\ trv1 = Opt.S\ tr1 \wedge Van.S\ trv2 = Opt.S\ tr2 \wedge$   
 $Van.A\ trv1 = Van.A\ trv2 \wedge$   
 $(statO = Eq \longrightarrow (statOO = Diff \longleftrightarrow Van.O\ trv1 \neq Van.O\ trv2)) \wedge$   
 $(statA = Eq \longrightarrow (statAA = Diff \longleftrightarrow Opt.O\ tr1 \neq Opt.O\ tr2)) \wedge$

—  
 $(statO = Diff \longrightarrow statOO = Diff) \wedge$

$(statAA = Diff \longrightarrow statOO = Diff) \wedge$

$\Delta w w1' w2' (lastt\ s1\ tr1) (lastt\ s2\ tr2) statAA (lastt\ sv1\ trv1) (lastt\ sv2\ trv2)$   
 $statOO$

**lemma**  $\varphi$ -*final*:

**assumes** *unw*:  $unwindCond\ \Delta$

**and** *r*:  $reachO\ s1\ reachO\ s2\ reachV\ sv1\ reachV\ sv2$

**and** *vtr14*:  $Opt.validFromS\ s1\ tr1\ Opt.validFromS\ s2\ tr2$

**and**  $\varphi$ :  $\varphi\ \Delta\ w\ w1\ w2\ w1'\ w2'\ statA\ s1\ tr1\ s2\ tr2\ statAA\ statO\ sv1\ trv1\ sv2\ trv2$   
 $statOO$

**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**  $\varphi$ [*unfolded*  $\varphi$ -*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 *utr14*(1) *utr14*(2) by auto  
qed

**lemma**  $\varphi$ -completedFrom: *unwindCond*  $\Delta \implies$   
*reachO* *s1*  $\implies$  *reachO* *s2*  $\implies$  *reachV* *sv1*  $\implies$  *reachV* *sv2*  $\implies$   
*Opt.validFromS* *s1* *tr1*  $\implies$  *completedFromO* *s1* *tr1*  $\implies$   
*Opt.validFromS* *s2* *tr2*  $\implies$  *completedFromO* *s2* *tr2*  $\implies$   
 $\varphi \Delta$  *statA* *w* *w1* *w2* *w1'* *w2'* *s1* *tr1* *s2* *tr2* *statAA* *statO* *sv1* *trv1* *sv2* *trv2* *statOO*  
 $\implies$  *completedFromV* *sv1* *trv1*  $\wedge$  *completedFromV* *sv2* *trv2*  
using  $\varphi$ -final  
by (*metis* *Van.completedFrom-def* *completedFromO-lasttt* *lasttt-def*)

**lemma** *unwindCond-ex- $\varphi$* :  
**assumes** *unwind*: *unwindCond*  $\Delta$   
**and**  $\Delta$ :  $\Delta$  *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**  $\exists w' w1' w2' trv1 trv2 statAA statOO. \varphi \Delta w' w1 w2 w1' w2' statA s1 tr1$   
 $s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO$   
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* =  $\langle statA = Diff \longrightarrow statO = Diff \rangle$   
**note**  $\Delta$  =  $\langle \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \rangle$   
**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*  $\wedge$  *finalV* *sv1* = *finalO* *s1*  $\wedge$  *finalV* *sv2* = *finalO*  
*s2*  
using  $\Delta$  *unwind*[*unfolded unwindCond-def*] *r* by auto

**note** *is1* =  $\langle isIntO (lastt s1 tr1) \rangle$   
**note** *is2* =  $\langle isIntO (lastt s2 tr2) \rangle$   
**note** *utr1* =  $\langle Opt.validFromS s1 tr1 \rangle$   
**note** *utr2* =  $\langle Opt.validFromS s2 tr2 \rangle$

**have** *proact-match*: ( $\exists v < w. proact \Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ )  $\vee$  *react*  
 $\Delta w1 w2 s1 s2 statA sv1 sv2 statO$   
using  $\Delta$  *unwind*[*unfolded unwindCond-def*] *r* by auto  
**show** ?*case* using *proact-match* **proof** *safe*  
**fix** *v* **assume** *v*: *v* < *w*

```

assume proact  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ 
thus ?thesis unfolding proact-def proof safe
assume sv1:  $\neg isSecV sv1 \neg isIntV sv1$  and move-1  $\Delta v w1 w2 s1 s2 statA$ 
sv1 sv2 statO
then obtain sv1'
where 0: validTransV (sv1,sv1')
and  $\Delta$ :  $\Delta v w1 w2 s1 s2 statA sv1' sv2 statO$ 
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  $\varphi$ :  $\varphi \Delta w' w1 w2 w1' w2'$ 
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  $\Delta r34 r1' r2 ok$ ]
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  $\varphi ok 0 sv1$  unfolding  $\varphi$ -def by auto
next
assume sv2:  $\neg isSecV sv2 \neg isIntV sv2$  and move-2  $\Delta v w1 w2 s1 s2 statA$ 
sv1 sv2 statO
then obtain sv2'
where 0: validTransV (sv2,sv2')
and  $\Delta$ :  $\Delta v w1 w2 s1 s2 statA sv1 sv2' statO$ 
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  $\varphi$ :  $\varphi \Delta w' w1 w2 w1' w2'$ 
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  $\Delta r34 r1 r2' ok$ ]
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  $\varphi ok 0 sv2$  unfolding  $\varphi$ -def by auto
next
assume sv12:  $\neg isSecV sv1 \neg isSecV sv2 Van.eqAct sv1 sv2$ 
and move-12  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ 
then obtain sv1' sv2' statO'
where 0: statO' = sstatO' statO sv1 sv2
validTransV (sv1,sv1')  $\neg isSecV sv1$ 
validTransV (sv2,sv2')  $\neg isSecV sv2$ 
Van.eqAct sv1 sv2
and  $\Delta$ :  $\Delta v w1 w2 s1 s2 statA sv1' sv2' statO'$ 
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  $\longrightarrow$  statO' = Diff
using ok 0 unfolding sstatO'-def by (cases statO, auto)

```

```

obtain  $w' w1' w2' trv1 trv2 statAA statOO$  where  $\varphi: \varphi \Delta w' w1 w2 w1' w2'$ 
 $statA s1 tr1 s2 tr2 statAA statO' sv1' trv1 sv2' trv2 statOO$ 
using  $less(2)[OF v, of tr1 tr2 w1 w2 s1 s2 statA sv1' sv2' statO', simplified,$ 
 $OF \Delta r34 r12' ok']$ 
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  $\varphi\ ok' 0 sv12 nev$  unfolding  $\varphi\text{-def}\ sstatO'\text{-def}$ 
by  $simp$  ( $smt$  ( $verit, ccfv\text{-SIG}$ )  $Statewise\text{-Attacker}\text{-Mod}\text{-eqAct}\text{-def}$ 
 $Van.A.Cons\text{-unfold}\ Van.O.Cons\text{-unfold}\ Van.Statewise\text{-Attacker}\text{-Mod}\text{-axioms}$ 
 $Van.validFromS\text{-Cons}\ list.inject\ newStat.simps(1)\ newStat.simps(4)$ )
qed
next
assume  $m: react \Delta w1 w2 s1 s2 statA sv1 sv2 statO$ 
define  $statA'$  where  $statA': statA' = sstatA' statA s1 s2$ 
show  $?thesis$ 
proof( $cases\ length\ tr1 \leq Suc\ 0$ )
case  $True$ 
hence  $tr1e: tr1 = [] \vee tr1 = [s1]$ 
by ( $metis\ Opt.validFromS\text{-singl}\text{-iff}\ Suc\text{-length}\text{-conv}\ le\text{-Suc}\text{-eq}\ le\text{-zero}\text{-eq}\ length\text{-0}\text{-conv}$ 
 $vtr1$ )
hence  $Opt.A\ tr1 = []$  by ( $simp\ add: True$ )
hence  $Opt.A\ tr2 = []$  using  $Opt.A.eq\text{-Nil}\text{-iff}\ nev4$  by  $blast$ 
show  $?thesis$ 
proof( $cases\ length\ tr2 \leq Suc\ 0$ )
case  $True$ 
hence  $tr2e: tr2 = [] \vee tr2 = [s2]$ 
by ( $metis\ Opt.validFromS\text{-def}\ Suc\text{-length}\text{-conv}\ le\text{-Suc}\text{-eq}\ le\text{-zero}\text{-eq}\ length\text{-0}\text{-conv}$ 
 $list.sel(1)\ vtr2$ )
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  $f34 \Delta$  apply ( $clarsimp\ simp: \varphi\text{-def}\ lastt\text{-def}$ )
apply( $cases\ statA, simp\text{-all}$ )
apply ( $metis\ Opt.O.simps(4)\ Opt.S.simps(4)\ last\text{-Cons}L$ )
by ( $metis\ Opt.S.simps(4)\ last.simps\ ok$ )
next
case  $False$ 
then obtain  $s24 tr2'$  where  $tr2: tr2 = s24 \# tr2'$  and  $tr2'NE: tr2' \neq []$ 
by ( $cases\ tr2, auto$ )
have  $s24[simp]: s24 = s2$  using  $\langle Opt.validFromS\ s2\ tr2 \rangle$ 
by ( $simp\ add: Opt.validFromS\text{-Cons}\text{-iff}\ tr2$ )
obtain  $s2'$  where
 $trn4: validTransO\ (s2, s2') \vee (s2 = s2' \wedge tr2' = [])$  and
 $tr2': Opt.validFromS\ s2'\ tr2'$  using  $\langle Opt.validFromS\ s2\ tr2 \rangle$ 

```

```

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
react-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  $\varphi$ -def tr2 s24
    by auto
  next
    assume sv1:  $\neg$  isSecV sv1 and trn24: eqSec sv2 s2 and
    Atrn12: Van.eqAct sv1 sv2 and match2-12  $\Delta \infty \infty s1 s2 s2'$  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
     $\Delta$ :  $\Delta \infty \infty \infty s1 s2' statA sv1' sv2' statO'$ 
    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  $\varphi$ :  $\varphi \Delta w' \infty \infty w1'$ 
    w2' statA s1 tr1 s2' tr2' statAA statO' sv1' trv1 sv2' trv2 statOO
    using less(1)[of tr1 tr2', OF -  $\Delta r3 r4' r12'$  - - - - 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  $\varphi$  O44' tr2'NE sv1
    using Van.validFromS-Cons trn1 trn2
    using isAO4 ok Atrn12 eqSec-S-Cons trn24 tr2' trn4
    unfolding  $\varphi$ -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 newStat.simps(2))
  qed
  qed
  next
  case False
  then obtain s13 tr1' where tr1: tr1 = s13 # tr1' and tr1'NE: tr1'  $\neq$  []
    by (cases tr1, auto)
  have s13[simp]: s13 = s1 using  $\langle$ Opt.validFromS s1 tr1 $\rangle$ 
    by (simp add: Opt.validFromS-Cons-iff tr1)
  obtain s1' where
    trn3: validTransO (s1,s1') and
    tr1': Opt.validFromS s1' tr1' using  $\langle$ Opt.validFromS s1 tr1 $\rangle$ 
  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:  $\neg$  finalO s1 using Opt.final-def trn3 by blast
  hence f4:  $\neg$  finalO s2 using f34 by blast
  have nev3': never isIntO (butlast tr1')
  using nev3 tr1 tr1'NE by auto
  have isAO3:  $\neg$  isIntO s1 using less.premis(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  $\Delta$  w1 w2 s1 s2 statA sv1 sv2 statO using m unfolding
react-def by auto
have ( $\exists w1' < w1. \exists w2' < w2. \neg isSecO s1 \wedge \Delta \infty w1' w2' s1' s2 statA sv1$ 
sv2 statO)  $\vee$ 
( $\exists w2' < w2. eqSec sv1 s1 \wedge \neg isIntV sv1 \wedge match1-1 \Delta \infty w2' s1 s1' s2$ 
statA sv1 sv2 statO)  $\vee$ 
(eqSec sv1 s1  $\wedge \neg isSecV sv2 \wedge Van.eqAct sv1 sv2 \wedge match1-12 \Delta \infty \infty$ 
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  $\neg isSecO s1$  and  $\Delta: \Delta \infty w1'' w2'' s1' s2 statA sv1 sv2 statO$ 
hence S3: Opt.S tr1' = Opt.S tr1 unfolding tr1 by auto
obtain w' w1' w2' trv1 trv2 statAA statOO where  $\varphi: \varphi \Delta w' w1'' w2'' w1'$ 
w2' statA s1' tr1' s2 tr2 statAA statO sv1 trv1 sv2 trv2 statOO
using less(1)[of tr1' tr2, OF -  $\Delta r3' r4' r12$ , 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  $\varphi$  O33' S3 tr1 tr1'NE tr1' trn3 w12' unfolding  $\varphi$ -def by auto
next
fix w2'' assume w2': w2'' < w2
assume trn13: eqSec sv1 s1 and
Atrn1:  $\neg isIntV sv1$  and match1-1  $\Delta \infty w2'' s1 s1' s2 statA sv1 sv2 statO$ 
then obtain sv1' where
trn1: validTransV (sv1, sv1') and
 $\Delta: \Delta \infty \infty w2'' s1' s2 statA sv1' sv2 statO$ 
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  $\varphi: \varphi \Delta w' \infty w2'' w1'$ 
w2' statA s1' tr1' s2 tr2 statAA statO sv1' trv1 sv2 trv2 statOO
using less(1)[of tr1' tr2, OF -  $\Delta r3' r4' r1' r2$ , 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  $\varphi$  O33' unfolding  $\varphi$ -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:  $\neg isSecV sv2$  and trn13: eqSec sv1 s1 and
Atrn12: Van.eqAct sv1 sv2 and match1-12  $\Delta \infty \infty s1 s1' s2 statA sv1 sv2$ 
statO
then obtain sv1' sv2' statO' where

```

$statO'$ :  $statO' = sstatO' statO sv1 sv2$  **and**  
 $trn1$ :  $validTransV (sv1, sv1')$  **and**  
 $trn2$ :  $validTransV (sv2, sv2')$  **and**  
 $\Delta$ :  $\Delta \infty \infty \infty s1' s2 statA sv1' sv2' statO'$   
**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**  $\varphi$ :  $\varphi \Delta w' \infty \infty w1'$   
 $w2' statA s1' tr1' s2 tr2 statAA statO' sv1' trv1 sv2' trv2 statOO$   
**using**  $less(1)[of tr1' tr2, OF - \Delta r3' r4 r12', unfolded O33', simplified]$   
**using**  $less.premis tr1' ok f3 f4 tr1'NE trn3 O33'(1)$  **unfolding**  $tr1 statO'$   
 $sstatO'-def$  **by**  $auto$

**have**  $trv1NE$ :  $trv1 \neq []$  **and**  $trv2NE$ :  $trv2 \neq []$  **using**  $\varphi$  **unfolding**  $\varphi-def$  **by**  
 $auto$

**have**  $[simp]$ :  $Van.O (sv1 \# trv1) = Van.O (sv2 \# trv2) \longleftrightarrow (isIntV sv1$   
 $\longrightarrow getObsV sv1 = getObsV sv2) \wedge Van.O trv1 = Van.O trv2$   
**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**  $\varphi O33' tr1'NE sv2$   
**using**  $Van.validFromS-Cons trn1 trn2$   
**using**  $isAO3 ok Atrn12 eqSec-S-Cons trn13 f3 f34 s13 tr1' trn3$   
**unfolding**  $\varphi-def tr1 Van.completedFrom-def Van.eqAct-def statO' sstatO'-def$   
**apply**  $clarsimp$

**by**  $(smt (verit, ccfv-SIG) Van.A.Cons-unfold newStat.simps(1) newStat.simps(2)$   
 $newStat.simps(4))$   
**qed**  
**qed**  
**qed**  
**qed**

**definition**  $\varphi a \Delta w w1 w2 w1' w2' statA s1 tr1 s2 tr2 statAA statO sv1 trv1 sv2$   
 $trv2 statOO \equiv$   
 $trv1 \neq [] \wedge trv2 \neq [] \wedge$   
 $(length trv1 > Suc 0 \vee w1' < w1) \wedge (length trv2 > Suc 0 \vee w2' < w2) \wedge$   
 $Van.validFromS sv1 trv1 \wedge$   
 $Van.validFromS sv2 trv2 \wedge$   
 $Van.S trv1 = Opt.S tr1 \wedge Van.S trv2 = Opt.S tr2 \wedge$   
 $Van.A trv1 = Van.A trv2 \wedge$   
 $(statO = Eq \longrightarrow (statOO = Diff \longleftrightarrow Van.O trv1 \neq Van.O trv2)) \wedge$   
 $(statA = Eq \longrightarrow (statAA = Diff \longleftrightarrow Opt.O tr1 \neq Opt.O tr2)) \wedge$   
 $\text{---}$   
 $(statO = Diff \longrightarrow statOO = Diff) \wedge$   
 $(statAA = Diff \longrightarrow statOO = Diff) \wedge$   
 $\Delta w w1' w2' (lastt s1 tr1) (lastt s2 tr2) statAA (lastt sv1 trv1) (lastt sv2 trv2)$   
 $statOO$

**lemma** *unwindCond-ex- $\varphi$ a-getActO*:  
**assumes** *unwind*: *unwindCond*  $\Delta$   
**and**  $\Delta$ :  $\Delta$  *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*  $\longrightarrow$  *statO = Diff*)  
**and** *v*: *validTransO (s1, s1')* *validTransO (s2, s2')*  
**and** *i34*: *isIntO s1 isIntO s2 getActO s1 = getActO s2*  
**shows**  $\exists w1' w2' trv1 trv2 statOO$ .  
 $\varphi a \Delta \infty w1 w2 w1' w2' statA s1 [s1, s1'] s2 [s2, s2'] (sstatA' statA s1 s2)$   
*statO sv1 trv1 sv2 trv2 statOO*  
**using**  $\Delta$  *r12 stat*  
**proof**(*induction w arbitrary: w1 w2 sv1 sv2 statO rule: less-induct*)  
**case** (*less w w1 w2 sv1 sv2 statO*)  
**note**  $\Delta = \langle \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \rangle$   
**note** *r12 = less.premis(2,3)*  
**note** *r1 = r12(1)* **note** *r2 = r12(2)*  
**note** *r = r34 r12*  
**note** *stat = \langle statA = Diff \longrightarrow statO = Diff \rangle*  
  
**have** *f34*: *finalO s1 = finalO s2*  $\wedge$  *finalV sv1 = finalO s1*  $\wedge$  *finalV sv2 = finalO s2*  
**using**  $\Delta$  *unwind[unfolded unwindCond-def]* *r* **by** *auto*  
  
**have** *proact-match*: ( $\exists v < w$ . *proact*  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ )  $\vee$  *react*  
 $\Delta w1 w2 s1 s2 statA sv1 sv2 statO$   
**using**  $\Delta$  *unwind[unfolded unwindCond-def]* *r* **by** *auto*  
**show** *?case* **using** *proact-match* **proof** *safe*  
**fix** *v* **assume** *v*: *v < w*  
**assume** *proact*  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$   
**thus** *?thesis* **unfolding** *proact-def* **proof** *safe*  
**assume** *sv1*:  $\neg isSecV sv1 \neg isIntV sv1$  **and** *move-1*  $\Delta v w1 w2 s1 s2 statA$   
*sv1 sv2 statO*  
**then obtain** *sv1'*  
**where** *0*: *validTransV (sv1,sv1')*  
**and**  $\Delta$ :  $\Delta v w1 w2 s1 s2 statA sv1' sv2 statO$   
**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**  
 $\varphi$ :  $\varphi a \Delta \infty w1 w2 w1' w2' statA s1 [s1, s1'] s2 [s2, s2'] (sstatA' statA s1 s2)$   
*statO sv1' trv1 sv2 trv2 statOO*  
**using** *less(1)[OF v  $\Delta$  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**  $\varphi 0 sv1$  **unfolding**  $\varphi a$ -*def* **apply** *simp*  
**by** (*metis Van.validFromS-Cons*)  
**next**  
**assume** *sv2*:  $\neg isSecV sv2 \neg isIntV sv2$  **and** *move-2*  $\Delta v w1 w2 s1 s2 statA$   
*sv1 sv2 statO*  
**then obtain** *sv2'*

**where**  $0$ :  $\text{validTransV } (sv2, sv2')$   
**and**  $\Delta$ :  $\Delta v w1 w2 s1 s2 \text{ statA } sv1 sv2' \text{ statO}$   
**unfolding**  $\text{move-2-def}$  **by**  $\text{auto}$   
**have**  $r2'$ :  $\text{reachV } sv2'$  **using**  $r2 0$  **by**  $(\text{metis Van.reach.Step fst-conv snd-conv})$   
**obtain**  $w1' w2' \text{ trv1 trv2 statOO}$  **where**  
 $\varphi$ :  $\varphi a \Delta \infty w1 w2 w1' w2' \text{ statA } s1 [s1, s1'] s2 [s2, s2'] (\text{sstatA}' \text{ statA } s1 s2)$   
 $\text{statO } sv1 \text{ trv1 } sv2' \text{ trv2 } \text{statOO}$   
**using**  $\text{less}(1)[OF v \Delta r1 r2' \text{ stat}]$  **by**  $\text{auto}$   
**show**  $?thesis$  **apply** $(\text{rule exI}[of - w1'])$  **apply** $(\text{rule exI}[of - w2'])$  **apply** $(\text{rule exI}[of - trv1])$  **apply** $(\text{rule exI}[of - sv2 \# trv2])$   
**using**  $\varphi 0 sv2$  **unfolding**  $\varphi a\text{-def}$  **apply**  $\text{simp}$  **by**  $(\text{metis Van.validFromS-Cons})$   
**next**  
**assume**  $sv12$ :  $\neg \text{isSecV } sv1 \neg \text{isSecV } sv2 \text{ Van.eqAct } sv1 sv2$   
**and**  $\text{move-12 } \Delta v w1 w2 s1 s2 \text{ statA } sv1 sv2 \text{ statO}$   
**then obtain**  $sv1' sv2' \text{ statO}'$   
**where**  $0$ :  $\text{statO}' = \text{sstatO}' \text{ statO } sv1 sv2$   
 $\text{validTransV } (sv1, sv1') \neg \text{isSecV } sv1$   
 $\text{validTransV } (sv2, sv2') \neg \text{isSecV } sv2$   
 $\text{Van.eqAct } sv1 sv2$   
**and**  $\Delta$ :  $\Delta v w1 w2 s1 s2 \text{ statA } sv1' sv2' \text{ statO}'$   
**unfolding**  $\text{move-12-def}$  **by**  $\text{auto}$   
**have**  $r12'$ :  $\text{reachV } sv1' \text{ reachV } sv2'$  **using**  $r1 r2 0$  **by**  $(\text{metis Van.reach.Step fst-conv snd-conv})+$   
**have**  $\text{stat}'$ :  $\text{statA} = \text{Diff} \longrightarrow \text{statO}' = \text{Diff}$   
**using**  $\text{stat } 0$  **unfolding**  $\text{sstatO}'\text{-def}$  **by**  $(\text{cases statO}, \text{auto})$   
**obtain**  $w1' w2' \text{ trv1 trv2 statOO}$  **where**  
 $\varphi$ :  $\varphi a \Delta \infty w1 w2 w1' w2' \text{ statA } s1 [s1, s1'] s2 [s2, s2'] (\text{sstatA}' \text{ statA } s1 s2)$   
 $\text{statO}' sv1' \text{ trv1 } sv2' \text{ trv2 } \text{statOO}$   
**using**  $\text{less}(1)[OF v \Delta r12' \text{ stat}']$  **unfolding**  $\varphi a\text{-def}$  **apply**  $\text{simp}$  **by**  $\text{metis}$   
**show**  $?thesis$  **apply** $(\text{rule exI}[of - w1'])$  **apply** $(\text{rule exI}[of - w2'])$  **apply** $(\text{rule exI}[of - sv1 \# trv1])$  **apply** $(\text{rule exI}[of - sv2 \# trv2])$   
**using**  $\varphi 0$  **unfolding**  $\varphi a\text{-def sstatO}'\text{-def}$  **apply**  $\text{clarsimp}$  **apply** $(\text{intro conjI})$   
**subgoal** **by**  $\text{auto}$   
**subgoal** **by**  $\text{auto}$   
**subgoal** **by**  $(\text{metis Van.A.Cons-unfold Van.eqAct-def})$   
**subgoal** **apply** $(\text{rule exI}[of - statOO])$  **apply**  $\text{simp}$   
**by**  $(\text{smt (verit, ccfv-threshold) Van.O.Cons-unfold Van.eqAct-def list.inject newStat.simps(1) newStat.simps(3)})$  .  
**qed**  
**next**  
**assume**  $m$ :  $\text{react } \Delta w1 w2 s1 s2 \text{ statA } sv1 sv2 \text{ statO}$   
**define**  $\text{statA}'$  **where**  $\text{statA}'$ :  $\text{statA}' = \text{sstatA}' \text{ statA } s1 s2$   
**have**  $m$ :  $\text{match12 } \Delta w1 w2 s1 s2 \text{ statA } sv1 sv2 \text{ statO}$  **using**  $m$  **unfolding**  
 $\text{react-def}$  **by**  $\text{auto}$   
**have**  $(\exists w1' w2'. w1' < w1 \wedge w2' < w2 \wedge \neg \text{isSecO } s1 \wedge \neg \text{isSecO } s2 \wedge (\text{statA} = \text{statA}' \vee \text{statO} = \text{Diff})) \wedge$   
 $\Delta \infty w1' w2' s1' s2' \text{ statA}' sv1 sv2 \text{ statO})$   
 $\vee$   
 $(\exists w2' < w2. \neg \text{isSecO } s2 \wedge$

```

    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)
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 tr1NE: 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**  $\exists w1'\ w2'\ trv1'\ trv2'\ statAA'\ statOO'$ .  
 $\varphi a\ \Delta\ \infty\ w1\ w2\ w1'\ w2'\ statA\ s1\ (tr1\ \#\#\ s1')\ s2\ (tr2\ \#\#\ s2')\ statAA'\ statO$   
 $sv1\ trv1'\ sv2\ trv2'\ statOO'$   
**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**  $\varphi$ :  $\varphi\ \Delta\ ww\ w1\ w2\ ww1$   
 $ww2\ statA\ s1\ tr1\ s2\ tr2\ statAA\ statO\ sv1\ trv1\ sv2\ trv2\ statOO$   
**using**  $unwindCond-ex-\varphi[OF\ unwind\ \Delta\ r\ stat\ v3\ v4\ i\ nev]$  **by**  $auto$   
  
**have**  $trv12NE$ :  $trv1 \neq []\ trv2 \neq []$  **using**  $\varphi$  **unfolding**  $\varphi-def$  **by**  $auto$   
  
**define**  $ss1\ ss2\ ssv1\ ssv2$  **where**  $ss1$ :  $ss1 \equiv lastt\ s1\ tr1$  **and**  $ss2$ :  $ss2 \equiv lastt\ s2$   
 $tr2$   
**and**  $ssv1$ :  $ssv1 \equiv lastt\ sv1\ trv1$  **and**  $ssv2$ :  $ssv2 \equiv lastt\ sv2\ trv2$   
  
**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**  $\varphi$  **unfolding**  $\varphi-def$  **by** ( $metis\ lastt-def\ ssv1$ )  
**have**  $trv1l$ :  $trv1 = butlast\ trv1\ @\ [ssv1]$  **by** ( $simp\ add: ssv1l\ trv12NE(1)$ )  
**have**  $ssv2l$ :  $ssv2 = last\ trv2$  **using**  $\varphi$  **unfolding**  $\varphi-def$  **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** ( $metis\ Opt.O-def\ \langle isIntO\ ss1 \rangle\ holds-filtermap-RCons\ snoc-eq-iff-butlast\ tr1l$ )  
**have**  $[simp]$ :  $Opt.O (tr2\ \#\#\ s2') = Opt.O\ tr2\ \#\#\ getObsO\ ss2$   
**by** ( $metis\ Opt.O-def\ \langle isIntO\ ss2 \rangle\ holds-filtermap-RCons\ snoc-eq-iff-butlast\ tr2l$ )  
  
**have**  $[simp]$ :  $Opt.A (tr1\ \#\#\ s1') = Opt.A\ tr1\ \#\#\ getActO\ ss1$   
**by** ( $metis\ Opt.A-def\ \langle isIntO\ ss1 \rangle\ holds-filtermap-RCons\ snoc-eq-iff-butlast\ tr1l$ )  
**have**  $[simp]$ :  $Opt.A (tr2\ \#\#\ s2') = Opt.A\ tr2\ \#\#\ getActO\ ss2$   
**by** ( $metis\ Opt.A-def\ \langle isIntO\ ss2 \rangle\ holds-filtermap-RCons\ snoc-eq-iff-butlast\ tr2l$ )  
**have**  $[simp]$ :  $Opt.A (tr1\ \#\#\ s1') = Opt.A (tr2\ \#\#\ s2') \longleftrightarrow Opt.A\ tr1 = Opt.A$   
 $tr2$  **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**  $Van.reach-validFromS-reach\ \varphi-def\ \varphi\ r(3)\ ssv1l$   
**apply**  $(smt\ (verit,\ del-insts))$   
**using**  $Van.reach-validFromS-reach\ \varphi-def\ \varphi\ r(4)\ ssv2l$   
**apply**  $(smt\ (verit,\ del-insts))$  .

**have**  $stat: statAA = Diff \longrightarrow statOO = Diff$   
**and**  $\Delta: \Delta\ ww\ ww1\ ww2\ ss1\ ss2\ statAA\ ssv1\ ssv2\ statOO$   
**using**  $\varphi$  **unfolding**  $\varphi-def\ ss1[symmetric]\ ss2[symmetric]\ ssv1[symmetric]\ ssv2[symmetric]$   
**by**  $auto$

**note**  $vs13 = s13'[unfolded\ ss1[symmetric]]$  **note**  $vs24 = s24'[unfolded\ ss2[symmetric]]$   
**have**  $\exists\ w1'\ w2'\ trv1'\ trv2'\ statA'\ statO'$   
 $\varphi a\ \Delta\ \infty\ ww1\ ww2\ w1'\ w2'\ statAA\ ss1\ [ss1,s1']\ ss2\ [ss2,s2']\ (sstata'\ statAA\ ss1\ ss2)\ statOO\ ssv1\ trv1'\ ssv2\ trv2'\ statO'$   
**using**  $unwindCond-ex-\varphi a-getActO[OF\ unwind\ \Delta\ rss\ stat\ vs13\ vs24\ iss14\ giss14]$   
**by**  $blast$

**then obtain**  $w1'\ w2'\ trv1'\ trv2'\ statA'\ statO'$  **where**  
 $\varphi 1: \varphi a\ \Delta\ \infty\ ww1\ ww2\ w1'\ w2'\ statAA\ ss1\ [ss1,s1']\ ss2\ [ss2,s2']\ statA'\ statOO\ ssv1\ trv1'\ ssv2\ trv2'\ statO'$  **by**  $auto$

**have**  $trv12'NE: trv1' \neq [] \wedge trv2' \neq []$  **using**  $\varphi 1$  **unfolding**  $\varphi a-def$  **by**  $auto$

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

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

**have**  $Van.A\ trv1' = Van.A\ trv2'$  **using**  $\varphi 1$  **unfolding**  $\varphi a-def$  **by**  $auto$   
**moreover have**  $length\ (Van.O\ trv1') = length\ (Van.A\ trv1') \wedge length\ (Van.O\ trv2') = length\ (Van.A\ trv2')$   
**unfolding**  $Van.A.map-filter\ Van.O.map-filter$  **by**  $auto$   
**ultimately have**  $length\ (Van.O\ trv1') = length\ (Van.O\ trv2')$  **by**  $auto$   
**hence**  $[simp]: Van.O\ trv1\ @\ Van.O\ trv1' = Van.O\ trv2\ @\ Van.O\ trv2' \longleftrightarrow Van.O\ trv1 = Van.O\ trv2 \wedge Van.O\ trv1' = Van.O\ trv2'$  **by**  $auto$

**have**  $len: trv1 \neq [] \wedge trv2 \neq [] \wedge trv1' \neq [] \wedge trv2' \neq [] \wedge$   
 $(Suc\ 0 < length\ trv1 \vee ww1 \leq w1) \wedge$   
 $(Suc\ 0 < length\ trv1' \vee w1' < ww1) \wedge$   
 $(Suc\ 0 < length\ trv2 \vee ww2 \leq w2) \wedge$   
 $(Suc\ 0 < length\ trv2' \vee w2' < ww2)$   
**using**  $\varphi\ \varphi 1$  **unfolding**  $\varphi-def\ \varphi a-def$  **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  $\varphi$ a-def apply(intro conjI)
  subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-def by auto
  subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-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 or-
der-le-less-trans trans-less-add2)
subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-def ssv1
  using Van.validFromS-append by auto
subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-def ssv2
  using Van.validFromS-append by auto
subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-def Van.S.map-filter Opt.S.map-filter

  apply(subst tr1l) apply(subst butlast-append) by simp
subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-def Van.S.map-filter Opt.S.map-filter

  apply(subst tr2l) apply(subst butlast-append) by simp
subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-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  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-def apply simp
  apply(cases Opt.O tr1 = Opt.O tr2, simp-all) apply clarify
    using status.exhaust by (metis (full-types))+
subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-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  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-def by simp
subgoal using  $\varphi$   $\varphi$ 1 unfolding  $\varphi$ -def  $\varphi$ a-def by simp
subgoal using  $\varphi$ 1 trv12'NE tr14NE unfolding  $\varphi$ -def  $\varphi$ a-def lastt-def by simp
.
qed

```

```

lemma unwindCond-ex- $\varphi$ a-aux2:
assumes unwind: unwindCond  $\Delta$ 
and  $\Delta$ :  $\Delta$  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**  $\exists w1'\ w2'\ trv1\ trv2\ statAA\ statOO$ .  
 $\varphi a\ \Delta\ \infty\ w1\ w2\ w1'\ w2'\ statA\ s1\ (tr1\ @\ [s1', s1''])\ s2\ (tr2\ @\ [s2', s2''])\ statAA$   
 $statO\ sv1\ trv1\ sv2\ trv2\ statOO$   
**proof** –  
**have**  $0$ :  $lastt\ s1\ (tr1\ ##\ s1') = s1'\ lastt\ s2\ (tr2\ ##\ s2') = s2'$   
**unfolding**  $lastt\text{-}def$  **by**  $auto$   
**show**  $?thesis$   
**apply**( $rule\ unwindCond\text{-}ex\text{-}\varphi a'\text{-}aux[OF\ unwind\ \Delta\ r\ stat,\ of\ tr1\ ##\ s1'\ tr2\ ##\ s2',\ unfolded\ 0,\ simplified]$ )  
**using**  $assms$  **by**  $auto$   
**qed**

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

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

**lemma**  $append\text{-}snoc2$ :  $tr1\ @\ [s1', s1''] = (tr1\ ##\ s1')\ ##\ s1''$   
**by**  $auto$

**definition**  $\varphi'$   $\Delta\ w1\ w2\ w1'\ w2'\ statA\ s1\ tr1\ s1'\ s1''\ s2\ tr2\ s2'\ s2''\ statAA\ statO$   
 $sv1\ trv1\ sv1''\ sv2\ trv2\ sv2''\ statOO \equiv$   
 $(trv1 \neq [] \vee w1' < w1) \wedge (trv2 \neq [] \vee w2' < w2) \wedge$   
 $Van.validFromS\ sv1\ (trv1\ ##\ sv1'') \wedge Van.validFromS\ sv2\ (trv2\ ##\ sv2'') \wedge$   
 $Van.S\ (trv1\ ##\ sv1'') = Opt.S\ ((tr1\ ##\ s1')\ ##\ s1'') \wedge Van.S\ (trv2\ ##\ sv2'')$   
 $= Opt.S\ ((tr2\ ##\ s2')\ ##\ s2'') \wedge$   
 $Van.A\ (trv1\ ##\ sv1'') = Van.A\ (trv2\ ##\ sv2'') \wedge$   
 $(statO = Eq \longrightarrow (statOO = Diff) = (Van.O\ (trv1\ ##\ sv1'') \neq Van.O\ (trv2\ ##\ sv2'')))$   
 $\wedge$   
 $(statA = Eq \longrightarrow (statAA = Diff) = (Opt.O\ ((tr1\ ##\ s1')\ ##\ s1'') \neq Opt.O\ ((tr2\ ##\ s2')\ ##\ s2'')))$   
 $\wedge$   
 $(statO = Diff \longrightarrow statOO = Diff) \wedge (statAA = Diff \longrightarrow statOO = Diff) \wedge$   
 $\Delta\ \infty\ w1'\ w2'\ s1''\ s2''\ statAA\ sv1''\ sv2''\ statOO$

**proposition**  $unwindCond\text{-}ex\text{-}\varphi'$ :  
**assumes**  $unwind$ :  $unwindCond\ \Delta$  **and**  $\Delta$ :  $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**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**  $\exists w1' w2' trv1 sv1'' trv2 sv2'' statAA statOO$ .  
 $\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO sv1 trv1 sv1'' sv2 trv2 sv2'' statOO$   
**using** *unwindCond-ex- $\varphi$ -aux2*[*unfolded  $\varphi$ -def*, *unfolded lastt-snoc lastt-snoc2 append-snoc2*, *OF assms*]  
**unfolding**  *$\varphi$ -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**  *$\varphi'$ -def* **apply** *simp* **by** *blast* . .

**definition**  $\chi^3 \Delta w (w1::enat) w2 w1' w2' s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO \equiv$   
 $trv1 \neq [] \wedge trv2 \neq [] \wedge (length\ trv2 > Suc\ 0 \vee w2' \leq w2) \wedge$   
 $Van.validFromS\ sv1\ trv1 \wedge Van.validFromS\ sv2\ trv2 \wedge$   
 $never\ isSecV\ (butlast\ trv1) \wedge$   
 $isSecV\ (lastt\ sv1\ trv1) \wedge getSecV\ (lastt\ sv1\ trv1) = getSecO\ (lastt\ s1\ tr1) \wedge$   
 $never\ isSecV\ (butlast\ trv2) \wedge$   
 $Van.A\ trv1 = Van.A\ trv2 \wedge$   
 $\Delta\ w\ w1'\ w2'\ (lastt\ s1\ tr1)\ s2\ statAA\ (lastt\ sv1\ trv1)\ (lastt\ sv2\ trv2)\ statOO$

**lemma**  *$\chi^3$ -final*:

**assumes** *unw*: *unwindCond*  $\Delta$   
**and** *r*: *reachO* *s1* *reachO* *s2* *reachV* *sv1* *reachV* *sv2*  
**and** *vtr1*: *Opt.validFromS* *s1* *tr1*  
**and**  $\chi^3$ :  $\chi^3 \Delta w w1 w2 w1' w2' s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO$   
**shows** (*finalV* (*lastt* *sv1* *trv1*)  $\longleftrightarrow$  *finalO* (*lastt* *s1* *tr1*))  $\wedge$  (*finalV* (*lastt* *sv2* *trv2*)  $\longleftrightarrow$  *finalO* *s2*)

**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** *rs1*: *Opt.validFromS* *s1* *tr1*  $\longrightarrow$  *reachO* (*lastt* *s1* *tr1*)  
**using** *r*  
**by** (*simp* *add*: *Opt.reach-validFromS-reach* *lastt-def*)  
**show** *?thesis* **using**  $\chi^3$ [*unfolded  $\chi^3$ -def*] *rsv12* *rs1* **using** *unw*[*unfolded unwindCond-def*, *rule-format*,  
*of lastt s1 tr1 s2 lastt sv1 trv1 lastt sv2 trv2 w w1' w2' statAA statOO*]  
**using** *vtr1*  $\langle reachO\ s2 \rangle$  **by** *auto*  
**qed**

**lemma**  *$\chi^3$ -completedFrom*: *unwindCond*  $\Delta \implies$

*reachO* *s1*  $\implies$  *reachO* *s2*  $\implies$  *reachV* *sv1*  $\implies$  *reachV* *sv2*  $\implies$   
*Opt.validFromS* *s1* *tr1*  $\implies$  *completedFromO* *s1* *tr1*  $\implies$   
 $\chi^3 \Delta w w1 w2 w1' w2' s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO$

$\implies \text{completedFromV } sv1 \text{ } trv1 \wedge \text{completedFromV } sv2 \text{ } trv2$   
**by** (*metis Van.final-not-isSec*  $\chi^3$ -def  $\chi^3$ -final *completedFromO-lastt*)

**lemma** *unwindCond-ex- $\chi^3$* :

**assumes** *unwind*: *unwindCond*  $\Delta$

**and**  $\Delta$ :  $\Delta \ w \ w1 \ w2 \ s1 \ s2 \ statA \ sv1 \ sv2 \ statO$

**and** *r*: *reachO* *s1* *reachO* *s2* *reachV* *sv1* *reachV* *sv2*

**and** *vtr1*: *Opt.validFromS* *s1* *tr1*

**and** *nis1*:  $\neg \text{isIntO } s1$  **and** *nis2*:  $\neg \text{isIntO } s2$

**and** *inter3*: *never isIntO* *tr1*

**and** *sec*: *never isSecO* (*butlast* *tr1*) *isSecO* (*lastt* *s1* *tr1*)

**shows**  $\exists w' \ w1' \ w2' \ trv1 \ trv2 \ statOO. \chi^3 \ \Delta \ w' \ w1 \ w2 \ w1' \ w2' \ s1 \ tr1 \ s2 \ statA \ sv1 \ trv1 \ sv2 \ trv2 \ statOO$

**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**  $\Delta = \langle \Delta \ w \ w1 \ w2 \ s1 \ s2 \ statA \ sv1 \ sv2 \ statO \rangle$

**note** *nis1* = *less*(9) **note** *nis2* = *less*(10)

**note** *inter3* = *less*(11)

**note** *sec3* = *less*(12,13)

**note** *r34* = *less.prem*s(2,3) **note** *r12* = *less.prem*s(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*  $\implies \text{isIntO } s1 = \text{isIntO } s2$

**and** *f34*: *finalO* *s1* = *finalO* *s2*  $\wedge$  *finalV* *sv1* = *finalO* *s1*  $\wedge$  *finalV* *sv2* = *finalO* *s2*

**using**  $\Delta$  *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

**have** *proact-match*: ( $\exists v < w. \text{proact } \Delta \ v \ w1 \ w2 \ s1 \ s2 \ statA \ sv1 \ sv2 \ statO$ )  $\vee$  *react*  $\Delta \ w1 \ w2 \ s1 \ s2 \ statA \ sv1 \ sv2 \ statO$

**using**  $\Delta$  *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*

**show** *?case* **using** *proact-match* **proof** *safe*

**fix** *v* **assume** *v*: *v* < *w*

**assume** *proact*  $\Delta \ v \ w1 \ w2 \ s1 \ s2 \ statA \ sv1 \ sv2 \ statO$

**thus** *?thesis* **unfolding** *proact-def* **proof** *safe*

**assume** *sv1*:  $\neg \text{isSecV } sv1 \ \neg \text{isIntV } sv1$  **and** *move-1*  $\Delta \ v \ w1 \ w2 \ s1 \ s2 \ statA \ sv1 \ sv2 \ statO$

**then obtain** *sv1'*

**where** *0*:*validTransV* (*sv1*,*sv1'*)

**and**  $\Delta$ :  $\Delta \ v \ w1 \ w2 \ s1 \ s2 \ statA \ sv1' \ sv2 \ statO$

**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**  $\chi^3$ :  $\chi^3 \ \Delta \ w' \ w1 \ w2 \ w1' \ w2' \ s1 \ tr1 \ s2 \ statA \ sv1' \ trv1 \ sv2 \ trv2 \ statOO$

**using** *less*(2)[*OF* *v*, *of* *tr1* *w1* *w2* *s1* *s2* *statA* *sv1'* *sv2* *statO*,

```

      simplified, OF  $\Delta$  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  $\chi^3$  0 sv1 unfolding  $\chi^3$ -def by auto
  next
    assume sv2:  $\neg$  isSecV sv2  $\neg$  isIntV sv2 and move-2  $\Delta$  v w1 w2 s1 s2 statA
sv1 sv2 statO
    then obtain sv2'
    where 0: validTransV (sv2,sv2')
    and  $\Delta$ :  $\Delta$  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  $\chi^3$ :  $\chi^3$   $\Delta$  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  $\Delta$  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  $\chi^3$  0 sv2 unfolding  $\chi^3$ -def by auto
  next
    assume sv12:  $\neg$  isSecV sv1  $\neg$  isSecV sv2 Van.eqAct sv1 sv2
    and move-12  $\Delta$  v w1 w2 s1 s2 statA sv1 sv2 statO
    then obtain sv1' sv2' statO'
    where 0: statO' = sstatO' statO sv1 sv2
      validTransV (sv1,sv1')  $\neg$  isSecV sv1
      validTransV (sv2,sv2')  $\neg$  isSecV sv2
      Van.eqAct sv1 sv2
    and  $\Delta$ :  $\Delta$  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  $\chi^3$ :  $\chi^3$   $\Delta$  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  $\Delta$  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  $\chi^3$  0 sv12 unfolding  $\chi^3$ -def sstatO'-def
    by (auto simp: Van.eqAct-def)
  qed
next
  assume m: react  $\Delta$  w1 w2 s1 s2 statA sv1 sv2 statO
  define statA' where statA': statA' = sstatA' statA s1 s2
  show ?thesis
  proof(cases length tr1  $\leq$  Suc 0)
  case True
  hence tr1e: tr1 = []  $\vee$  tr1 = [s1]

```

**by** (*metis* *Opt.validFromS-singl-iff* *Suc-length-conv* *le-Suc-eq* *le-zero-eq* *length-0-conv* *utr1*)  
**hence** *Opt.A* *tr1* = [] **by** (*simp* *add: True*)  
**have** *is1: isSecO* *s1*  
**by** (*metis* *last.simps* *lastt-def* *sec3(2)* *tr1e*)  
**hence**  $\neg$  *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*  $\wedge$  *getSecV* *sv1* = *getSecO* *s1* **using** *m* *is1* *nis1*  
**unfolding** *react-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*  $\Delta$  **by** (*clarsimp* *simp:  $\chi$ 3-def* *lastt-def*)  
**next**  
**case** *False*  
**then obtain** *s13* *tr1'* **where** *tr1: tr1* = *s13* # *tr1'* **and** *tr1'NE: tr1'  $\neq$*  []  
**by** (*cases* *tr1*, *auto*)  
**have** *s13[simp]: s13* = *s1* **using**  $\langle$ *Opt.validFromS* *s1* *tr1* $\rangle$   
**by** (*simp* *add: Opt.validFromS-Cons-iff* *tr1*)  
**obtain** *s1'* **where**  
*trn3: validTransO* (*s1*, *s1'*) **and**  
*tr1': Opt.validFromS* *s1'* *tr1'* **using**  $\langle$ *Opt.validFromS* *s1* *tr1* $\rangle$   
**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:  $\neg$  finalO* *s1* **using** *Opt.final-def* *trn3* **by** *blast*  
**hence** *f4:  $\neg$  finalO* *s2* **using** *f34* **by** *blast*  
**have** *nev3': never isIntO* *tr1'*  
**using** *inter3* *tr1* *tr1'NE* **by** *auto*  
**have** *isAO3:  $\neg$  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*  $\Delta$  *w1* *w2* *s1* *s2* *statA* *sv1* *sv2* *statO* **using** *m* **unfolding**  
*react-def* **by** *auto*  
**have** ( $\exists w1' < w1. \exists w2' < w2. \neg isSecO$  *s1*  $\wedge$   $\Delta \infty w1' w2' s1' s2$  *statA* *sv1* *sv2* *statO*)  $\vee$   
( $\exists w2' < w2. eqSec$  *sv1* *s1*  $\wedge$   $\neg isIntV$  *sv1*  $\wedge$  *match1-1*  $\Delta \infty w2' s1 s1' s2$  *statA* *sv1* *sv2* *statO*)  $\vee$   
(*eqSec* *sv1* *s1*  $\wedge$   $\neg isSecV$  *sv2*  $\wedge$  *Van.eqAct* *sv1* *sv2*  $\wedge$  *match1-12*  $\Delta \infty \infty$  *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**  $\neg isSecO$  *s1* **and**  $\Delta: \Delta \infty w1'' w2'' s1' s2$  *statA* *sv1* *sv2* *statO*  
**hence** *S3: Opt.S* *tr1'* = *Opt.S* *tr1* **unfolding** *tr1* **by** *auto*  
**obtain** *w' w1' w2' trv1 trv2 statOO* **where**  $\chi$ 3:  $\chi$ 3  $\Delta w' w1'' w2'' w1' w2'$  *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.premis 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.premis 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**  $\chi^3$  *O33' tr1'NE sv2*  
**using** *Van.validFromS-Cons trn1 trn2*  
**using** *isAO3 Atrn12 eqSec-S-Cons trn13 f3 f34 s13 tr1' trn3*  
**unfolding**  $\chi^3$ -*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**  $\chi^3a$  **where**  $\chi^3a \Delta w (w1::enat) w2 w1' w2' s1 s1' s2 statAA sv1 trv1 sv2 trv2 statOO \equiv$   
 $trv1 \neq [] \wedge trv2 \neq [] \wedge (length\ trv2 > Suc\ 0 \vee w2' < w2) \wedge$   
 $Van.validFromS\ sv1\ trv1 \wedge Van.validFromS\ sv2\ trv2 \wedge$   
 $Van.S\ trv1 = [getSecO\ s1] \wedge$   
 $never\ isSecV\ (butlast\ trv2) \wedge$   
 $Van.A\ trv1 = Van.A\ trv2 \wedge$   
 $\Delta\ w\ w1'\ w2'\ s1'\ s2\ statAA\ (lastt\ sv1\ trv1)\ (lastt\ sv2\ trv2)\ statOO$

**lemma** *unwindCond-ex- $\chi^3a$ -getSec*:  
**assumes** *unwind: unwindCond  $\Delta$*   
**and**  $\Delta: \Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**and**  $r^3_4: reachO\ s1\ reachO\ s2$  **and**  $r^{12}: reachV\ sv1\ reachV\ sv2$   
**and**  $v: validTransO\ (s1, s1')$   
**and**  $ii^3: \neg isIntO\ s1$   
**and**  $is^1: isSecO\ s1$  **and**  $isv^{13}: isSecV\ sv1\ getSecO\ s1 = getSecV\ sv1$   
**shows**  $\exists w1'\ w2'\ trv1\ trv2\ statOO.$   
 $\chi^3a \Delta \infty w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv2 trv2 statOO$   
**using**  $\Delta\ r^{12}\ isv^{13}$   
**proof**(*induction w arbitrary: w1 w2 sv1 sv2 statO rule: less-induct*)  
**case** (*less w w1 w2 sv1 sv2 statO*)  
**note**  $\Delta = \langle \Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \rangle$   
**note**  $r^{12} = less.prem(2,3)$   
**note**  $r^1 = r^{12}(1)$  **note**  $r^2 = r^{12}(2)$   
**note**  $r = r^3_4\ r^{12}$   
**note**  $isv^{13} = \langle isSecV\ sv1 \rangle \langle getSecO\ s1 = getSecV\ sv1 \rangle$   
  
**have**  $f^3_4: finalO\ s1 = finalO\ s2 \wedge finalV\ sv1 = finalO\ s1 \wedge finalV\ sv2 = finalO\ s2$   
**using**  $\Delta\ unwind[unfolded\ unwindCond-def]$  **r** **by** *auto*

**have** *proact-match*:  $(\exists v < w. proact\ \Delta\ v\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO) \vee react$   
 $\Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**using**  $\Delta\ unwind[unfolded\ unwindCond-def]$  **r** **by** *auto*  
**show** *?case* **using** *proact-match* **proof** *safe*  
**fix**  $v$  **assume**  $v: v < w$

```

assume proact  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ 
thus ?thesis unfolding proact-def proof safe
  assume  $sv1: \neg isSecV sv1 \neg isIntV sv1$  and move-1  $\Delta v w1 w2 s1 s2 statA$ 
sv1 sv2 statO
  hence False using isv13 by blast
  thus ?thesis by auto
next
  assume  $sv2: \neg isSecV sv2 \neg isIntV sv2$  and move-2  $\Delta v w1 w2 s1 s2 statA$ 
sv1 sv2 statO
  then obtain  $sv2'$ 
  where  $0: validTransV (sv2, sv2')$ 
  and  $\Delta: \Delta v w1 w2 s1 s2 statA sv1 sv2' statO$ 
  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
   $\chi3a: \chi3a \Delta \infty w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv2' trv2 statOO$ 
  using less(1)[OF v  $\Delta r1 r2' isv13$ ] 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  $\chi3a 0 sv2$  unfolding  $\chi3a-def$  by auto
next
  assume  $sv12: \neg isSecV sv1 \neg isSecV sv2 Van.eqAct sv1 sv2$ 
  and move-12  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ 
  hence False using isv13 by blast
  thus ?thesis by auto
qed
next
  assume  $m: react \Delta w1 w2 s1 s2 statA sv1 sv2 statO$ 
  have  $m: match1 \Delta w1 w2 s1 s2 statA sv1 sv2 statO$  using  $m$  unfolding
react-def by auto
  have  $(\exists w1' w2'. w1' < w1 \wedge w2' < w2 \wedge \neg isSecO s1 \wedge \Delta \infty w1' w2' s1' s2$ 
statA sv1 sv2 statO) \vee
   $(\exists w2' < w2. eqSec sv1 s1 \wedge \neg isIntV sv1 \wedge match1-1 \Delta \infty w2' s1 s1' s2$ 
statA sv1 sv2 statO) \vee
   $(eqSec sv1 s1 \wedge \neg isSecV sv2 \wedge Van.eqAct sv1 sv2 \wedge match1-12 \Delta \infty \infty$ 
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 -  $\infty$ ]) 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  $\chi3a-def$ 
  by (auto simp : sstatA'-def lastt-def) .
  subgoal apply(rule exI[of -  $\infty$ ]) apply(rule exI[of -  $\infty$ ])
  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  $\chi^3a$ -def
by (auto simp : sstatA'-def sstatO'-def lastt-def Van.eqAct-def) . .
qed
qed

```

**definition**  $\chi^3b$   $\Delta w (w1::enat) w2 w1' w2' s1 tr1 s2 statAA sv1 trv1 sv2 trv2$   
 $statOO \equiv$   
 $trv1 \neq [] \wedge$   
 $trv2 \neq [] \wedge (length\ trv2 > Suc\ 0 \vee w2' < w2) \wedge$   
 $Van.validFromS\ sv1\ trv1 \wedge$   
 $Van.validFromS\ sv2\ trv2 \wedge$   
 $Van.S\ trv1 = Opt.S\ tr1 \wedge$   
 $never\ isSecV\ (butlast\ trv2) \wedge Van.A\ trv1 = Van.A\ trv2 \wedge$   
 $\Delta w\ w1'\ w2'\ (lastt\ s1\ tr1)\ s2\ statAA\ (lastt\ sv1\ trv1)\ (lastt\ sv2\ trv2)\ statOO$

**lemma** *unwindCond-ex- $\chi^3b$ -aux*:

**assumes** *unwind*: *unwindCond*  $\Delta$

**and**  $\Delta$ :  $\Delta w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$  **and**

*r*: *reachO* *s1* *reachO* *s2* *reachV* *sv1* *reachV* *sv2*

**and** *tr1NE*:  $tr1 \neq []$

**and** *v3'*: *Opt.validFromS* *s1* (*tr1* ## *s1'*)

**and** *nis1*:  $\neg isIntO\ s1$  **and** *nis2*:  $\neg isIntO\ s2$

**and** *ninter3'*: *never isIntO* (*tr1* ## *s1'*)

**and** *sec*: *never isSecO* (*butlast* *tr1*) *isSecO* (*lastt* *s1* *tr1*)

**shows**  $\exists w1'\ w2'\ trv1\ trv2\ statOO. \chi^3b\ \Delta\ \infty\ w1\ w2\ w1'\ w2'\ s1\ (tr1\ \#\#\ s1')\ s2$   
 $statA\ sv1\ trv1\ sv2\ trv2\ statOO$

**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'*:  $\neg isIntO\ s1'$

**using** *ninter3'* **by** *auto*

**obtain** *ww* *ww1* *ww2* *trv1* *trv2* *statOO* **where**  $\chi^3$ :  $\chi^3\ \Delta\ ww\ w1\ w2\ ww1\ ww2\ s1$   
 $tr1\ s2\ statA\ sv1\ trv1\ sv2\ trv2\ statOO$

**using** *unwindCond-ex- $\chi^3$* [*OF* *unwind*  $\Delta$  *r* *v3* *nis1* *nis2* *ninter3* *sec*] **by** *auto*

**have** *trv12NE*:  $trv1 \neq []\ trv2 \neq []$  **using**  $\chi^3$  **unfolding**  $\chi^3$ -def **by** *auto*

**define** *ss1* *ssv1* *ssv2* **where** *ss1*:  $ss1 \equiv lastt\ s1\ tr1$

**and** *ssv1*:  $ssv1 \equiv lastt\ sv1\ trv1$  **and** *ssv2*:  $ssv2 \equiv lastt\ sv2\ trv2$

**have** *ss1l*:  $ss1 = last\ tr1$  **by** (*simp* *add*: *lastt-def* *ss1* *tr1NE*)

**have** *tr1l*:  $tr1 = butlast\ tr1\ @\ [ss1]$  **by** (*simp* *add*: *ss1* *tr1NE*)

**have** *ssv1l*: *ssv1 = last trv1 using  $\chi^3$  unfolding  $\chi^3$ -def by (metis lastt-def ssv1)*  
**have** *trv1l*: *trv1 = butlast trv1 @ [ssv1] by (simp add: ssv1l trv12NE(1))*  
**have** *ssv2l*: *ssv2 = last trv2 using  $\chi^3$  unfolding  $\chi^3$ -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**  $\chi^3$  **unfolding**  $\chi^3$ -*def ssv1 ss1 by auto*

**have** *niss1*:  $\neg$  *isIntO ss1*  
**using** *ninter3 tr1l apply (auto simp add: list-all-iff)*  
**using** *in-set-conv-decomp-last by fastforce*

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

**have**  $\Delta$ :  $\Delta$  *ww ww1 ww2 ss1 s2 statA ssv1 ssv2 statOO*  
**using**  $\chi^3$  **unfolding**  $\chi^3$ -*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**  
 $\chi^3a$ :  $\chi^3a \Delta \infty$  *ww1 ww2 w1' w2' ss1 s1' s2 statA ssv1 trv1' ssv2 trv2' statO'*  
**using** *unwindCond-ex- $\chi^3a$ -getSec[OF unwind  $\Delta$  rss1 r(2) rsv12 s13' niss1 iss1*  
*issv1 gissv13]*  
**by** *blast*

**have** *trv12'NE*: *trv1'  $\neq$  [] trv2'  $\neq$  [] using  $\chi^3a$  unfolding  $\chi^3a$ -def by auto*

**have** [*simp*]: *Van.O (butlast trv1 @ trv1') = Van.O trv1 @ Van.O trv1'*  
**using** *trv12'NE unfolding  $\chi^3$ -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  $\chi^3$ -def Van.O.map-filter Opt.O.map-filter apply (subst*  
*butlast-append) by simp*

**have** *Van.A trv1' = Van.A trv2'* **using**  $\chi^3a$  **unfolding**  $\chi^3a$ -*def by auto*  
**moreover** **have** *length (Van.O trv1') = length (Van.A trv1')  $\wedge$  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  $[simp]: Van.O\ trv1 @ Van.O\ trv1' = Van.O\ trv2 @ Van.O\ trv2' \longleftrightarrow$   
 $Van.O\ trv1 = Van.O\ trv2 \wedge Van.O\ trv1' = Van.O\ trv2'$  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  $\chi3b-def$  apply(intro conjI)  
 subgoal using  $\chi3\ \chi3a$  unfolding  $\chi3-def\ \chi3a-def$  by auto  
 subgoal using  $\chi3\ \chi3a$  unfolding  $\chi3-def\ \chi3a-def$  by auto  
 subgoal using  $\chi3\ \chi3a$  unfolding  $\chi3-def\ \chi3a-def$   
 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  $\chi3\ \chi3a$  unfolding  $\chi3-def\ \chi3a-def\ ssv1$   
 using Van.validFromS-append by auto  
 subgoal using  $\chi3\ \chi3a$  unfolding  $\chi3-def\ \chi3a-def\ ssv2$   
 using Van.validFromS-append by auto  
 subgoal using  $\chi3\ \chi3a$  unfolding  $\chi3-def\ \chi3a-def$  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  $\chi3\ \chi3a$  unfolding  $\chi3-def\ \chi3a-def$   
 by (simp add: butlast-append)  
 subgoal using  $\chi3\ \chi3a$  unfolding  $\chi3-def\ \chi3a-def$  Van.A.map-filter Opt.A.map-filter

apply(subst trv1l) apply(subst trv2l) by (simp add: butlast-append)  
 subgoal using  $\chi3a\ trv12'NE\ tr1NE$  unfolding  $\chi3a-def\ lastt-def$  by simp .

qed

lemma unwindCond-ex- $\chi3b-aux2$ :

assumes unwind: unwindCond  $\Delta$

and  $\Delta: \Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$

and r: reachO s1 reachO s2 reachV sv1 reachV sv2

and  $v3'$ : Opt.validFromS s1 (tr1 @ [s1',s1'])

and nis1:  $\neg isIntO\ s1$  and nis2:  $\neg isIntO\ s2$

and ninter3': never isIntO (tr1 @ [s1',s1'])

and sec: never isSecO tr1 isSecO s1'

shows  $\exists w1'\ w2'\ trv1\ trv2\ statOO.\ \chi3b\ \Delta\ \infty\ w1\ w2\ w1'\ w2'\ s1\ (tr1\ @\ [s1',s1'])$

$s2\ statA\ sv1\ trv1\ sv2\ trv2\ statOO$

proof -

have 0: lastt s1 (tr1 ## s1') = s1'

unfolding lastt-def by auto

show ?thesis

using unwindCond-ex- $\chi3b-aux2$ [OF unwind  $\Delta$  r, of tr1 ## s1', unfolded 0, sim-  
 plified]

using assms by auto

qed

**definition**  $\chi^3 \Delta w_1 w_2 w_1' w_2' s_1 tr_1 s_1' s_1'' s_2 statAA sv_1 trv_1 sv_1'' sv_2 trv_2 sv_2'' statOO \equiv$

$Van.validFromS sv_1 (trv_1 \#\# sv_1'') \wedge Van.validFromS sv_2 (trv_2 \#\# sv_2'') \wedge$   
 $Van.S (trv_1 \#\# sv_1'') = Opt.S ((tr_1 \#\# s_1') \#\# s_1'') \wedge never isSecV trv_2 \wedge$   
 $Van.A (trv_1 \#\# sv_1'') = Van.A (trv_2 \#\# sv_2'') \wedge$   
 $trv_1 \neq [] \wedge (trv_2 \neq [] \vee w_2' < w_2) \wedge$   
 $\Delta \infty w_1' w_2' s_1'' s_2 statAA sv_1'' sv_2'' statOO$

**proposition** *unwindCond-ex- $\chi^3$* :

**assumes** *unwind*: *unwindCond*  $\Delta$

**and**  $\Delta$ :  $\Delta w w_1 w_2 s_1 s_2 statA sv_1 sv_2 statO$  **and**

*r*: *reachO* *s1* *reachO* *s2* *reachV* *sv1* *reachV* *sv2*

**and** *v3'*: *Opt.validFromS* *s1* ((*tr1*  $\#\#$  *s1'*)  $\#\#$  *s1''*)

**and** *nis1*:  $\neg isIntO$  *s1* **and** *nis2*:  $\neg isIntO$  *s2*

**and** *ninter3'*: *never isIntO* ((*tr1*  $\#\#$  *s1'*)  $\#\#$  *s1''*)

**and** *sec*: *never isSecO* *tr1* *isSecO* *s1'*

**shows**  $\exists w_1' w_2' trv_1 sv_1'' trv_2 sv_2'' statOO. \chi^3 \Delta w_1 w_2 w_1' w_2' s_1 tr_1 s_1' s_1'' s_2 statA sv_1 trv_1 sv_1'' sv_2 trv_2 sv_2'' statOO$

**using** *unwindCond-ex- $\chi^3b$ -aux2*[*unfolded  $\varphi$ -def*, *unfolded lastt-snoc lastt-snoc2 append-snoc2*, *OF assms*]

**unfolding**  *$\chi^3b$ -def* **apply**(*elim exE*) **subgoal for**  $w_1' w_2' trv_1 trv_2 statOO$

**apply**(*cases trv1 rule: rev-cases*)

**subgoal by** *auto*

**subgoal for**  $trv_1' sv_1''$  **apply**(*cases trv2 rule: rev-cases*)

**subgoal by** *auto*

**subgoal for**  $trv_2' sv_2''$  **unfolding**  *$\chi^3$ -def*

**apply**(*rule exI*[*of - w1*  $\uparrow$ ]) **apply**(*rule exI*[*of - w2*  $\uparrow$ ])

**apply**(*rule exI*[*of - trv1*  $\uparrow$ ]) **apply**(*rule exI*[*of - sv1''*  $\uparrow$ ])

**apply**(*rule exI*[*of - trv2*  $\uparrow$ ]) **apply**(*rule exI*[*of - sv2''*  $\uparrow$ ])

**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**  $\omega^3 \Delta w_1 w_2 w_1' w_2' s_1 s_1' s_2 statAA sv_1 trv_1 sv_1' sv_2 trv_2 sv_2' statOO \equiv$

$Van.validFromS sv_1 (trv_1 \#\# sv_1') \wedge Van.validFromS sv_2 (trv_2 \#\# sv_2') \wedge$   
 $never isSecV trv_1 \wedge never isSecV trv_2 \wedge$   
 $Van.A (trv_1 \#\# sv_1') = Van.A (trv_2 \#\# sv_2') \wedge$   
 $(trv_1 \neq [] \vee w_1' < w_1) \wedge (trv_2 \neq [] \vee w_2' < w_2) \wedge$   
 $\Delta \infty w_1' w_2' s_1' s_2 statAA sv_1' sv_2' statOO$

**proposition** *unwindCond-ex- $\omega^3$* :

**assumes** *unwind*: *unwindCond*  $\Delta$   
**and**  $\Delta$ :  $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
**and** *r34*: *reachO s1 reachO s2* **and** *r12*: *reachV sv1 reachV sv2*  
**and** *v3*: *validTransO (s1,s1')*  
**and** *nis1*:  $\neg isIntO s1 \neg isIntO s1' \neg isSecO s1$   
**and** *nis2*:  $\neg isIntO s2$   
**shows**  $\exists w1' w2' trv1 sv1' trv2 sv2' statOO. \omega3 \Delta w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO$   
**using**  $\Delta r12$   
**proof**(*induction w arbitrary: w1 w2 sv1 sv2 statO rule: less-induct*)  
**case** (*less w w1 w2 sv1 sv2 statO*)  
**note**  $\Delta = \langle \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \rangle$   
**note** *r12* = *less.prem*s(2,3)  
**note** *r1* = *r12*(1) **note** *r2* = *r12*(2)  
**note** *r* = *r34 r12*  
  
**have** *f34*: *finalO s1 = finalO s2*  $\wedge$  *finalV sv1 = finalO s1*  $\wedge$  *finalV sv2 = finalO s2*  
**using**  $\Delta unwind[unfolded unwindCond-def]$  *r* **by** *auto*  
  
**have** *proact-match*: ( $\exists v < w. proact \Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ )  $\vee$  *react*  
 $\Delta w1 w2 s1 s2 statA sv1 sv2 statO$   
**using**  $\Delta unwind[unfolded unwindCond-def]$  *r* **by** *auto*  
**show** ?*case* **using** *proact-match* **proof** *safe*  
**fix** *v* **assume** *v*: *v < w*  
**assume** *proact*  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$   
**thus** ?*thesis* **unfolding** *proact-def* **proof** *safe*  
**assume** *sv1*:  $\neg isSecV sv1 \neg isIntV sv1$  **and** *move-1*  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$   
**then obtain** *sv1'* **where** *0*: *validTransV (sv1, sv1')* **and**  $\Delta$ :  $\Delta v w1 w2 s1 s2 statA sv1' sv2 statO$   
**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**  
 $\omega3$ :  $\omega3 \Delta w1 w2 w1' w2' s1 s1' s2 statA sv1' trv1 sv1'' sv2 trv2 sv2' statOO$   
  
**using** *less*(1)[*OF v*  $\Delta r1' r2$ ] **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**  $\omega3 0 sv1$  **unfolding**  $\omega3-def$  **by** *auto*  
**next**  
**assume** *sv2*:  $\neg isSecV sv2 \neg isIntV sv2$  **and** *move-2*  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$   
**then obtain** *sv2'*  
**where** *0*: *validTransV (sv2,sv2')*  
**and**  $\Delta$ :  $\Delta v w1 w2 s1 s2 statA sv1 sv2' statO$   
**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
 $\omega3: \omega3 \Delta w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2' trv2 sv2'' statOO$ 

using  $less(1)[OF v \Delta r1 r2]$  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  $\omega3 0 sv2$  unfolding  $\omega3\text{-def}$  by auto
next
assume  $sv1: \neg isSecV sv1$  and  $sv2: \neg isSecV sv2$  and
 $move\text{-}12 \Delta v w1 w2 s1 s2 statA sv1 sv2 statO$  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  $\Delta: \Delta v w1 w2 s1 s2 statA sv1' sv2' statO'$ 
unfolding  $move\text{-}12\text{-def}$  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
 $\omega3: \omega3 \Delta w1 w2 w1' w2' s1 s1' s2 statA sv1' trv1 sv1'' sv2' trv2 sv2'' statOO$ 

using  $less(1)[OF v \Delta r1' r2']$  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  $\omega3 0 sv1 sv2 sv12$  unfolding  $\omega3\text{-def}$   $statO'$  by (auto simp: Van.eqAct-def)
qed
next
assume  $m: react \Delta w1 w2 s1 s2 statA sv1 sv2 statO$ 
have  $m: match1 \Delta w1 w2 s1 s2 statA sv1 sv2 statO$  using  $m$  unfolding
 $react\text{-def}$  by auto
have ( $\exists w1' w2'. w1' < w1 \wedge w2' < w2 \wedge \neg isSecO s1 \wedge \Delta \infty w1' w2' s1' s2$ 
 $statA sv1 sv2 statO$ )  $\vee$ 
( $\exists w2' < w2. eqSec sv1 s1 \wedge \neg isIntV sv1 \wedge match1\text{-}1 \Delta \infty w2' s1 s1' s2$ 
 $statA sv1 sv2 statO$ )  $\vee$ 
( $eqSec sv1 s1 \wedge \neg isSecV sv2 \wedge Van.eqAct sv1 sv2 \wedge match1\text{-}12 \Delta \infty \infty$ 
 $s1 s1' s2 statA sv1 sv2 statO$ )
using  $m v3 nis1$  unfolding  $match1\text{-def}$  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  $\omega3\text{-def}$ 
by auto
subgoal for  $w2'$ 

```

```

apply(rule exI[of -  $\infty$ ]) 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  $\omega3$ -def using nis1(3) by (auto simp: eqSec-def) .
subgoal
apply(rule exI[of -  $\infty$ ]) apply(rule exI[of -  $\infty$ ])
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  $\omega3$ -def using nis1(3) apply (auto simp: eqSec-def
sstatA'-def sstatO'-def lastt-def Van.eqAct-def) . . .

```

qed  
qed

**definition**  $\chi_4 \Delta w w1 (w2::enat) w1' w2' s1 s2 tr2 statAA sv1 trv1 sv2 trv2$   
 $statOO \equiv$   
 $trv1 \neq [] \wedge trv2 \neq [] \wedge (length\ trv1 > Suc\ 0 \vee w1' \leq w1) \wedge$   
 $Van.validFromS\ sv1\ trv1 \wedge Van.validFromS\ sv2\ trv2 \wedge$   
 $never\ isSecV\ (butlast\ trv1) \wedge$   
 $never\ isSecV\ (butlast\ trv2) \wedge$   
 $isSecV\ (lastt\ sv2\ trv2) \wedge getSecV\ (lastt\ sv2\ trv2) = getSecO\ (lastt\ s2\ tr2) \wedge$   
 $Van.A\ trv1 = Van.A\ trv2 \wedge$   
 $\Delta\ w\ w1'\ w2'\ s1\ (lastt\ s2\ tr2)\ statAA\ (lastt\ sv1\ trv1)\ (lastt\ sv2\ trv2)\ statOO$

**lemma**  $\chi_4$ -final:

```

assumes unw: unwindCond  $\Delta$ 
and r: reachO s1 reachO s2 reachV sv1 reachV sv2
and vtr2: Opt.validFromS s2 tr2
and  $\chi_4$ :  $\chi_4 \Delta w w1 w2 w1' w2' s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO$ 
shows (finalV (lastt sv1 trv1)  $\longleftrightarrow$  finalO s1)  $\wedge$  (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 rs2: Opt.validFromS s2 tr2  $\longrightarrow$  reachO (lastt s2 tr2)
using r
by (simp add: Opt.reach-validFromS-reach lastt-def)+
show ?thesis using  $\chi_4$ [unfolded  $\chi_4$ -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**  $\chi_4$ -completedFrom: unwindCond  $\Delta \implies$   
 $reachO\ s1 \implies reachO\ s2 \implies reachV\ sv1 \implies reachV\ sv2 \implies$   
 $Opt.validFromS\ s2\ tr2 \implies completedFromO\ s2\ tr2 \implies$   
 $\chi_4\ \Delta\ w\ w1\ w2\ w1'\ w2'\ s1\ s2\ tr2\ statAA\ sv1\ trv1\ sv2\ trv2\ statOO$   
 $\implies completedFromV\ sv1\ trv1 \wedge completedFromV\ sv2\ trv2$   
**by** (metis Van.final-not-isSec  $\chi_4$ -def  $\chi_4$ -final completedFromO-lastt)

**proposition** unwindCond-ex- $\chi_4$ :  
**assumes** unwind: unwindCond  $\Delta$   
**and**  $\Delta$ :  $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**and**  $r$ :  $reachO\ s1\ reachO\ s2\ reachV\ sv1\ reachV\ sv2$   
**and**  $vtr2$ :  $Opt.validFromS\ s2\ tr2$   
**and**  $nis2$ :  $\neg isIntO\ s1$  **and**  $nis2$ :  $\neg isIntO\ s2$   
**and**  $inter_4$ : never isIntO  $tr2$   
**and**  $sec$ : never isSecO (butlast  $tr2$ ) isSecO (lastt  $s2\ tr2$ )  
**shows**  $\exists w'\ w1'\ w2'\ trv1\ trv2\ statOO. \chi_4\ \Delta\ w'\ w1\ w2\ w1'\ w2'\ s1\ s2\ tr2\ statA\ sv1$   
 $trv1\ sv2\ trv2\ statOO$   
**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**  $\Delta = \langle \Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \rangle$   
**note**  $nis1 = less(9)$  **note**  $nis2 = less(10)$   
**note**  $inter_4 = less(11)$   
**note**  $sec_4 = less(12,13)$   
**note**  $r_{34} = less.prem(2,3)$  **note**  $r_{12} = less.prem(4,5)$   
**note**  $r = r_{34}\ r_{12}$   
**note**  $r_3 = r_{34}(1)$  **note**  $r_4 = r_{34}(2)$  **note**  $r_1 = r_{12}(1)$  **note**  $r_2 = r_{12}(2)$   
  
**have**  $i_{34}$ :  $statA = Eq \longrightarrow isIntO\ s1 = isIntO\ s2$   
**and**  $f_{34}$ :  $finalO\ s1 = finalO\ s2 \wedge finalV\ sv1 = finalO\ s1 \wedge finalV\ sv2 = finalO$   
 $s2$   
**using**  $\Delta$  unwind[unfolded unwindCond-def]  $r$  **by** auto  
  
**have**  $proact$ -match:  $(\exists v < w. proact\ \Delta\ v\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO) \vee react$   
 $\Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**using**  $\Delta$  unwind[unfolded unwindCond-def]  $r$  **by** auto  
**show** ?case **using**  $proact$ -match **proof** safe  
**fix**  $v$  **assume**  $v: v < w$   
**assume**  $proact\ \Delta\ v\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**thus** ?thesis **unfolding**  $proact$ -def **proof** safe  
**assume**  $sv1$ :  $\neg isSecV\ sv1 \neg isIntV\ sv1$  **and**  $move-1\ \Delta\ v\ w1\ w2\ s1\ s2\ statA$   
 $sv1\ sv2\ statO$   
**then obtain**  $sv1'$   
**where**  $0$ : validTransV ( $sv1, sv1'$ )  
**and**  $\Delta$ :  $\Delta\ v\ w1\ w2\ s1\ s2\ statA\ sv1'\ sv2\ statO$   
**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  $\chi_4: \chi_4 \Delta w' w1 w2 w1' w2' s1$ 
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  $\Delta r3_4 r1' r2 vtr2 nis1 nis2 inter_4 sec_4$ ] 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  $\chi_4$  0 sv1 unfolding  $\chi_4$ -def by auto
next
assume sv2:  $\neg isSecV sv2 \neg isIntV sv2$  and move-2  $\Delta v w1 w2 s1 s2 statA$ 
sv1 sv2 statO
then obtain sv2'
where 0: validTransV (sv2,sv2')
and  $\Delta: \Delta v w1 w2 s1 s2 statA sv1 sv2' statO$ 
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  $\chi_4: \chi_4 \Delta w' w1 w2 w1' w2' s1$ 
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  $\Delta r3_4 r1 r2' vtr2 nis1 nis2 inter_4 sec_4$ ] 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  $\chi_4$  0 sv2 unfolding  $\chi_4$ -def by auto
next
assume sv12:  $\neg isSecV sv1 \neg isSecV sv2 Van.eqAct sv1 sv2$ 
and move-12  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ 
then obtain sv1' sv2' statO'
where 0: statO' = sstatO' statO sv1 sv2
validTransV (sv1,sv1')  $\neg isSecV sv1$ 
validTransV (sv2,sv2')  $\neg isSecV sv2$ 
Van.eqAct sv1 sv2
and  $\Delta: \Delta v w1 w2 s1 s2 statA sv1' sv2' statO'$ 
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  $\chi_4: \chi_4 \Delta w' w1 w2 w1' w2' s1$ 
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  $\Delta r3_4 r12' vtr2 nis1 nis2 inter_4 sec_4$ ] 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  $\chi_4$  0 sv12 unfolding  $\chi_4$ -def sstatO'-def
by (auto simp: Van.eqAct-def)
qed
next
assume m: react  $\Delta w1 w2 s1 s2 statA sv1 sv2 statO$ 
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 react-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
react-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) ∨
      (eqSec sv2 s2 ∧ ¬ isSecV sv1 ∧ Van.eqAct sv1 sv2 ∧ 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 ¬ isSecO s2 and Δ: Δ ∞ w1'' w2'' s1 s2' statA sv1 sv2 statO

```

hence  $S_4$ :  $Opt.S\ tr2' = Opt.S\ tr2$  **unfolding**  $tr2$  **by** *auto*  
**obtain**  $w' w1' w2' trv1\ trv2\ statOO$  **where**  $\chi_4$ :  $\chi_4\ \Delta\ w'\ w1''\ w2''\ w1'\ w2'$   
 $s1\ s2'\ tr2'\ statA\ sv1\ trv1\ sv2\ trv2\ statOO$   
**using**  $less(1)[of\ tr2',\ OF - \Delta\ r3\ r4'\ r12]$  **unfolding**  $tr2$   
**by** *simp* (*metis*  $Opt.S.eq-Nil-iff(2)$   $S_4$  *Simple-Transition-System.validFromS-def*  
*last.simps lastt-def*  
 $list.discI\ list-all-hd\ nev_4'\ nis1\ sec_4(1)\ sec_4(2)\ tr2\ tr2'\ tr2'NE$ )  
**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**  $Van.validFromS-Cons\ tr2'NE\ tr2'\ trn_4\ isAO_4\ w12'$  **by** *auto*  
**next**  
**fix**  $w1''$  **assume**  $w1'$ :  $w1'' < w1$   
**assume**  $trn2_4$ :  $eqSec\ sv2\ s2$  **and**  
 $Atrn2$ :  $\neg\ isIntV\ sv2$  **and**  $match2-1\ \Delta\ w1''\ \infty\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO$   
**then obtain**  $sv2'$  **where**  
 $trn2$ :  $validTransV\ (sv2,sv2')$  **and**  
 $\Delta$ :  $\Delta\ \infty\ w1''\ \infty\ s1\ s2'\ statA\ sv1\ sv2'\ statO$   
**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**  $\chi_4$ :  $\chi_4\ \Delta\ w'\ w1''\ \infty\ w1'\ w2'$   
 $s1\ s2'\ tr2'\ statA\ sv1\ trv1\ sv2'\ trv2\ statOO$   
**using**  $less(1)[of\ tr2',\ OF - \Delta\ r3\ r4'\ r1\ r2',\ unfolded\ O44',\ simplified]$   
**using**  $less.premis\ tr2'\ f3\ f4\ tr2'NE\ trn_4\ O44'(1)$   
**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**  $Van.validFromS-Cons\ trn2\ tr2'NE\ tr2'\ trn_4$   
**using**  $isAO_4\ Atrn2\ eqSec-S-Cons\ trn2_4\ w1'$   
**by** *simp* (*metis*  $Opt.S.Nil-iff\ Opt.S.eq-Nil-iff(1)\ eqSec-def\ nless-le\ order-le-less-trans$   
 $s2_4\ sec_4(1)\ tr2$ )  
**next**  
**assume**  $sv1$ :  $\neg\ isSecV\ sv1$  **and**  $trn2_4$ :  $eqSec\ sv2\ s2$  **and**  
 $Atrn12$ :  $Van.eqAct\ sv1\ sv2$  **and**  $match2-12\ \Delta\ \infty\ \infty\ s1\ s2\ s2'\ statA\ sv1\ sv2$   
 $statO$   
**then obtain**  $sv1'\ sv2'\ statO'$  **where**  
 $statO'$ :  $statO' = sstatO'\ statO\ sv1\ sv2$  **and**  
 $trn1$ :  $validTransV\ (sv1,sv1')$  **and**  
 $trn2$ :  $validTransV\ (sv2,sv2')$  **and**  
 $\Delta$ :  $\Delta\ \infty\ \infty\ \infty\ s1\ s2'\ statA\ sv1'\ sv2'\ statO'$   
**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**  $\chi_4$ :  $\chi_4\ \Delta\ w'\ \infty\ \infty\ w1'\ w2'\ s1$   
 $s2'\ tr2'\ statA\ sv1'\ trv1\ sv2'\ trv2\ statOO$   
**using**  $less(1)[of\ tr2',\ OF - \Delta\ r3\ r4'\ r12',\ unfolded\ O44',\ simplified]$

```

using less.premis 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  $\chi_4$  O44' tr2'NE sv1
using Van.validFromS-Cons trn1 trn2
using isAO4 Atrn12 eqSec-S-Cons trn24 f3 f34 s24 tr2' trn4
unfolding  $\chi_4$ -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**  $\chi_4a$  **where**  $\chi_4a \Delta w w1 (w2::enat) w1' w2' s1 s2 s2' statAA sv1 trv1$   
 $sv2 trv2 statOO \equiv$   
 $trv1 \neq [] \wedge trv2 \neq [] \wedge (length\ trv1 > Suc\ 0 \vee w1' < w1) \wedge$   
 $Van.validFromS\ sv1\ trv1 \wedge Van.validFromS\ sv2\ trv2 \wedge$   
 $never\ isSecV\ (butlast\ trv1) \wedge$   
 $Van.S\ trv2 = [getSecO\ s2] \wedge$   
 $Van.A\ trv1 = Van.A\ trv2 \wedge$   
 $\Delta\ w\ w1'\ w2'\ s1\ s2'\ statAA\ (lastt\ sv1\ trv1)\ (lastt\ sv2\ trv2)\ statOO$

**lemma** unwindCond-ex- $\chi_4a$ -getSec:

**assumes** unwind: unwindCond  $\Delta$

**and**  $\Delta$ :  $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$

**and** r34: reachO s1 reachO s2 **and** r12: reachV sv1 reachV sv2

**and** v: validTransO (s2, s2')

**and** ii4:  $\neg isIntO\ s2$

**and** is2: isSecO s2 **and** isv24: isSecV sv2 getSecO s2 = getSecV sv2

**shows**  $\exists w1' w2' trv1 trv2 statOO.$

$\chi_4a \Delta \infty w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv2 trv2 statOO$

**using**  $\Delta$  r12 isv24

**proof**(induction w arbitrary: w1 w2 sv1 sv2 statO rule: less-induct)

**case** (less w w1 w2 sv1 sv2 statO)

**note**  $\Delta = \langle \Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \rangle$

**note** r12 = less.premis(2,3)

**note** r1 = r12(1) **note** r2 = r12(2)

**note** r = r34 r12

**note** isv24 =  $\langle isSecV\ sv2 \rangle \langle getSecO\ s2 = getSecV\ sv2 \rangle$

**have** f34: finalO s1 = finalO s2  $\wedge$  finalV sv1 = finalO s1  $\wedge$  finalV sv2 = finalO s2

**using**  $\Delta$  unwind[unfolded unwindCond-def] r **by** auto

**have** proact-match:  $(\exists v < w. proact\ \Delta\ v\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO) \vee react\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$

```

using  $\Delta$  unwind[unfolded unwindCond-def] r by auto
show ?case using proact-match proof safe
  fix v assume v: v < w
  assume proact  $\Delta$  v w1 w2 s1 s2 statA sv1 sv2 statO
  thus ?thesis unfolding proact-def proof safe
    assume sv1:  $\neg$  isSecV sv1  $\neg$  isIntV sv1 and move-1  $\Delta$  v w1 w2 s1 s2 statA
    sv1 sv2 statO
    then obtain sv1'
    where 0: validTransV (sv1,sv1')
    and  $\Delta$ :  $\Delta$  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
     $\chi4a$ :  $\chi4a$   $\Delta$   $\infty$  w1 w2 w1' w2' s1 s2 s2' statA sv1' trv1 sv2 trv2 statOO
    using less(1)[OF v  $\Delta$  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  $\chi4a$  0 sv1 unfolding  $\chi4a$ -def by auto
    next
    assume sv2:  $\neg$  isSecV sv2  $\neg$  isIntV sv2 and move-2  $\Delta$  v w1 w2 s1 s2 statA
    sv1 sv2 statO
    hence False using isv24 by blast
    thus ?thesis by auto
    next
    assume sv12:  $\neg$  isSecV sv1  $\neg$  isSecV sv2 Van.eqAct sv1 sv2
    and move-12  $\Delta$  v w1 w2 s1 s2 statA sv1 sv2 statO
    hence False using isv24 by blast
    thus ?thesis by auto
  qed
next
  assume m: react  $\Delta$  w1 w2 s1 s2 statA sv1 sv2 statO
  have m: match2  $\Delta$  w1 w2 s1 s2 statA sv1 sv2 statO using m unfolding
  react-def by auto
  have ( $\exists$  w1' w2'. w1' < w1  $\wedge$  w2' < w2  $\wedge$   $\neg$  isSecO s2  $\wedge$   $\Delta$   $\infty$  w1' w2' s1 s2'
  statA sv1 sv2 statO)  $\vee$ 
  ( $\exists$  w1' < w1. eqSec sv2 s2  $\wedge$   $\neg$  isIntV sv2  $\wedge$  match2-1  $\Delta$  w1'  $\infty$  s1 s2 s2'
  statA sv1 sv2 statO)  $\vee$ 
  (eqSec sv2 s2  $\wedge$   $\neg$  isSecV sv1  $\wedge$  Van.eqAct sv1 sv2  $\wedge$  match2-12  $\Delta$   $\infty$   $\infty$ 
  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 -  $\infty$ ])
  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  $\chi4a$ -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**  $tr2l$ :  $tr2 = \text{butlast } tr2 \text{ @ } [ss2]$  **by** (*simp add: ss2l tr2NE*)  
**have**  $ssv1l$ :  $ssv1 = \text{last } trv1$  **using**  $\chi_4$  **unfolding**  $\chi_4\text{-def}$  **by** (*metis lastt-def ssv1*)  
**have**  $trv1l$ :  $trv1 = \text{butlast } trv1 \text{ @ } [ssv1]$  **by** (*simp add: ssv1l trv12NE(1)*)  
**have**  $ssv2l$ :  $ssv2 = \text{last } trv2$  **using**  $\chi_4$  **unfolding**  $\chi_4\text{-def}$  **by** (*metis lastt-def ssv2*)  
**have**  $trv2l$ :  $trv2 = \text{butlast } trv2 \text{ @ } [ssv2]$  **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**  $\chi_4$  **unfolding**  $\chi_4\text{-def}$   $ssv2 \ ss2$  **by** *auto*

**have**  $niss2$ :  $\neg isIntO \ ss2$   
**using**  $ninter4 \ tr2l \ ss2l \ tr2NE$  **by** (*simp add: list-all-iff*)

**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**  $\chi_4$  **unfolding**  $\chi_4\text{-def}$   
**using**  $Van.reach-validFromS-reach \ ssv1 \ ssv2 \ ssv1l \ ssv2l$  **by** *auto metis+*

**have**  $\Delta$ :  $\Delta \ ww \ ww1 \ ww2 \ s1 \ ss2 \ statA \ ssv1 \ ssv2 \ statOO$   
**using**  $\chi_4$  **unfolding**  $\chi_4\text{-def}$   $ss2[symmetric] \ ssv1[symmetric] \ ssv2[symmetric]$  **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**  
 $\chi_4a$ :  $\chi_4a \ \Delta \ \infty \ ww1 \ ww2 \ w1' \ w2' \ s1 \ ss2 \ s2' \ statA \ ssv1 \ trv1' \ ssv2 \ trv2' \ statO'$   
**using**  $unwindCond-ex-\chi_4a-getSec[OF \ unwind \ \Delta \ r(1) \ rss2 \ rssv12 \ s13' \ niss2 \ iss2 \ issv2 \ gissv24]$   
**by** *blast*

**have**  $trv12'NE$ :  $trv1' \neq [] \ trv2' \neq []$  **using**  $\chi_4a$  **unfolding**  $\chi_4a\text{-def}$  **by** *auto*

**have**  $[simp]$ :  $Van.O \ (\text{butlast } trv1 \text{ @ } trv1') = Van.O \ trv1 \text{ @ } Van.O \ trv1'$   
**using**  $trv12'NE$  **unfolding**  $\chi_4\text{-def}$   $Van.O.map-filter \ Opt.O.map-filter$  **apply** (*subst butlast-append*) **by** *simp*

**have**  $[simp]$ :  $Van.O \ (\text{butlast } trv2 \text{ @ } trv2') = Van.O \ trv2 \text{ @ } Van.O \ trv2'$   
**using**  $trv12'NE$  **unfolding**  $\chi_4\text{-def}$   $Van.O.map-filter \ Opt.O.map-filter$  **apply** (*subst butlast-append*) **by** *simp*

**have**  $Van.A \ trv1' = Van.A \ trv2'$  **using**  $\chi_4a$  **unfolding**  $\chi_4a\text{-def}$  **by** *auto*  
**moreover** **have**  $length \ (Van.O \ trv1') = length \ (Van.A \ trv1') \wedge length \ (Van.O \ trv2') = length \ (Van.A \ trv2')$   
**unfolding**  $Van.A.map-filter \ Van.O.map-filter$  **by** *auto*  
**ultimately** **have**  $length \ (Van.O \ trv1') = length \ (Van.O \ trv2')$  **by** *auto*

hence  $[simp]: Van.O\ trv1\ @\ Van.O\ trv1'\ =\ Van.O\ trv2\ @\ Van.O\ trv2'\ \longleftrightarrow$   
 $Van.O\ trv1\ =\ Van.O\ trv2\ \wedge\ Van.O\ trv1'\ =\ Van.O\ trv2'$  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  $\chi_4b\text{-def}$  apply(intro conjI)  
 subgoal using  $\chi_4\ \chi_4a$  unfolding  $\chi_4\text{-def}\ \chi_4a\text{-def}$  by auto  
 subgoal using  $\chi_4\ \chi_4a$  unfolding  $\chi_4\text{-def}\ \chi_4a\text{-def}$  by auto  
 subgoal using  $\chi_4\ \chi_4a$  unfolding  $\chi_4\text{-def}\ \chi_4a\text{-def}$   
 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  $\chi_4\ \chi_4a$  unfolding  $\chi_4\text{-def}\ \chi_4a\text{-def}\ ssv1$   
 using *Van.validFromS-append* by auto  
 subgoal using  $\chi_4\ \chi_4a$  unfolding  $\chi_4\text{-def}\ \chi_4a\text{-def}\ ssv2$   
 using *Van.validFromS-append* by auto  
 subgoal using  $\chi_4\ \chi_4a$  unfolding  $\chi_4\text{-def}\ \chi_4a\text{-def}$   
 by (simp add: butlast-append)  
 subgoal using  $\chi_4\ \chi_4a$  unfolding  $\chi_4\text{-def}\ \chi_4a\text{-def}$  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  $\chi_4\ \chi_4a$  unfolding  $\chi_4\text{-def}\ \chi_4a\text{-def}\ Van.A.map-filter\ Opt.A.map-filter$   
 apply(subst trv1l) apply(subst trv2l)  
 apply(subst butlast-append) apply simp apply(subst butlast-append) by simp  
 subgoal using  $\chi_4a\ trv12'\ NE\ tr2NE$  unfolding  $\chi_4a\text{-def}\ lastt\text{-def}$  by simp .

qed

lemma *unwindCond-ex- $\chi_4b$ -aux2*:

assumes *unwind*: *unwindCond*  $\Delta$

and  $\Delta$ :  $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$  and

*r*: *reachO* *s1* *reachO* *s2* *reachV* *sv1* *reachV* *sv2*

and  $v_4'$ : *Opt.validFromS* *s2* (*tr2* @ [*s2'*,*s2'*'])

and *nis1*:  $\neg$  *isIntO* *s1* and *nis2*:  $\neg$  *isIntO* *s2*

and *ninter4'*: *never isIntO* (*tr2* @ [*s2'*,*s2'*'])

and *sec*: *never isSecO* *tr2* *isSecO* *s2'*

shows  $\exists w1'\ w2'\ trv1\ trv2\ statOO.\ \chi_4b\ \Delta\ \infty\ w1\ w2\ w1'\ w2'\ s1\ s2\ (tr2\ @\ [s2',s2'])$

*statA* *sv1* *trv1* *sv2* *trv2* *statOO*

proof –

have 0: *lastt* *s2* (*tr2* ## *s2'*) = *s2'*

unfolding *lastt-def* by auto

show *?thesis*

using *unwindCond-ex- $\chi_4b$ -aux*[OF *unwind*  $\Delta$  *r*, of *tr2* ## *s2'*, unfolded 0, *sim-*

plified]  
 using *assms* by *auto*  
 qed

**definition**  $\chi_4' \Delta w_1 w_2 w_1' (w_2'::\text{enat}) s_1 s_2 tr_2 s_2' s_2'' \text{statAA } sv_1 trv_1 sv_1'' sv_2 trv_2 sv_2'' \text{statOO} \equiv$   
 $Van.\text{validFromS } sv_1 (trv_1 \#\# sv_1'') \wedge Van.\text{validFromS } sv_2 (trv_2 \#\# sv_2'') \wedge$   
 $never \text{isSecV } (\text{butlast } (trv_1 \#\# sv_1'')) \wedge$   
 $Van.S (trv_2 \#\# sv_2'') = Opt.S ((tr_2 \#\# s_2') \#\# s_2'') \wedge$   
 $Van.A (trv_1 \#\# sv_1'') = Van.A (trv_2 \#\# sv_2'') \wedge$   
 $trv_2 \neq [] \wedge (trv_1 \neq [] \vee w_1' < w_1) \wedge$   
 $\Delta \infty w_1' w_2' s_1 s_2'' \text{statAA } sv_1'' sv_2'' \text{statOO}$

**proposition** *unwindCond-ex- $\chi_4'$* :  
**assumes** *unwind*: *unwindCond*  $\Delta$   
**and**  $\Delta$ :  $\Delta w_1 w_2 s_1 s_2 \text{statA } sv_1 sv_2 \text{statO}$  **and**  
*r*: *reachO*  $s_1$  *reachO*  $s_2$  *reachV*  $sv_1$  *reachV*  $sv_2$   
**and**  $v_4'$ : *Opt.validFromS*  $s_2 ((tr_2 \#\# s_2') \#\# s_2'')$   
**and** *nis1*:  $\neg \text{isIntO } s_1$  **and** *nis2*:  $\neg \text{isIntO } s_2$   
**and** *ninter4'*: *never isIntO*  $((tr_2 \#\# s_2') \#\# s_2'')$   
**and** *sec*: *never isSecO*  $tr_2$  *isSecO*  $s_2'$   
**shows**  $\exists w_1' w_2' trv_1 sv_1'' trv_2 sv_2'' \text{statOO}. \chi_4' \Delta w_1 w_2 w_1' w_2' s_1 s_2 tr_2 s_2' s_2'' \text{statA } sv_1 trv_1 sv_1'' sv_2 trv_2 sv_2'' \text{statOO}$   
**using** *unwindCond-ex- $\chi_4'b$ -aux2*[*unfolded  $\varphi$ -def*, *unfolded lastt-snoc lastt-snoc2 append-snoc2*, *OF assms*]  
**unfolding**  *$\chi_4'b$ -def* **apply**(*elim exE*) **subgoal for**  $w_1' w_2' trv_1 trv_2 \text{statOO}$   
**apply**(*cases trv1 rule: rev-cases*)  
**subgoal by auto**  
**subgoal for**  $trv_1' sv_1''$  **apply**(*cases trv2 rule: rev-cases*)  
**subgoal by auto**  
**subgoal for**  $trv_2' sv_2''$  **unfolding**  *$\chi_4'$ -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**  $\omega_4 \Delta w_1 w_2 w_1' (w_2'::\text{enat}) s_1 s_2 s_2' \text{statAA } sv_1 trv_1 sv_1' sv_2 trv_2 sv_2' \text{statOO} \equiv$   
 $Van.\text{validFromS } sv_1 (trv_1 \#\# sv_1') \wedge Van.\text{validFromS } sv_2 (trv_2 \#\# sv_2') \wedge$   
 $never \text{isSecV } trv_1 \wedge never \text{isSecV } trv_2 \wedge$   
 $Van.A (trv_1 \#\# sv_1') = Van.A (trv_2 \#\# sv_2') \wedge$

$$(trv1 \neq [] \vee w1' < w1) \wedge (trv2 \neq [] \vee w2' < w2) \wedge \\ \Delta \infty w1' w2' s1 s2' statAA sv1' sv2' statOO$$

**proposition** *unwindCond-ex- $\omega_4$* :  
**assumes** *unwind*: *unwindCond*  $\Delta$   
**and**  $\Delta$ :  $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
**and** *r34*: *reachO* *s1* *reachO* *s2* **and** *r12*: *reachV* *sv1* *reachV* *sv2*  
**and** *nis1*:  $\neg isIntO$  *s1*  
**and** *v4*: *validTransO* (*s2*, *s2'*)  
**and** *nis2*:  $\neg isIntO$  *s2*  $\neg isIntO$  *s2'*  $\neg isSecO$  *s2*  
**shows**  $\exists w1' w2' trv1 sv1' trv2 sv2' statOO. \omega_4 \Delta w1 w2 w1' w2' s1 s2 s2' statA$   
 $sv1 trv1 sv1' sv2 trv2 sv2' statOO$   
**using**  $\Delta r12$   
**proof**(*induction* *w* *arbitrary*: *w1 w2 sv1 sv2 statO* *rule*: *less-induct*)  
  **case** (*less* *w w1 w2 sv1 sv2 statO*)  
  **note**  $\Delta = \langle \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \rangle$   
  **note** *r12* = *less.prem*s(2,3)  
  **note** *r1* = *r12*(1) **note** *r2* = *r12*(2)  
  **note** *r* = *r34* *r12*  
  
  **have** *f34*: *finalO* *s1* = *finalO* *s2*  $\wedge$  *finalV* *sv1* = *finalO* *s1*  $\wedge$  *finalV* *sv2* = *finalO*  
  *s2*  
  **using**  $\Delta$  *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*  
  
  **have** *proact-match*: ( $\exists v < w. proact \Delta v w1 w2 s1 s2 statA sv1 sv2 statO$ )  $\vee$  *react*  
 $\Delta w1 w2 s1 s2 statA sv1 sv2 statO$   
  **using**  $\Delta$  *unwind*[*unfolded unwindCond-def*] *r* **by** *auto*  
  **show** ?*case* **using** *proact-match* **proof** *safe*  
  **fix** *v* **assume** *v*: *v* < *w*  
  **assume** *proact*  $\Delta v w1 w2 s1 s2 statA sv1 sv2 statO$   
  **thus** ?*thesis* **unfolding** *proact-def* **proof** *safe*  
  **assume** *sv1*:  $\neg isSecV$  *sv1*  $\neg isIntV$  *sv1* **and** *move-1*  $\Delta v w1 w2 s1 s2 statA$   
  *sv1 sv2 statO*  
  **then obtain** *sv1'* **where** *0*: *validTransV* (*sv1*, *sv1'*) **and**  $\Delta$ :  $\Delta v w1 w2 s1$   
  *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**  
   $\omega_4$ :  $\omega_4 \Delta w1 w2 w1' w2' s1 s2 s2' statA sv1' trv1 sv1'' sv2 trv2 sv2' statOO$   
  
  **using** *less*(1)[*OF* *v*  $\Delta r1' r2$ ] **by** *auto*  
  **show** ?*thesis* **apply**(*rule* *exI*[*of* - *w1*  $\uparrow$ ]) **apply**(*rule* *exI*[*of* - *w2*  $\uparrow$ ]) **apply**(*rule*  
  *exI*[*of* - *sv1*  $\#$  *trv1*]) **apply**(*rule* *exI*[*of* - *sv1''*  $\uparrow$ ])  
  **apply**(*rule* *exI*[*of* - *trv2*]) **apply**(*rule* *exI*[*of* - *sv2*  $\uparrow$ ])  
  **using**  $\omega_4$  *0* *sv1* **unfolding**  $\omega_4$ -*def* **by** *auto*  
  **next**  
  **assume** *sv2*:  $\neg isSecV$  *sv2*  $\neg isIntV$  *sv2* **and** *move-2*  $\Delta v w1 w2 s1 s2 statA$   
  *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: react Δ w1 w2 s1 s2 statA sv1 sv2 statO
have m: match2 Δ w1 w2 s1 s2 statA sv1 sv2 statO using m unfolding
react-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  $\omega_4$ -def
by auto
subgoal for  $w1'$  apply(rule exI[of -  $w1'$ ]) apply(rule exI[of -  $\infty$ ])
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  $\omega_4$ -def using nis2(3) by (auto simp: eqSec-def) .
subgoal apply(rule exI[of -  $\infty$ ]) apply(rule exI[of -  $\infty$ ])
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  $\omega_4$ -def using nis2(3) apply (auto simp: eqSec-def
sstatA'-def sstatO'-def lastt-def Van.eqAct-def) . . .
qed
qed

```

**definition**  $\varphi\varphi$   $s1$   $ltr1$   $s2$   $ltr2$   $tr1$   $s1'$   $s1''$   $ltr1'$   $tr2$   $s2'$   $s2''$   $ltr2'$   $\equiv$

```

   $ltr1 = lappend$  ( $l$ list-of ( $tr1$  ##  $s1'$ )) ( $s1''$  $  $ltr1'$ )  $\wedge$ 
   $ltr2 = lappend$  ( $l$ list-of ( $tr2$  ##  $s2'$ )) ( $s2''$  $  $ltr2'$ )  $\wedge$ 
   $Opt.validFromS$   $s1$  (( $tr1$  ##  $s1'$ ) ##  $s1''$ )  $\wedge$   $Opt.validFromS$   $s2$  (( $tr2$  ##  $s2'$ )
##  $s2''$ )  $\wedge$ 
   $never$   $isIntO$   $tr1$   $\wedge$   $never$   $isIntO$   $tr2$   $\wedge$ 
   $isIntO$   $s1'$   $\wedge$   $isIntO$   $s2'$   $\wedge$   $getActO$   $s1' = getActO$   $s2'$   $\wedge$ 
   $Opt.lvalidFromS$   $s1''$  ( $s1''$  $  $ltr1'$ )  $\wedge$   $Opt.lcompletedFrom$   $s1''$  ( $s1''$  $  $ltr1'$ )  $\wedge$ 
   $Opt.lvalidFromS$   $s2''$  ( $s2''$  $  $ltr2'$ )  $\wedge$   $Opt.lcompletedFrom$   $s2''$  ( $s2''$  $  $ltr2'$ )  $\wedge$ 
   $Opt.lA$  ( $s1''$  $  $ltr1'$ ) =  $Opt.lA$  ( $s2''$  $  $ltr2'$ )

```

**lemma**  $isIntO$ - $\varphi\varphi$ :

**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**  $inter3$ :  $\neg$   $lnever$   $isIntO$   $ltr1$

**shows**  $\exists$   $tr1$   $s1'$   $s1''$   $ltr1'$   $tr2$   $s2'$   $s2''$   $ltr2'$ .  $\varphi\varphi$   $s1$   $ltr1$   $s2$   $ltr2$   $tr1$   $s1'$   $s1''$   $ltr1'$   $tr2$   $s2'$   $s2''$   $ltr2'$

**proof** –

**have**  $03$ :  $\exists s \in lset$   $ltr1$ .  $isIntO$   $s$  **using**  $inter3$  **unfolding**  $l$ list.pred-set **by** auto

**define**  $ttr1$  **where**  $ttr1$ :  $ttr1 \equiv ltakeUntil$   $isIntO$   $ltr1$

**define**  $lltr1'$  **where**  $lltr1'$ :  $lltr1' \equiv ldropUntil$   $isIntO$   $ltr1$

**have**  $ltr1$ :  $ltr1 = lappend$  ( $l$ list-of  $ttr1$ )  $lltr1'$

**unfolding**  $ttr1$   $lltr1'$   $lappend$ - $ltakeUntil$ - $ldropUntil$ [OF  $03$ ] ..

**have**  $13$ :  $ttr1 \neq [] \wedge \text{never isIntO (butlast ttr1)} \wedge \text{isIntO (last ttr1)}$   
**unfolding**  $ttr1$   
**using**  $ltakeUntil\text{-last}[OF\ 03]$   $ltakeUntil\text{-not-Nil}[OF\ 03]$   $ltakeUntil\text{-never-butlast}[OF\ 03]$  **by**  $simp$   
**then obtain**  $tr1\ s1'$  **where**  $ttr1\text{-eq}$ :  $ttr1 = tr1 \#\# s1'$   
**using**  $rev\text{-exhaust}$  **by**  $blast$   
**hence**  $tr1s1'$ :  $\text{never isIntO } tr1 \text{ isIntO } s1'$  **using**  $13$  **by**  $auto$   
**have**  $lfinite\ ltr1 \implies s1' \neq \text{last } ltr1$   
**by** ( $metis\ Opt.\text{final-not-isInt}\ Opt.\text{lcompletedFrom-def}\ \text{llast-last-llist-of } tr1s1'(2)$ )  
 $vltr1(2)$   
**hence**  $ne$ :  $lltr1' \neq []$   
**using**  $ltr1$  **unfolding**  $ttr1\text{-eq}$  **by**  $auto$   
**then obtain**  $s1''\ ltr1'$  **where**  $lltr1'$ :  $lltr1' = s1'' \$ ltr1'$   
**by** ( $meson\ llist.\text{exhaust}$ )  
**have** [ $simp$ ]:  $\text{filter isIntO } tr1 = []$   
**by** ( $metis\ never\text{-Nil-filter } tr1s1'(1)$ )  
**have**  $clltr1'$ :  $Opt.\text{lcompletedFrom } s1\ lltr1'$   
**by** ( $metis\ Opt.\text{lcompletedFrom-def}\ lfinite\text{-lappend}\ lfinite\text{-llist-of}\ \text{llast-lappend-LCons}$   
 $\text{llast-last-llist-of } lltr1'\ ltr1\ ne\ vltr1(2)$ )

**have**  $inter4$ :  $\neg \text{never isIntO } ltr2$  **using**  $A\ inter3$   
**by** ( $metis\ Opt.\text{lA.eq-LNil-iff}\ Opt.\text{lO}\ Opt.\text{lO.eq-LNil-iff}\ lfiltermap\text{-LNil-never}$   
 $lfiltermap\text{-lmap-lfilter } vltr1(2)\ vltr2(2)$ )  
**have**  $04$ :  $\exists s \in \text{lset } ltr2. \text{isIntO } s$  **using**  $inter4$  **unfolding**  $l\text{list.pred-set}$  **by**  $auto$   
**define**  $ttr2$  **where**  $ttr2$ :  $ttr2 \equiv \text{ltakeUntil isIntO } ltr2$   
**define**  $lltr2'$  **where**  $lltr2'$ :  $lltr2' \equiv \text{ldropUntil isIntO } ltr2$   
**have**  $ltr2$ :  $ltr2 = \text{lappend (llist-of } ttr2) lltr2'$   
**unfolding**  $ttr2\ lltr2'$   $\text{lappend-ltakeUntil-ldropUntil}[OF\ 04]$  **..**  
**have**  $14$ :  $ttr2 \neq [] \wedge \text{never isIntO (butlast ttr2)} \wedge \text{isIntO (last ttr2)}$   
**unfolding**  $ttr2$   
**using**  $ltakeUntil\text{-last}[OF\ 04]$   $ltakeUntil\text{-not-Nil}[OF\ 04]$   $ltakeUntil\text{-never-butlast}[OF\ 04]$  **by**  $simp$   
**then obtain**  $tr2\ s2'$  **where**  $ttr2\text{-eq}$ :  $ttr2 = tr2 \#\# s2'$   
**using**  $rev\text{-exhaust}$  **by**  $blast$   
**hence**  $tr2s2'$ :  $\text{never isIntO } tr2 \text{ isIntO } s2'$  **using**  $14$  **by**  $auto$   
**have**  $lfinite\ ltr2 \implies s2' \neq \text{last } ltr2$   
**by** ( $metis\ Opt.\text{final-not-isInt}\ Opt.\text{lcompletedFrom-def}\ \text{llast-last-llist-of } tr2s2'(2)$ )  
 $vltr2(2)$   
**hence**  $ne$ :  $lltr2' \neq []$   
**using**  $ltr2$  **unfolding**  $ttr2\text{-eq}$  **by**  $auto$   
**then obtain**  $s2''\ ltr2'$  **where**  $lltr2'$ :  $lltr2' = s2'' \$ ltr2'$   
**by** ( $meson\ llist.\text{exhaust}$ )  
**have** [ $simp$ ]:  $\text{filter isIntO } tr2 = []$   
**by** ( $metis\ never\text{-Nil-filter } tr2s2'(1)$ )  
**have**  $clltr2'$ :  $Opt.\text{lcompletedFrom } s2\ lltr2'$   
**by** ( $metis\ Opt.\text{lcompletedFrom-def}\ lfinite\text{-lappend}\ lfinite\text{-llist-of}\ \text{llast-lappend-LCons}$   
 $\text{llast-last-llist-of } lltr2'\ ltr2\ ne\ vltr2(2)$ )

**have**  $AA: \text{Opt.lA } lltr1' = \text{Opt.lA } lltr2'$   
**unfolding**  $\text{Opt.lA}[OF \text{ cltr1}'] \text{Opt.lA}[OF \text{ cltr2}']$   
**using**  $A[\text{unfolded } \text{Opt.lA}[OF \text{ vltr1}(2)] \text{Opt.lA}[OF \text{ vltr2}(2)]] \text{tr1s1}' \text{tr2s2}'$   
**unfolding**  $\text{ltr1 } \text{ltr2 } \text{ttr1-eq } \text{ttr2-eq}$   
**unfolding**  $\text{lfilter-lappend-llist-of}$  **by**  $\text{simp}$

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

**qed**

**definition**  $\chi\chi \text{ s1 } \text{ltr1 } \text{tr1 } \text{s1}' \text{s1}'' \text{ltr1}' \equiv$   
 $\text{ltr1} = \text{lappend } (\text{llist-of } (\text{tr1 } \#\#\text{ s1}')) (\text{s1}'' \text{ } \text{ltr1}') \wedge$   
 $\text{Opt.validFromS } \text{s1} ((\text{tr1 } \#\#\text{ s1}') \#\#\text{ s1}'') \wedge$   
 $\text{never } \text{isIntO } \text{tr1} \wedge \neg \text{isIntO } \text{s1}' \wedge \neg \text{isIntO } \text{s1}'' \wedge$   
 $\text{never } \text{isSecO } \text{tr1} \wedge \text{isSecO } \text{s1}' \wedge$   
 $\text{Opt.lvalidFromS } \text{s1}'' (\text{s1}'' \text{ } \text{ltr1}') \wedge \text{Opt.lcompletedFrom } \text{s1}'' (\text{s1}'' \text{ } \text{ltr1}')$

**lemma**  $\text{isSecO-}\chi\chi$ :

**assumes**  $\text{vltr1: } \text{Opt.lvalidFromS } \text{s1 } \text{ltr1 } \text{Opt.lcompletedFrom } \text{s1 } \text{ltr1}$   
**and**  $\text{inter: } \text{lnever } \text{isIntO } \text{ltr1}$  **and**  $\text{isec: } \neg \text{lnever } \text{isSecO } \text{ltr1}$

shows  $\exists tr1\ s1'\ s1''\ ltr1'. \chi\chi\ s1\ ltr1\ tr1\ s1'\ s1''\ ltr1'$   
**proof** –  
 have 0:  $\exists s \in lset\ ltr1. isSecO\ s$  **using** *isec* **unfolding** *lset.pred-set* **by** *auto*  
 define *ttr1* **where** *ttr1*: *ttr1*  $\equiv ltakeUntil\ isSecO\ ltr1$   
 define *lltr1'* **where** *lltr1'*: *lltr1'*  $\equiv ldropUntil\ isSecO\ ltr1$   
 have *ltr1*: *ltr1* = *lappend* (*lset-of* *ttr1*) *lltr1'*  
**unfolding** *ttr1* *lltr1'* *lappend-ltakeUntil-ldropUntil*[*OF* 0] **..**  
 have 1: *ttr1*  $\neq []$   $\wedge$  *never isSecO* (*butlast* *ttr1*)  $\wedge$  *isSecO* (*last* *ttr1*)  
**unfolding** *ttr1*  
**using** *ltakeUntil-last*[*OF* 0] *ltakeUntil-not-Nil*[*OF* 0] *ltakeUntil-never-butlast*[*OF* 0]  
 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*  $\wedge$   $\neg$  *isIntO* *s1'*  $\wedge$  *lnever isIntO* *lltr1'*  
**using** *inter* **unfolding** *ltr1* *ttr1-eq*  
**unfolding** *lset-all-lappend-llist-of* *list-all-append* **by** *simp*  
 have *lfinite* *ltr1*  $\implies$  *s1' \neq llast* *ltr1*  
**by** (*metis* *Opt.final-not-isSec* *Opt.lcompletedFrom-def* *llast-last-llist-of* *tr1s1'*(2))  
*vltr1*(2))  
 hence *ne*: *lltr1' \neq []*  
**using** *ltr1* **unfolding** *ttr1-eq* **by** *auto*  
 then **obtain** *s1''* *ltr1'* **where** *lltr1'*: *lltr1'* = *s1''*  $\$$  *ltr1'*  
**by** (*meson* *lset.exhaust*)  
**show** *?thesis* **apply**(*rule* *exI*[*of* - *tr1*]) **apply**(*rule* *exI*[*of* - *s1'*])  
**apply**(*rule* *exI*[*of* - *s1''*]) **apply**(*rule* *exI*[*of* - *ltr1'*])  
**unfolding**  $\chi\chi$ -*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*  $\times$  *enat*  $\times$   
*'stA*  $\times$  *'stA* *lset*  $\times$   
*'stA*  $\times$  *'stA* *lset*  $\times$

```

    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*) =  
(*length ltr1* ≤ *Suc 0* ∨ *length 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* ≠ [] *length ltr1* ≤ *Suc 0* ∨ *length 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** *length ltr1* > *Suc 0* *length 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,statO*)))

**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* = [] ⇒ *ltr2* = [] ⇒ *lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)*  
= []

**unfolding** *lltrv1-def f-LNil* **by** *simp*

**lemma** *lltrv2-LNil[simp]*:

*ltr1* = [] ⇒ *ltr2* = [] ⇒ *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*  $\Delta$   
**and**  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**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*  $\neg$  *isIntO* *s2*  
**and** *len*:  $llength\ ltr1 > Suc\ 0$   $llength\ ltr2 > Suc\ 0$   
**and** *l*: *trn = L* *lnever isSecO* *ltr1*  
**shows**  $\exists w1'\ w2'\ s1'\ ltr1'\ trv1\ sv1'\ trv2\ sv2'\ statOO.$   
 $ltr1 = s1\ \$\ ltr1' \wedge validTransO\ (s1, s1') \wedge$   
 $Opt.lvalidFromS\ s1'\ ltr1' \wedge Opt.lcompletedFrom\ s1'\ ltr1' \wedge lnever\ isIntO\ ltr1' \wedge$   
 $\omega3\ \Delta\ w1\ w2\ w1'\ w2'\ s1\ s1'\ s2\ statA\ sv1\ trv1\ sv1'\ sv2\ trv2\ sv2'\ statOO \wedge$   
 $h-t\ (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $((trv1, sv1', trv2, sv2', statA, statOO),$   
 $(if\ trv1 = []\ then\ L\ else\ R,$   
 $w1', w2', s1', ltr1', s2, ltr2, statA, sv1', sv2', statOO))$

**proof**–  
**have** *s1*:  $\neg isIntO\ s1$  **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'*:  $ltr1' = tl\ ltr1$  **by** *auto*  
**have** *ltr1'ne*:  $ltr1' \neq []$  **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'*:  $s1' = lhd\ (tl\ ltr1)$   
**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*)<sup>+</sup>

**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'*)<sup>+</sup>

**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 unω 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*  $\Delta$   
**and**  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$   
**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*  $\neg$  *isIntO* *s2*  
**and** *len*: *llength* *ltr1*  $>$  *Suc* 0 *llength* *ltr2*  $>$  *Suc* 0  
**and** *l*: *trn* = *L* *lnever isSecO* *ltr1*  
**shows**  $\exists w1' w2' s1' ltr1' trv1 sv1' trv2 sv2' statOO$ .  
 $ltr1 = s1 \$ ltr1' \wedge validTransO (s1, s1') \wedge$   
 $Opt.lvalidFromS s1' ltr1' \wedge Opt.lcompletedFrom s1' ltr1' \wedge lnever isIntO ltr1' \wedge$   
 $\omega3 \Delta w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO \wedge$   
 $lltrv1 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\text{list-of } trv1) (lltrv1 (if trv1 = [] \text{ then } L \text{ else } R,$   
 $w1', w2', s1', ltr1', s2, ltr2, statA, sv1', sv2', statOO)) \wedge$   
 $lltrv2 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\text{list-of } trv2) (lltrv2 (if trv1 = [] \text{ then } L \text{ else } R,$   
 $w1', w2', s1', ltr1', s2, ltr2, statA, sv1', sv2', statOO))$

**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*  $\uparrow$ ]) **apply**(*rule exI*[*of* - *w2*  $\uparrow$ ])  
**apply**(*rule exI*[*of* - *s1*  $\uparrow$ ]) **apply**(*rule exI*[*of* - *ltr1*  $\uparrow$ ])  
**apply**(*rule exI*[*of* - *trv1*]) **apply**(*rule exI*[*of* - *sv1*  $\uparrow$ ])  
**apply**(*rule exI*[*of* - *trv2*]) **apply**(*rule exI*[*of* - *sv2*  $\uparrow$ ])  
**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*  $\Delta$   
**and**  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$

**and**  $r$ :  $\text{reachO } s1 \text{ reachO } s2 \text{ reachV } sv1 \text{ reachV } sv2$   
**and**  $\text{ltr1}$ :  $\text{Opt.lvalidFromS } s1 \text{ ltr1 Opt.lcompletedFrom } s1 \text{ ltr1}$   
**and**  $l'$ :  $\text{lnever isIntO } \text{ltr1} \neg \text{isIntO } s2$   
**and**  $\text{len}$ :  $\text{llength } \text{ltr1} > \text{Suc } 0 \text{ llength } \text{ltr2} > \text{Suc } 0$   
**and**  $l$ :  $\text{trn} = L \neg \text{lnever isSecO } \text{ltr1}$   
**shows**  $\exists w1' w2' tr1 s1' s1'' \text{ltr1}' trv1 sv1'' trv2 sv2'' \text{statOO}$ .  
 $\chi\chi s1 \text{ ltr1 } tr1 s1' s1'' \text{ltr1}' \wedge$   
 $\chi\chi^3 \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 \text{statA } sv1 trv1 sv1'' sv2 trv2 sv2'' \text{statOO}$   
 $\wedge$   
 $h\text{-t } (\text{trn}, w1, w2, s1, \text{ltr1}, s2, \text{ltr2}, \text{statA}, sv1, sv2, \text{statO}) =$   
 $((trv1, sv1'', trv2, sv2'', \text{statA}, \text{statOO}),$   
 $(R, w1', w2', s1'', s1'' \$ \text{ltr1}', s2, \text{ltr2}, \text{statA}, sv1'', sv2'', \text{statOO}))$   
**proof** –  
**have**  $s1$ :  $\neg \text{isIntO } s1$  **using**  $l' \text{ ltr1}$   
**by**  $(\text{metis Opt.lvalidFromS-def } l(2) \text{ lhd-LCons } \text{llist.exhaust } \text{llist.pred-inject}(1) \text{ llist.pred-inject}(2))$   
  
**obtain**  $tr1 s1' s1'' \text{ltr1}'$   
**where**  $\chi\chi$ :  $\chi\chi s1 \text{ ltr1 } tr1 s1' s1'' \text{ltr1}'$   
**using**  $\text{isSecO-}\chi\chi[\text{OF } \text{ltr1 } l'(1) l(2)]$  **by**  $\text{auto}$   
  
**define**  $tp$  **where**  
 $tp = (\text{SOME } k. \text{case } k \text{ of } (tr1, s1', s1'', \text{ltr1}') \Rightarrow$   
 $\chi\chi s1 \text{ ltr1 } tr1 s1' s1'' \text{ltr1}')$   
  
**have**  $0$ :  $\text{case } tp \text{ of } (tr1, s1', s1'', \text{ltr1}') \Rightarrow$   
 $\chi\chi s1 \text{ ltr1 } tr1 s1' s1'' \text{ltr1}'$   
**using**  $\chi\chi$  **unfolding**  $tp\text{-def}$  **apply**– **apply** $(\text{rule someI-ex})$   
**apply** $(\text{rule exI[of - } (tr1, s1', s1'', \text{ltr1}')])$  **by**  $\text{auto}$   
  
**obtain**  $tr1 s1' s1'' \text{ltr1}'$  **where**  
 $tp$ :  $tp = (tr1, s1', s1'', \text{ltr1}')$  **by** $(\text{cases } tp, \text{auto})$   
  
**have**  $\chi\chi$ :  $\chi\chi s1 \text{ ltr1 } tr1 s1' s1'' \text{ltr1}'$   
**using**  $0$  **unfolding**  $tp$  **by**  $\text{auto}$   
  
**obtain**  $w1' w2' trv1 sv1'' trv2 sv2'' \text{statOO}$   
**where**  $\chi\chi^3$ :  $\chi\chi^3 \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 \text{statA } sv1 trv1 sv1'' sv2 trv2$   
 $sv2'' \text{statOO}$   
**using**  $\text{unwindCond-ex-}\chi\chi^3[\text{OF } \text{unw } \Delta r, \text{of } tr1 s1' s1'']$   
**using**  $\chi\chi l' s1$  **unfolding**  $\chi\chi\text{-def}$  **by**  $\text{auto}$   
  
  
**define**  $tp'$  **where**  
 $tp' = (\text{SOME } k'. \text{case } k' \text{ of } (w1', w2', trv1, sv1'', trv2, sv2'', \text{statOO}) \Rightarrow$   
 $\chi\chi^3 \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 \text{statA } sv1 trv1 sv1'' sv2 trv2 sv2''$   
 $\text{statOO})$

```

have 1: case  $tp'$  of  $(w1',w2',trv1,sv1'',trv2,sv2'',statOO) \Rightarrow$ 
 $\chi^{3'} \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2''$ 
statOO
using  $\chi^{3'}$  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  $\chi^{3'}: \chi^{3'} \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2$ 
 $sv2'' statOO$ 
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 -  $trv1$ ]) 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  $\chi\chi$  .
subgoal using  $\chi^{3'}$  .
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  $\Delta$ 
and  $\Delta: \Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
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  $\neg$  isIntO s2
and len: llength ltr1 > Suc 0 llength ltr2 > Suc 0
and l: trn = L  $\neg$  lnever isSecO ltr1
shows  $\exists w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO.$ 
 $\chi\chi s1 ltr1 tr1 s1' s1'' ltr1' \wedge$ 
 $\chi^{3'} \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO$ 
 $\wedge$ 
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))
 $\wedge$ 
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  $unw$ :  $unwindCond \Delta$ 
and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
and  $r$ :  $reachO s1 reachO s2 reachV sv1 reachV sv2$ 
and  $ltr2$ :  $Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2$ 
and  $l'$ :  $\neg isIntO s1 lnever isIntO ltr2$ 
and  $len$ :  $llength ltr1 > Suc 0 llength ltr2 > Suc 0$ 
and  $l$ :  $trn = R lnever isSecO ltr2$ 
shows  $\exists w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO.$ 
   $ltr2 = s2 \$ ltr2' \wedge validTransO (s2, s2') \wedge$ 
   $Opt.lvalidFromS s2' ltr2' \wedge Opt.lcompletedFrom s2' ltr2' \wedge lnever isIntO ltr2' \wedge$ 

```

```

 $\omega_4 \Delta w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO \wedge$ 
 $h-t (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$ 
 $((trv1, sv1', trv2, sv2', statA, statOO),$ 
 $(if trv2 = [] then R else L,$ 
 $w1', w2', s1, ltr1, s2', ltr2', statA, sv1', sv2', statOO))$ 

```

**proof**–

```

have  $s2$ :  $\neg isIntO s2$  using  $l' ltr2$ 
by ( $metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil lhd-LCons$ 
 $l\list.exhaust l\list.pred-inject(2)$ )

obtain  $ltr2'$  where  $ltr2_4$ :  $ltr2 = s2 \$ ltr2'$ 
by ( $metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil l\list.exhaust-sel$ 
 $ltr2(1) ltr2(2)$ )
hence  $ltr2'$ :  $ltr2' = ltl ltr2$  by auto
have  $ltr2'ne$ :  $ltr2' \neq []$  using  $len(2)$  unfolding  $ltr2_4$ 
by ( $metis One-nat-def llength-LCons llength-LNil one-eSuc one-enat-def or-$ 
 $der-less-irrefl$ )
define  $s2'$  where  $s2'$ :  $s2' = lhd (ltl ltr2)$ 
have  $v_4$ :  $validTransO (s2, s2')$  and  $vv_4$ :  $Opt.lvalidFromS s2' ltr2' Opt.lcompletedFrom$ 

```

```

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':  $\neg$  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:  $\neg$  isSecO s2
  using l(2) ltr24 by auto

  obtain w1' w2' trv1 sv1' trv2 sv2' statOO
  where  $\omega_4$ :  $\omega_4 \Delta w1 w2 w1' w2' s1 s2$  (lhd (ltl ltr2)) statA sv1 trv1 sv1' sv2
trv2 sv2' statOO
  using unwindCond-ex- $\omega_4$ [OF unW  $\Delta r$  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)  $\Rightarrow$ 
 $\omega_4 \Delta w1 w2 w1' w2' s1 s2$  (lhd (ltl ltr2)) statA sv1 trv1 sv1' sv2 trv2 sv2'
statOO)

  have 1: case tp' of (w1',w2',trv1,sv1',trv2,sv2',statOO)  $\Rightarrow$ 
 $\omega_4 \Delta w1 w2 w1' w2' s1 s2 s2'$  statA sv1 trv1 sv1' sv2 trv2 sv2' statOO
  using  $\omega_4$  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  $\omega_4$ :  $\omega_4 \Delta w1 w2 w1' w2' s1 s2 s2'$  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  $\Delta: \Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
 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  $\exists w1'\ w2'\ s2'\ ltr2'\ trv1\ sv1'\ trv2\ sv2'\ statOO.$   
 $ltr2 = s2 \$ ltr2' \wedge validTransO (s2, s2') \wedge$   
 $Opt.lvalidFromS s2'\ ltr2' \wedge Opt.lcompletedFrom s2'\ ltr2' \wedge lnever isIntO ltr2' \wedge$

$\omega_4 \Delta\ w1\ w2\ w1'\ w2'\ s1\ s2\ s2'\ statA\ sv1\ trv1\ sv1'\ sv2\ trv2\ sv2'\ statOO \wedge$   
 $lltrv1 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\ list\ of\ trv1) (lltrv1 (if\ trv2 = []\ then\ R\ else\ L,$   
 $w1', w2', s1, ltr1, s2', ltr2', statA, sv1', sv2', statOO)) \wedge$   
 $lltrv2 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\ list\ of\ trv2) (lltrv2 (if\ trv2 = []\ then\ R\ else\ L,$   
 $w1', w2', s1, ltr1, s2', ltr2', statA, sv1', sv2', statOO))$

**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**  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**and**  $r$ :  $reachO\ s1\ reachO\ s2\ reachV\ sv1\ reachV\ sv2$   
**and**  $ltr2$ :  $Opt.lvalidFromS\ s2\ ltr2\ Opt.lcompletedFrom\ s2\ ltr2$   
**and**  $l'$ :  $\neg isIntO\ s1\ lnever\ isIntO\ ltr2$   
**and**  $len$ :  $llength\ ltr1 > Suc\ 0\ llength\ ltr2 > Suc\ 0$   
**and**  $l$ :  $trn = R\ \neg lnever\ isSecO\ ltr2$   
**shows**  $\exists w1'\ w2'\ tr2\ s2'\ s2''\ ltr2'\ trv1\ sv1''\ trv2\ sv2''\ statOO$ .  
 $\chi\chi\ s2\ ltr2\ tr2\ s2'\ s2''\ ltr2' \wedge$   
 $\chi\chi_4' \Delta\ w1\ w2\ w1'\ w2'\ s1\ s2\ tr2\ s2'\ s2''\ statA\ sv1\ trv1\ sv1''\ sv2\ trv2\ sv2''\ statOO$   
 $\wedge$   
 $h-t\ (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $((trv1, sv1'', trv2, sv2'', statA, statOO),$   
 $(L, w1', w2', s1, ltr1, s2'', s2'' \$ ltr2', statA, sv1'', sv2'', statOO))$   
**proof** –  
**have**  $s2$ :  $\neg isIntO\ s2$  **using**  $l'\ ltr2$   
**by** (*metis Simple-Transition-System.lvalidFromS-def l(2) lhd-LCons llist.pred-inject(1)*)  
  
 $l\text{list}.pred\text{-inject}(2)\ \text{neq-LNil-conv}$   
  
**obtain**  $tr2\ s2'\ s2''\ ltr2'$   
**where**  $\chi\chi$ :  $\chi\chi\ s2\ ltr2\ tr2\ s2'\ s2''\ ltr2'$   
**using**  $isSecO\text{-}\chi\chi[OF\ ltr2\ l'(2)\ l(2)]$  **by** *auto*  
  
**define**  $tp$  **where**  
 $tp = (SOME\ k.\ case\ k\ of\ (tr2, s2', s2'', ltr2') \Rightarrow$   
 $\chi\chi\ s2\ ltr2\ tr2\ s2'\ s2''\ ltr2')$   
  
**have**  $0$ :  $case\ tp\ of\ (tr2, s2', s2'', ltr2') \Rightarrow$   
 $\chi\chi\ s2\ ltr2\ tr2\ s2'\ s2''\ ltr2'$   
**using**  $\chi\chi$  **unfolding**  $tp\text{-def}$  **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**  $\chi\chi$ :  $\chi\chi\ s2\ ltr2\ tr2\ s2'\ s2''\ ltr2'$   
**using**  $0$  **unfolding**  $tp$  **by** *auto*  
  
**obtain**  $w1'\ w2'\ trv1\ sv1''\ trv2\ sv2''\ statOO$   
**where**  $\chi\chi_4'$ :  $\chi\chi_4'\ \Delta\ w1\ w2\ w1'\ w2'\ s1\ s2\ tr2\ s2'\ s2''\ statA\ sv1\ trv1\ sv1''\ sv2\ trv2$   
 $sv2''\ statOO$   
**using**  $unwindCond\text{-ex-}\chi\chi_4'[OF\ unw\ \Delta\ r,\ of\ tr2\ s2'\ s2'']$   
**using**  $\chi\chi\ l'\ s2$  **unfolding**  $\chi\chi\text{-def}$  **by** *auto*  
  
**define**  $tp'$  **where**  
 $tp' = (SOME\ k'.\ case\ k'\ of\ (w1', w2', trv1, sv1'', trv2, sv2'', statOO) \Rightarrow$   
 $\chi\chi_4'\ \Delta\ w1\ w2\ w1'\ w2'\ s1\ s2\ tr2\ s2'\ s2''\ statA\ sv1\ trv1\ sv1''\ sv2\ trv2\ sv2''$

```

statOO)

have 1: case tp' of (w1',w2',trv1,sv1'',trv2,sv2'',statOO) =>
   $\chi_4' \Delta w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2''$ 
statOO
using  $\chi_4'$  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  $\chi_4'$ :  $\chi_4' \Delta w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2$ 
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  $\chi\chi$  .
subgoal using  $\chi_4'$  .
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  $\Delta$ 
and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
and r: reachO s1 reachO s2 reachV sv1 reachV sv2
and ltr2: Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2
and l':  $\neg isIntO s1 lnever isIntO ltr2$ 
and len: llength ltr1 > Suc 0 llength ltr2 > Suc 0
and l: trn = R  $\neg lnever isSecO ltr2$ 
shows  $\exists w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO.$ 
   $\chi\chi s2 ltr2 tr2 s2' s2'' ltr2' \wedge$ 
   $\chi_4' \Delta w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO$ 
 $\wedge$ 
  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))
 $\wedge$ 
  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 unv 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  $\omega 3$  unfolding  $\omega 3$ -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  $\omega 3$  unfolding  $\omega 3$ -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  $\omega 3$  unfolding  $\omega 3$ -def sv1' by simp
    subgoal using  $\omega 3$  unfolding  $\omega 3$ -def
    by (metis Opt.reach.Step ωω(2) fst-conv r(1) snd-conv)
    subgoal by fact subgoal by fact
    subgoal using  $\omega 3$  unfolding  $\omega 3$ -def
    by (metis Nil-is-append-conv Van.reach-validFromS-reach last-snoc
not-Cons-self2 r(4))
    subgoal using  $\omega\omega$  by simp subgoal using  $\omega\omega$  by simp subgoal
using  $\omega\omega$  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  $\omega 3$  unfolding  $\omega 3$ -def
      by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
      subgoal by fact
      subgoal using  $\omega 3$  unfolding  $\omega 3$ -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 - trn1])
      apply(rule exI[of - w11]) apply(rule exI[of - w21]) apply(rule exI[of
- s11]) apply(rule exI[of - ltr11])
      apply(rule exI[of - s2]) apply(rule exI[of - ltr2])
      apply(rule exI[of - statA]) apply(rule exI[of - sv21]) apply(rule
exI[of - statOO])
      apply(intro conjI)
      subgoal ..
      subgoal using trv1 unfolding ltrv1' trn' by auto
      subgoal using  $\omega 3$  unfolding  $\omega 3$ -def by simp
      subgoal using  $\omega 3$  unfolding  $\omega 3$ -def
      by (metis Opt.reach.Step  $\omega\omega$ (2) fst-conv r(1) snd-conv)
      subgoal by fact
      subgoal using  $\omega 3$  unfolding  $\omega 3$ -def
      by (metis Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
      subgoal using  $\omega 3$  unfolding  $\omega 3$ -def
      by (metis Simple-Transition-System.reach-validFromS-reach r(4)
snoc-eq-iff-butlast)
      subgoal using  $\omega\omega$  by auto
      subgoal using  $\omega\omega$  by auto
      subgoal using  $\omega\omega$ 
      using llist-all-lappend-llist-of ltr1(3) by blast
      subgoal using  $\omega\omega$  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
      XX: XX s1 ltr1 tr1 s1' s1'' ltr1' and
      X3': X3'  $\Delta$  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 un $\omega$   $\Delta$  r ltr1 isi4 current]
      unfolding ltrv1 by blast
      define ltrv1' where ltrv1': ltrv1'  $\equiv$  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.lvalidFromS-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  $\chi^{3'}$  unfolding  $\chi^{3'-def}$ 
  by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
  subgoal using  $\chi^{3'}$  unfolding  $\chi^{3'-def}$  by simp
  subgoal using  $\chi^{3'}$  unfolding  $\chi^{3'-def}$ 
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  $\chi^{3'}$  unfolding  $\chi^{3'-def}$  by simp
  subgoal using  $\chi^{3'}$  unfolding  $\chi^{3'-def}$ 
by (metis Simple-Transition-System.reach-validFromS-reach  $\chi\chi$   $\chi\chi$ -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  $trn = R$  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
   $\omega\omega$ :  $ltr2 = s2$   $\$$   $ltr2'$  validTransO ( $s2, s2'$ ) Opt.lvalidFromS  $s2' ltr2'$ 
  lcompletedFromO  $s2' ltr2' lnever isIntO ltr2'$  and
   $\omega4$ :  $\omega4 \Delta w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2'$ 
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 un $\omega$   $\Delta r$   $ltr2(1,2)$   $isi3$   $ltr2(3)$   $current$ ]
  unfolding  $ltrv1$  by blast
  define  $ltrv1'$  where  $ltrv1'$ :  $ltrv1' \equiv$  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  $\omega4$  unfolding  $\omega4$ -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  $\omega4$  unfolding  $\omega4$ -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  $\omega4$  unfolding  $\omega4$ -def  $sv1'$  by simp
  subgoal by fact
  subgoal using  $\omega4$  unfolding  $\omega4$ -def
  by (metis Opt.reach.Step  $\omega\omega(2)$  fst-conv  $r(2)$  snd-conv)
  subgoal by fact
  subgoal using  $\omega4$  unfolding  $\omega4$ -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  $\omega\omega$  by simp subgoal using  $\omega\omega$  by simp subgoal
using  $\omega\omega$  by simp . .
next
  case False note trv1 = False
  show ?thesis
  apply(rule Van.lvalidFromS-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  $\omega_4$  unfolding  $\omega_4$ -def
  by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
  subgoal by fact
  subgoal using  $\omega_4$  unfolding  $\omega_4$ -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  $\omega_4$  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 Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
  subgoal using  $\omega_4$  unfolding  $\omega_4$ -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  $\omega\omega$  by auto
  subgoal using  $\omega\omega$  by auto
  subgoal using  $\omega\omega$ 
  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  $\chi\chi$  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 . .
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  $\chi_4'$  unfolding  $\chi_4'$ -def
by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
  subgoal using trv1 .
  subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -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  $\chi_4'$  unfolding  $\chi_4'$ -def by simp
  subgoal by fact
  subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -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  $\Delta$ 
and  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$ 
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  $\exists trn\ w1\ w2\ s1\ ltr1\ s2\ ltr2\ statA\ sv1\ statO.$ 
       $n = w2 \wedge$ 
       $ltrv2 = lltrv2\ (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) \wedge$ 
       $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$ 
       $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$ 
       $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge lnever\ isIntO\ ltr1 \wedge$ 
       $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge lnever\ isIntO\ ltr2$ 
    hence Van.llvalidFromS n sv2 ltrv2
    proof(coinduct rule: Van.llvalidFromS.coinduct[of  $\lambda n\ sv2\ ltrv2.$ 
       $\exists trn\ w1\ w2\ s1\ ltr1\ s2\ ltr2\ statA\ sv1\ statO.$ 
         $n = w2 \wedge$ 
         $ltrv2 = lltrv2\ (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) \wedge$ 
         $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$ 
         $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$ 
         $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge lnever\ isIntO\ ltr1 \wedge$ 
         $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge lnever\ isIntO\ ltr2$ ])
    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  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$ 
      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:  $\neg$  isIntO s1 using ltr1
by (metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel
llist.pred-inject(2))
have isi4:  $\neg$  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 = []  $\wedge$  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  $\neq$  []  $\vee$  ltr2  $\neq$  [] by auto
show ?thesis proof(cases llength ltr1  $\leq$  Suc 0  $\vee$  llength ltr2  $\leq$  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
 $\omega\omega$ : ltr1 = s1 $ ltr1' validTransO (s1, s1') Opt.lvalidFromS s1' ltr1'
lcompletedFromO s1' ltr1' lnever isIntO ltr1' and
 $\omega\omega\omega$ :  $\omega\omega\omega$   $\Delta$  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 un $\omega$   $\Delta$  r ltr1 isi4 current]
unfolding ltrv2 by blast
define ltrv2' where ltrv2': ltrv2'  $\equiv$  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  $\omega 3$  unfolding  $\omega 3$ -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  $\omega 3$  unfolding  $\omega 3$ -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  $\omega 3$  unfolding  $\omega 3$ -def sv2' by simp
  subgoal using  $\omega 3$  unfolding  $\omega 3$ -def
  by (metis Opt.reach.Step  $\omega\omega$ (2) fst-conv r(1) snd-conv)
  subgoal by fact
  subgoal using  $\omega 3$  unfolding  $\omega 3$ -def
  by (metis Nil-is-append-conv Van.reach-validFromS-reach last-snoc
not-Cons-self2 r(3))
  subgoal by fact
  subgoal using  $\omega\omega$  by simp subgoal using  $\omega\omega$  by simp subgoal
using  $\omega\omega$  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  $\omega 3$  unfolding  $\omega 3$ -def
by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
subgoal by fact
subgoal using  $\omega 3$  unfolding  $\omega 3$ -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 χ3'-def Simple-Transition-System.validFromS-Cons-iff χ3'
append.simps(1))
  have w2' < w2 using trv2 χ3' unfolding χ3'-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  $\chi^{3'}$  unfolding  $\chi^{3'}$ -def
  by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
  subgoal by fact
  subgoal using  $\chi^{3'}$  unfolding  $\chi^{3'}$ -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  $\chi^{\beta'}$  unfolding  $\chi^{\beta'}$ -def by simp
  subgoal using  $\chi^{\beta'}$  unfolding  $\chi^{\beta'}$ -def
    by (metis Simple-Transition-System.reach-validFromS-reach  $\chi\chi$ 
 $\chi\chi$ -def
append-is-Nil-conv last-snoc not-Cons-self2 r(1))
  subgoal by fact
  subgoal using  $\chi^{\beta'}$  unfolding  $\chi^{\beta'}$ -def
    by (metis Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
  subgoal using  $\chi^{\beta'}$  unfolding  $\chi^{\beta'}$ -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
  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
 $\omega\omega$ : ltr2 = s2 $ ltr2' validTransO (s2, s2') Opt.lvalidFromS s2' ltr2'
lcompletedFromO s2' ltr2' lnever isInto ltr2' and
 $\omega\omega$ :  $\omega\omega \Delta w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2'$ 
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 un $\omega \Delta r$  ltr2(1,2) isi3 ltr2(3) current]
  unfolding ltrv2 by blast
  define ltrv2' where ltrv2': ltrv2'  $\equiv$  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  $\omega_4$  unfolding  $\omega_4$ -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  $\omega_4$  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 Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
      subgoal using  $\omega_4$  unfolding  $\omega_4$ -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  $\omega\omega$  by auto
      subgoal using  $\omega\omega$  by auto
      subgoal using  $\omega\omega$ 
      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
   $\chi\chi$ :  $\chi\chi$  s2 ltr2 tr2 s2' s2'' ltr2' and
   $\chi_4'$ :  $\chi_4'$   $\Delta$  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 un $\omega$   $\Delta$  r ltr2(1,2) isi3 ltr2(3)
current]
  unfolding ltrv2 by blast
  define ltrv2' where ltrv2': ltrv2'  $\equiv$  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  $\neq$  [] using  $\chi_4'$  unfolding  $\chi_4'$ -def by auto

  show ?thesis
  apply(rule Van.linvalidFromS-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  $\chi_4'$  unfolding  $\chi_4'$ -def
  by (metis Nil-is-append-conv Van.validFromS-def Van.validS-append1
hd-append2)
  subgoal using trv2 .
  subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -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  $\chi_4'$  unfolding  $\chi_4'$ -def by simp
  subgoal by fact
  subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -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
}
```

```

thus ?thesis apply–apply(rule Van.lvalidFromS-imp-lvalidFromS)
using assms by blast
qed

```

```

lemma lcompletedFrom-lltrv1:
assumes unw: unwindCond  $\Delta$ 
and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
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  $\neq [] \wedge$  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  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
      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:  $\neg$  isIntO s1 using ltr1
      by (metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel
llist.pred-inject(2))
      have isi4:  $\neg$  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 = [[]]  $\wedge$  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  $\neq [] \vee$  ltr2  $\neq []$  by auto
      show ?thesis proof(cases length ltr1  $\leq$  Suc 0  $\vee$  length ltr2  $\leq$  Suc 0)
        case True note ltr14 = ltr14 True
          hence ltrv1: list-of ltrv1 = [sv1] unfolding ltrv1 by simp
          have length ltr1 = Suc 0  $\vee$  length 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-tnull-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 unω 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)  $\wedge$  w1' < w1
using trv1ne lfin lfin' by (simp add: list-of-lappend lltrv1)

have 0: list-of ltrv1'  $\neq$  []  $\wedge$  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  $\omega_4$  unfolding  $\omega_4$ -def by simp
    subgoal by fact
    subgoal using r(2)  $\omega\omega$  by (metis Opt.reach.Step fst-conv snd-conv)
    subgoal using  $\omega_4$  unfolding  $\omega_4$ -def
      by (metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2 r(3))
    subgoal using  $\omega_4$  unfolding  $\omega_4$ -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  $\omega_4$  unfolding  $\omega_4$ -def by simp
    subgoal by fact
    subgoal 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 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
   $\chi\chi$ :  $\chi\chi$  s2 ltr2 tr2 s2' s2'' ltr2' and
   $\chi\chi'$ :  $\chi\chi'$   $\Delta$  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 un $\omega$   $\Delta$  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 ltrv1) ltrv1'
  unfolding ltrv1 ltrv1' ..

  have trv1ne: ltrv1 ≠ [] ∨ w1' < w1 using  $\chi_4'$  unfolding  $\chi_4'$ -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  $\chi_4'$  unfolding  $\chi_4'$ -def by simp
      subgoal by fact
      subgoal using r(2)  $\chi\chi$  unfolding  $\chi\chi$ -def
        by (metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2)
      subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -def
        by (metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2 r(3))
      subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -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  $\chi\chi$  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 .
  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: lfn' list-of-lappend)
  qed
  qed
  qed
  qed
  qed
  }
  thus ?thesis unfolding Van.lcompletedFrom-def by auto
  qed

```

```

lemma lcompletedFrom-lltrv2:
  assumes unw: unwindCond  $\Delta$ 
  and  $\Delta$ :  $\Delta \infty w_1 w_2 s_1 s_2 \text{statA } sv_1 sv_2 \text{statO}$ 
  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 lfn: lfinite ltrv2
      hence list-of ltrv2  $\neq [] \wedge$  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 lfn: lfinite ltrv2
          and  $\Delta$ :  $\Delta \infty w_1 w_2 s_1 s_2 \text{statA } sv_1 sv_2 \text{statO}$ 
          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:  $\neg$  isIntO s1 using ltr1
          by (metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel
            llist.pred-inject(2))
          have isi4:  $\neg$  isIntO s2 using ltr2
          by (metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel

```

*l*list.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' ..

```

```

auto
  have trv2ne: trv2 ≠ [] ∨ w2' < w2 using ω3 unfolding ω3-def by
  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
  XX: XX s1 ltr1 tr1 s1' s1'' ltr1' and
  X3': X3' 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  $\chi\chi$  unfolding  $\chi\chi$ -def by simp
    subgoal using  $\chi\chi$  unfolding  $\chi\chi$ -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  $\chi^{3'}$  unfolding  $\chi^{3'}$ -def by simp
subgoal using  $\chi\chi$  unfolding  $\chi\chi$ -def
  by (metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc

```

```

not-Cons-self2 r(1))
  subgoal by fact
  subgoal using  $\chi\beta'$  unfolding  $\chi\beta'$ -def
  by (metis Van.reach-validFromS-reach r(3) snoc-eq-iff-butlast)
  subgoal using  $\chi\beta'$  unfolding  $\chi\beta'$ -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 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 .
qed
show ?thesis unfolding lltrv2 using 0
by (simp add: lfn' 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
 $\omega\omega$ : ltr2 = s2 $ ltr2' validTransO (s2, s2') Opt.lvalidFromS s2' ltr2'
lcompletedFromO s2' ltr2' lnever isIntO ltr2' and
 $\omega4$ :  $\omega4 \Delta w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2'$ 
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  $\Delta r$  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  $\neq$  []  $\vee$   $w2' < w2$  using  $\omega4$  unfolding  $\omega4$ -def by
auto
have lfn': lfinite ltrv2'
using lfn trv2ne unfolding lltrv2 by simp
have len: length (list-of ltrv2') < length (list-of ltrv2)  $\vee$ 
length (list-of ltrv2') = length (list-of ltrv2)  $\wedge$   $w2' < w2$ 
using trv2ne lfn lfn' by (simp add: list-of-lappend lltrv2)

have 0: list-of ltrv2'  $\neq$  []  $\wedge$  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  $\omega_4$  unfolding  $\omega_4$ -def by simp
  subgoal by fact
  subgoal using r(2)  $\omega\omega$  by (metis Opt.reach.Step fst-conv snd-conv)
  subgoal using  $\omega_4$  unfolding  $\omega_4$ -def
  by (metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2 r(3))
  subgoal using  $\omega_4$  unfolding  $\omega_4$ -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  $\omega_4$  unfolding  $\omega_4$ -def by simp
  subgoal by fact
  subgoal 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 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: lfn' list-of-lappend)
next
case False note current = current False
obtain w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO where
 $\chi\chi$ :  $\chi\chi$  s2 ltr2 tr2 s2' s2'' ltr2' and
 $\chi_4'$ :  $\chi_4'$   $\Delta$  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 un $\omega$   $\Delta$  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 λtrn 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 ω3 unfolding ω3-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 - ltr1]) apply(rule exI[of - ltr1])
apply(rule exI[of - w2])
apply(intro conjI)
  subgoal by fact
  subgoal unfolding nL .. subgoal unfolding nR ..
  subgoal unfolding ltr1 trv1 by simp
  subgoal unfolding  $\omega\omega(1)$  by simp
  subgoal by fact subgoal unfolding trv1 using  $\omega3$ -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 ltr1' by simp
  subgoal using  $\omega3$  unfolding  $\omega3$ -def by simp
  subgoal using  $\omega3$  unfolding  $\omega3$ -def
  by (metis Opt.reach.Step  $\omega\omega(2)$  fst-conv r(1) snd-conv)
  subgoal by fact
  subgoal using  $\omega3$  unfolding  $\omega3$ -def
  by (metis Simple-Transition-System.reach-validFromS-reach r(3))
snoc-eq-iff-butlast)
  subgoal using  $\omega3$  unfolding  $\omega3$ -def
  by (metis Simple-Transition-System.reach-validFromS-reach r(4))
snoc-eq-iff-butlast)
  subgoal using  $\omega\omega$  by auto
  subgoal using  $\omega\omega$  by auto
  subgoal using  $\omega\omega$ 
  using llist-all-lappend-llist-of ltr1(3) by blast
  subgoal using  $\omega\omega$  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 - ltr1]) 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  $\omega\omega(1)$  by simp
subgoal by fact
subgoal using  $\omega\exists$  unfolding  $\omega\exists$ -def by simp
subgoal using ltr1(3)  $\omega\exists$  unfolding  $\omega\exists$ -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  $\omega\exists$  unfolding  $\omega\exists$ -def by simp
subgoal using  $\omega\exists$  unfolding  $\omega\exists$ -def
by (metis Opt.reach.Step  $\omega\omega(2)$  fst-conv r(1) snd-conv)
subgoal by fact
subgoal using  $\omega\exists$  unfolding  $\omega\exists$ -def
by (metis Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
subgoal using  $\omega\exists$  unfolding  $\omega\exists$ -def
by (metis Simple-Transition-System.reach-validFromS-reach r(4)
snoc-eq-iff-butlast)
subgoal using  $\omega\omega$  by auto
subgoal using  $\omega\omega$  by auto
subgoal using  $\omega\omega$ 
using llist-all-lappend-llist-of ltr1(3) by blast
subgoal using  $\omega\omega$  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
 $\chi\chi$ :  $\chi\chi$  s1 ltr1 tr1 s1' s1'' ltr1' and
 $\chi\exists'$ :  $\chi\exists'$   $\Delta$  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 un $\omega$   $\Delta$  r ltr1 isi4 current]
unfolding ltrv1 by blast
define ltrv1' where ltrv1': ltrv1'  $\equiv$  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  $\chi\chi$  unfolding  $\chi\chi$ -def by simp
  subgoal using  $\chi\beta'$  unfolding  $\chi\beta'$ -def by simp
  subgoal by simp
  subgoal using  $\chi\beta'$  unfolding  $\chi\beta'$ -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  $\chi\beta'$  unfolding  $\chi\beta'$ -def by simp
    subgoal using  $\chi\beta'$  unfolding  $\chi\beta'$ -def
by (metis Simple-Transition-System.reach-validFromS-reach  $\chi\chi$   $\chi\chi$ -def

  append-is-Nil-conv last-snoc not-Cons-self2 r(1))
  subgoal by fact
  subgoal using  $\chi\beta'$  unfolding  $\chi\beta'$ -def
  by (metis Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
  subgoal using  $\chi\beta'$  unfolding  $\chi\beta'$ -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
   $\omega4$ :  $\omega4\ \Delta\ w1\ w2\ w1'\ w2'\ s1\ s2\ s2'\ statA\ sv1\ trv1\ sv1'\ sv2\ trv2\ sv2'$ 
  statOO
  and  $trn'$ :  $trn' = (if\ trv2 = []\ then\ R\ else\ L)$ 
  and  $ltrv1$ :  $ltrv1 =$ 
   $lappend\ (l\ list\ of\ trv1)\ (lltrv1\ (trn',\ w1',\ w2',\ s1,\ ltr1,\ s2',\ ltr2',\ statA,$ 
   $sv1',\ sv2',\ statOO))$ 
  using  $lltrv1\ lltrv2\ lnever\ R[OF\ un\ w\ \Delta\ r\ ltr2(1,2)\ isi3\ ltr2(3)\ current]$ 
  unfolding  $ltrv1$  by blast
  define  $ltrv1'$  where  $ltrv1'$ :  $ltrv1' \equiv lltrv1\ (trn',\ w1',\ w2',\ s1,\ ltr1,\ s2',$ 
   $ltr2',\ statA,\ sv1',\ sv2',\ statOO)$ 
  have  $ltrv1$ :  $ltrv1 = lappend\ (l\ list\ of\ trv1)\ ltrv1'$ 
  unfolding  $ltrv1\ ltrv1'$  ..
  have  $nev1$ : never isSecV trv1 using  $\omega4$  unfolding  $\omega4\text{-def}$  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  $\omega4\ trv2$  unfolding  $\omega4\text{-def}$  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  $\omega4$  unfolding  $\omega4\text{-def}$  by simp
    subgoal by fact
    subgoal using  $\omega4$  unfolding  $\omega4\text{-def}$ 

```

```

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  $\omega\omega(2)$  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 . .
qed
next
case False note current = current False
obtain w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO where

```

```

     $\chi\chi$ :  $\chi\chi$   $s2$   $ltr2$   $tr2$   $s2'$   $s2''$   $ltr2'$  and
     $\chi4'$ :  $\chi4'$   $\Delta$   $w1$   $w2$   $w1'$   $w2'$   $s1$   $s2$   $tr2$   $s2'$   $s2''$   $statA$   $sv1$   $trv1$   $sv1''$   $sv2$ 
 $trv2$   $sv2''$   $statOO$ 
    and  $ltrv1$ :  $ltrv1$  =
     $lappend$  ( $l$ list-of  $trv1$ ) ( $lltrv1$  ( $L$ ,  $w1'$ ,  $w2'$ ,  $s1$ ,  $ltr1$ ,  $s2''$ ,  $s2''$   $\$$   $ltr2'$ ,
 $statA$ ,  $sv1''$ ,  $sv2''$ ,  $statOO$ ))
    using  $lltrv1$ - $lltrv2$ -not-lnever-R[ $OF$   $unw$   $\Delta$   $r$   $ltr2(1,2)$   $isi3$   $ltr2(3)$ 
 $current$ ]
    unfolding  $ltrv1$  by  $blast$ 
    define  $ltrv1'$  where  $ltrv1'$ :  $ltrv1' \equiv lltrv1$  ( $L$ ,  $w1'$ ,  $w2'$ ,  $s1$ ,  $ltr1$ ,  $s2''$ ,
 $s2''$   $\$$   $ltr2'$ ,  $statA$ ,  $sv1''$ ,  $sv2''$ ,  $statOO$ )
    have  $ltrv1$ :  $ltrv1 = lappend$  ( $l$ list-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  $\chi4'$  unfolding  $\chi4'$ - $def$  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  $\chi4'$  unfolding  $\chi4'$ - $def$  by  $simp$ 
  subgoal by  $fact$ 
  subgoal using  $r(2)$   $\chi\chi$  unfolding  $\chi\chi$ - $def$ 
  by ( $metis$   $Opt.reach-validFromS-reach$   $append-is-Nil-conv$   $last-snoc$ 
 $not-Cons-self2$ )
  subgoal using  $r(3)$   $\chi4'$  unfolding  $\chi4'$ - $def$ 
  by ( $metis$   $Van.reach-validFromS-reach$   $snoc-eq-iff-butlast$ )
  subgoal using  $r(4)$   $\chi4'$  unfolding  $\chi4'$ - $def$ 
  by ( $metis$   $Van.reach-validFromS-reach$   $snoc-eq-iff-butlast$ )
  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  $l$ list-all-lappend-llist-of  $ltr2(3)$  by  $blast$  . .
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  $\Delta$ 
and  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$ 
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  $\exists w1\ w2\ s1\ s2\ ltr1\ statA\ sv1\ sv2\ statO.$ 

```

```

   $nL = w1 \wedge nR = w2 \wedge$ 

```

```

   $ltrv2 = lltrv2 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) \wedge$ 

```

```

   $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$ 

```

```

   $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$ 

```

```

   $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge lnever\ isIntO\ ltr1 \wedge$ 

```

```

   $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge lnever\ isIntO\ ltr2$ 

```

```

hence TwoFuncPred.sameFM2 isSecV isSecO getSecV getSecO trn nL nR ltrv2
ltr2

```

```

proof(coinduct rule: TwoFuncPred.sameFM2.coinduct[of  $\lambda trn\ nL\ nR\ ltrv2\ ltr2.$ 

```

```

   $\exists w1\ w2\ s1\ s2\ ltr1\ statA\ sv1\ sv2\ statO.$ 

```

```

   $nL = w1 \wedge nR = w2 \wedge$ 

```

```

   $ltrv2 = lltrv2 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) \wedge$ 

```

```

   $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$ 

```

```

   $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$ 

```

```

   $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge lnever\ isIntO\ ltr1 \wedge$ 

```

```

   $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge lnever\ isIntO\ ltr2,$ 

```

```

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  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$ 

```

```

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:  $\neg$  isIntO s1 using ltr1
by (metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel
llist.pred-inject(2))
have isi4:  $\neg$  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 = []  $\wedge$  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  $\neq$  []  $\vee$  ltr2  $\neq$  [] by auto
show ?thesis proof(cases llength ltr1  $\leq$  Suc 0  $\vee$  llength ltr2  $\leq$  Suc 0)
  case True note ltr14 = ltr14 True
  hence ltrv2: ltrv2 = [[sv2]] unfolding ltrv2 by simp
  have llength ltr1 = Suc 0  $\vee$  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  $\vee$  finalO s2
  using Opt.lcompletedFrom-singl ltr1(1) ltr1(2) ltr2(1) ltr2(2) by blast
  hence fs2: finalO s2
  using  $\Delta$  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  $\Delta$  fs2 r(1) r(2) r(3) r(4) unW unwindCond-final by blast
  have isv24:  $\neg$  isSecV sv2  $\wedge$   $\neg$  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
   $\omega$ : 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: lltrv2 =
  lappend (llist-of trv2) (lltrv2 (trn', w1', w2', s1', ltr1', s2, ltr2, statA,
sv1', sv2', statOO))
using lltrv1-lltrv2-lnever-L[OF unω Δ r ltr1 isi4 current]
unfolding lltrv2 by blast
define lltrv2' where lltrv2': lltrv2' ≡ lltrv2 (trn', w1', w2', s1', ltr1',
s2, ltr2, statA, sv1', sv2', statOO)
have lltrv2: lltrv2 = lappend (llist-of trv2) lltrv2'
unfolding lltrv2 lltrv2' ..
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 - lltrv2 ↑]) 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 lltrv2' 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  $\omega\omega$  by auto
      subgoal using  $\omega\omega$  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  $\omega\omega(1)$  by simp
subgoal using  $\omega^3$  unfolding  $\omega^3$ -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  $\omega^3$  unfolding  $\omega^3$ -def by simp
  subgoal using  $\omega^3$  unfolding  $\omega^3$ -def
  by (metis Opt.reach.Step  $\omega\omega(2)$  fst-conv  $r(1)$  snd-conv)
  subgoal by fact
  subgoal using  $\omega^3$  unfolding  $\omega^3$ -def
  by (metis Van.reach-validFromS-reach  $r(3)$  snoc-eq-iff-butlast)
  subgoal using  $\omega^3$  unfolding  $\omega^3$ -def
  by (metis Van.reach-validFromS-reach  $r(4)$  snoc-eq-iff-butlast)
  subgoal using  $\omega\omega$  by auto
  subgoal using  $\omega\omega$  by auto
  subgoal using  $\omega\omega$  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
 $\chi\chi$ :  $\chi\chi$   $s1$   $ltr1$   $tr1$   $s1' s1'' ltr1'$  and
 $\chi^3$ :  $\chi^3 \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2$ 
 $trv2 sv2'' statOO$ 
and  $lltrv2$ :  $ltrv2 =$ 
lappend (llist-of  $trv2$ ) ( $lltrv2$  ( $R, w1', w2', s1'', s1''$  $  $ltr1', s2, ltr2, statA, sv1'', sv2'', statOO$ ))

```

```

using ltrv1-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  $\chi^{3'}$  unfolding  $\chi^{3'-def}$  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  $\chi^{3'}$  unfolding  $\chi^{3'-def}$  by simp
  subgoal using r(1) χχ unfolding  $\chi\chi-def$ 
  by (metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2)
  subgoal by fact
  subgoal using r(3) χ^{3'} unfolding  $\chi^{3'-def}$ 
  by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
  subgoal using r(4) χ^{3'} unfolding  $\chi^{3'-def}$ 
  by (metis Van.reach-validFromS-reach 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 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
ωw: 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 unω 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 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  $\omega_4$  unfolding  $\omega_4$ -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  $\omega\omega$  by auto
  subgoal using  $\omega\omega$  by auto
  subgoal using  $\omega\omega$ 
  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  $\omega\omega(1)$  by simp
subgoal by fact
subgoal using  $\omega_4$  unfolding  $\omega_4$ -def by simp
subgoal using ltr1(3)  $\omega_4$  unfolding  $\omega_4$ -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  $\omega_4$  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 Simple-Transition-System.reach-validFromS-reach r(3))
snoc-eq-iff-butlast)
subgoal using  $\omega_4$  unfolding  $\omega_4$ -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  $\omega\omega$  by auto
    subgoal using  $\omega\omega$  by auto
    subgoal using  $\omega\omega$ 
    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
 $\chi\chi: \chi\chi s2 ltr2 tr2 s2' s2'' ltr2'$  and
 $\chi4': \chi4' \Delta w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2$ 
 $trv2 sv2'' statOO$ 
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  $unw \Delta r ltr2(1,2) isi3 ltr2(3)$ 
current]
  unfolding  $ltrv2$  by blast
define  $ltrv2'$  where  $ltrv2': ltrv2' \equiv lltrv2 (L, w1', w2', s1, ltr1, s2'',$ 
 $s2'' \$ ltr2', statA, sv1'', sv2'', statOO)$ 
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  $\chi\chi$  unfolding  $\chi\chi$ -def by simp
  subgoal using  $\chi4'$  unfolding  $\chi4'$ -def by simp
  subgoal by simp
  subgoal using  $\chi4'$  unfolding  $\chi4'$ -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  $\chi4'$  unfolding  $\chi4'$ -def by simp
  subgoal by fact

```

```

    subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -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 Van.reach-validFromS-reach r(3) snoc-eq-iff-butlast)
    subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -def
  by (metis Van.reach-validFromS-reach r(4) snoc-eq-iff-butlast)
    subgoal by fact subgoal by fact subgoal by fact
    subgoal using  $\chi\chi$  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
}
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  $\Delta$

and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$

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-lltrv1[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  $\exists trn w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO$ .

nL = w1  $\wedge$  nR = w2  $\wedge$

ltrv1 = lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\wedge$

ltrv2 = lltrv2 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\wedge$

$\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO \wedge$

reachO s1  $\wedge$  reachO s2  $\wedge$  reachV sv1  $\wedge$  reachV sv2  $\wedge$

Opt.lvalidFromS s1 ltr1  $\wedge$  Opt.lcompletedFrom s1 ltr1  $\wedge$  lnever isIntO ltr1  $\wedge$

Opt.lvalidFromS s2 ltr2  $\wedge$  Opt.lcompletedFrom s2 ltr2  $\wedge$  lnever isIntO ltr2

**hence** *TwoFuncPred.sameFM isIntV isIntV getActV getActV nL nR ltrv1 ltrv2*  
**proof**(*coinduct rule: TwoFuncPred.sameFM.coinduct[of λnL nR ltrv1 ltrv2.*  
 $\exists$  *trn w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO.*  
 $nL = w1 \wedge nR = w2 \wedge$   
 $ltrv1 = lltrv1 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) \wedge$   
 $ltrv2 = lltrv2 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) \wedge$   
 $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO \wedge$   
 $reachO s1 \wedge reachO s2 \wedge reachV sv1 \wedge reachV sv2 \wedge$   
 $Opt.lvalidFromS s1 ltr1 \wedge Opt.lcompletedFrom s1 ltr1 \wedge lnever isIntO ltr1 \wedge$   
 $Opt.lvalidFromS s2 ltr2 \wedge Opt.lcompletedFrom s2 ltr2 \wedge lnever isIntO ltr2$ ])  
**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**  $\Delta: \Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$   
**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**  $isi3: \neg isIntO s1$  **using**  $ltr1$   
**by** (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel*  
*llist.pred-inject(2)*)  
**have**  $isi4: \neg isIntO s2$  **using**  $ltr2$   
**by** (*metis Opt.lcompletedFrom-def Opt.lvalidFromS-def lfinite-LNil llist.exhaust-sel*  
*llist.pred-inject(2)*)  
  
**show**  $?case$  **proof**(*cases*  $ltr1 = [] \wedge ltr2 = []$ )  
**case** *True* **note**  $ltr14 = True$   
**hence**  $ltrv1: ltrv1 = []$  **unfolding**  $ltrv1$  **by** *simp*  
**show**  $?thesis$  **using**  $ltr14$  **unfolding**  $ltrv1 ltrv2$  **apply-apply**(*rule Two-*  
*FuncPred.sameFM-selectLNil*) **by auto**  
**next**  
**case** *False* **hence**  $ltr14: ltr1 \neq [] \vee ltr2 \neq []$  **by auto**  
**show**  $?thesis$  **proof**(*cases*  $llength ltr1 \leq Suc 0 \vee llength ltr2 \leq Suc 0$ )  
**case** *True* **note**  $ltr14 = ltr14 True$   
**hence**  $ltrv1: ltrv1 = [[sv1]]$  **and**  $ltrv2: ltrv2 = [[sv2]]$  **unfolding**  $ltrv1 ltrv2$   
**by auto**  
**have**  $llength ltr1 = Suc 0 \vee llength ltr2 = Suc 0$   
**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**  $finalO s1 \vee finalO s2$   
**using** *Opt.lcompletedFrom-singl ltr1(1) ltr1(2) ltr2(1) ltr2(2)* **by blast**  
**hence**  $fs1: finalO s1 \wedge finalO s2$   
**using**  $\Delta r(1) r(2) r(3) r(4) unw unwindCond-def$  **by auto**  
  
**have**  $fsv12: finalV sv1 \wedge finalV sv2$

```

using  $\Delta$  fs1 r(1) r(2) r(3) r(4) unW unwindCond-final by blast
have isv12:  $\neg$  isIntV sv1  $\wedge$   $\neg$  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
 $\omega\omega$ : ltr1 = s1 $ ltr1' validTransO (s1, s1') Opt.lvalidFromS s1' ltr1'
lcompletedFromO s1' ltr1' lnever isIntO ltr1' and
 $\omega\beta$ :  $\omega\beta$   $\Delta$  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  $\Delta$  r ltr1 isi4 current]
unfolding ltrv1 ltrv2 by blast
define ltrv1' where ltrv1': ltrv1'  $\equiv$  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'  $\equiv$  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  $\omega\beta$  unfolding  $\omega\beta$ -def by simp

```

```

    subgoal using  $\omega^3$  unfolding  $\omega^3$ -def by simp
    subgoal using  $\omega^3$  unfolding  $\omega^3$ -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  $\omega^3$  unfolding  $\omega^3$ -def by simp
    subgoal using  $\omega^3$  unfolding  $\omega^3$ -def
    by (metis Opt.reach.Step  $\omega\omega$ (2) fst-conv r(1) snd-conv)
    subgoal by fact
    subgoal using  $\omega^3$  unfolding  $\omega^3$ -def
    by (metis Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
    subgoal using  $\omega^3$  unfolding  $\omega^3$ -def
    by (metis Simple-Transition-System.reach-validFromS-reach r(4)
snoc-eq-iff-butlast)
    subgoal using  $\omega\omega$  by auto
    subgoal using  $\omega\omega$  by auto
    subgoal using  $\omega\omega$ 
    using llist-all-lappend-llist-of ltr1(3) by blast
    subgoal using  $\omega\omega$  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
 $\chi\chi$ :  $\chi\chi$  s1 ltr1 tr1 s1' s1'' ltr1' and
 $\chi^3$ :  $\chi^3$   $\Delta$  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 un $\omega$   $\Delta$  r ltr1 isi4 current]
unfolding ltrv1 ltrv2 by blast
define ltrv1' where ltrv1': ltrv1'  $\equiv$  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 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 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))
    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 unω 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  $\omega_4$  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 Simple-Transition-System.reach-validFromS-reach r(3)
snoc-eq-iff-butlast)
  subgoal using  $\omega_4$  unfolding  $\omega_4$ -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 by fact
  subgoal using  $\omega\omega$  by auto
  subgoal using  $\omega\omega$  by auto . .
next
case False note current = current False
obtain w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO where
 $\chi\chi$ :  $\chi\chi$  s2 ltr2 tr2 s2' s2'' ltr2' and
 $\chi_4'$ :  $\chi_4' \Delta$  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 un $\omega$   $\Delta$  r ltr2(1,2) isi3 ltr2(3)
current]
  unfolding ltrv1 ltrv2 by blast
  define ltrv1' where ltrv1': ltrv1'  $\equiv$  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'  $\equiv$  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  $\chi_4'$  unfolding  $\chi_4'$ -def by simp
  subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -def by simp
  subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -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  $\chi_4'$  unfolding  $\chi_4'$ -def by simp
  subgoal by fact
  subgoal using  $\chi_4'$  unfolding  $\chi_4'$ -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
}
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**  $ltr1 \neq []$   $ltr2 \neq []$  *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**  $ltr1 \neq []$   $ltr2 \neq []$   $\neg$  *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* ::  $('stateO, 'stateV)tuple34 \Rightarrow 'stateV$  **l***list* **where**

*ltrv1 tp* = *lconcat (lmap ( $\lambda(trv1, sv1'', trv2, sv2'', statAA, statOO)$ ). *l**list-of trv1*) (F tp))*

**definition** *firstHolds1* ::  $('stateO, 'stateV)tuple34 \Rightarrow nat$  **where**

*firstHolds1 tp* = *firstNC (lmap ( $\lambda(trv1, sv1'', trv2, sv2'', statAA, statOO)$ ). *trv1*) (F tp))*

**definition** *ltrv2* ::  $('stateO, 'stateV)tuple34 \Rightarrow 'stateV$  **l***list* **where**

*ltrv2 tp* = *lconcat (lmap ( $\lambda(trv1, sv1'', trv2, sv2'', statAA, statOO)$ ). *l**list-of trv2*) (F tp))*

**definition** *firstHolds2* ::  $('stateO, 'stateV)tuple34 \Rightarrow nat$  **where**

*firstHolds2 tp* = *firstNC (lmap ( $\lambda(trv1, sv1'', trv2, sv2'', statAA, statOO)$ ). *trv2*) (F tp))*

**lemma** *ltrv1-ne-imp*:

**assumes** *ltrv1 tp*  $\neq []$

**shows**  $\exists trv1 sv1'' trv2 sv2'' statAA statOO. (trv1, sv1'', trv2, sv2'', statAA, statOO) \in$  *l**set (F tp)*  $\wedge$

$trv1 \neq []$

**using** *assms* **unfolding** *ltrv1-def* **unfolding** *lconcat-eq-LNil-iff* **by force**

**lemma** *ltrv2-ne-imp*:

**assumes** *ltrv2 tp*  $\neq []$

**shows**  $\exists trv1 sv1'' trv2 sv2'' statAA statOO. (trv1, sv1'', trv2, sv2'', statAA, statOO)$

$\in \text{lset } (F \text{ tp}) \wedge$   
 $\text{trv2} \neq []$   
**using** *assms* **unfolding** *ltrv2-def* **unfolding** *lconcat-eq-LNil-iff* **by** *force*

**lemma** *ltrv1-LNil[simp]*:  
 $\text{ltr1} = [] \vee \text{ltr2} = [] \implies \text{ltrv1 } (w1, w2, s1, \text{ltr1}, s2, \text{ltr2}, \text{statA}, sv1, sv2, \text{statO}) = []$   
**unfolding** *ltrv1-def F-LNil* **by** *simp*  
**lemma** *ltrv2-LNil[simp]*:  
 $\text{ltr1} = [] \vee \text{ltr2} = [] \implies \text{ltrv2 } (w1, w2, s1, \text{ltr1}, s2, \text{ltr2}, \text{statA}, sv1, sv2, \text{statO}) = []$   
**unfolding** *ltrv2-def F-LNil* **by** *simp*

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

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

**lemma** *H-T-not-lnever*:  
**assumes** *unw: unwindCond Δ*  
**and**  $\Delta: \Delta \infty w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$   
**and**  $r: \text{reachO } s1 \ \text{reachO } s2 \ \text{reachV } sv1 \ \text{reachV } sv2$   
**and**  $\text{stat}: \text{statA} = \text{Diff} \longrightarrow \text{statO} = \text{Diff}$   
**and**  $\text{ltr1}: \text{Opt.lvalidFromS } s1 \ \text{ltr1} \ \text{Opt.lcompletedFrom } s1 \ \text{ltr1}$   
**and**  $\text{ltr2}: \text{Opt.lvalidFromS } s2 \ \text{ltr2} \ \text{Opt.lcompletedFrom } s2 \ \text{ltr2}$   
**and**  $l: \neg \text{lnever isIntO ltr1} \ \text{Opt.lA ltr1} = \text{Opt.lA ltr2}$   
**shows**  $\exists w1' \ w2' \ tr1 \ s1' \ s1'' \ \text{ltr1}' \ tr2 \ s2' \ s2'' \ \text{ltr2}' \ \text{trv1} \ sv1'' \ \text{trv2} \ sv2'' \ \text{statAA} \ \text{statOO}$ .  
 $\varphi\varphi \ s1 \ \text{ltr1} \ s2 \ \text{ltr2} \ tr1 \ s1' \ s1'' \ \text{ltr1}' \ tr2 \ s2' \ s2'' \ \text{ltr2}' \wedge$   
 $\varphi' \ \Delta \ w1 \ w2 \ w1' \ w2' \ \text{statA} \ s1 \ tr1 \ s1' \ s1'' \ s2 \ tr2 \ s2' \ s2'' \ \text{statAA} \ \text{statO} \ sv1 \ \text{trv1} \ sv1'' \ sv2 \ \text{trv2} \ sv2'' \ \text{statOO} \wedge$   
 $H-T \ (w1, w2, s1, \text{ltr1}, s2, \text{ltr2}, \text{statA}, sv1, sv2, \text{statO}) =$   
 $((\text{trv1}, sv1'', \text{trv2}, sv2'', \text{statAA}, \text{statOO}),$   
 $(w1', w2', s1'', s1'' \ \$ \ \text{ltr1}', s2'', s2'' \ \$ \ \text{ltr2}', \text{statAA}, sv1'', sv2'', \text{statOO}))$   
**proof** –  
**obtain**  $tr1 \ s1' \ s1'' \ \text{ltr1}' \ tr2 \ s2' \ s2'' \ \text{ltr2}'$   
**where**  $\varphi\varphi: \varphi\varphi \ s1 \ \text{ltr1} \ s2 \ \text{ltr2} \ tr1 \ s1' \ s1'' \ \text{ltr1}' \ tr2 \ s2' \ s2'' \ \text{ltr2}'$

```

using isIntO- $\varphi\varphi$ [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')  $\Rightarrow$ 
 $\varphi\varphi$  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')  $\Rightarrow$ 
 $\varphi\varphi$  s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'
using  $\varphi\varphi$  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  $\varphi\varphi$ :  $\varphi\varphi$  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  $\varphi'$ :  $\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO$ 
sv1 trv1 sv1'' sv2 trv2 sv2'' statOO
using unwindCond-ex- $\varphi'$ [OF unW  $\Delta$  r stat, of tr1 s1' s1'' tr2 s2' s2'']
using  $\varphi\varphi$  unfolding  $\varphi\varphi$ -def by auto

define tp' where
tp' = (SOME k'. case k' of (w1',w2',trv1,sv1'',trv2,sv2'',statAA,statOO)  $\Rightarrow$ 
 $\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO$  sv1
trv1 sv1'' sv2 trv2 sv2'' statOO)

have 1: case tp' of (w1',w2',trv1,sv1'',trv2,sv2'',statAA,statOO)  $\Rightarrow$ 
 $\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO$  sv1
trv1 sv1'' sv2 trv2 sv2'' statOO
using  $\varphi'$  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  $\varphi'$ :  $\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO$ 
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  $\varphi\varphi$  .
  subgoal using  $\varphi'$  .
  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*  $\Delta$

**and**  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$

**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** *l*:  $\neg$  *lnever isIntO* *ltr1* *Opt.lA* *ltr1* = *Opt.lA* *ltr2*

**shows**  $\exists w1' w2' tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statAA statOO$ .

$\varphi\varphi s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2' \wedge$

$\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO sv1 trv1 sv1'' sv2 trv2 sv2'' statOO \wedge$

*ltrv1* (*w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*) =

*lappend* (*l**list-of* *trv1*) (*ltrv1* (*w1'*, *w2'*, *s1''*, *s1''* \$ *ltr1'*, *s2''*, *s2''* \$ *ltr2'*, *statAA*, *sv1''*, *sv2''*, *statOO*))

$\wedge$

*ltrv2* (*w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*) =

*lappend* (*l**list-of* *trv2*) (*ltrv2* (*w1'*, *w2'*, *s1''*, *s1''* \$ *ltr1'*, *s2''*, *s2''* \$ *ltr2'*, *statAA*, *sv1''*, *sv2''*, *statOO*))

**proof** –

**have** *ltr1NE*: *ltr1*  $\neq$  [] **using** *l(1)* **by** *auto*

**hence** *ltr2NE*: *ltr2*  $\neq$  [] **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*  $\Delta$   
**and**  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**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** *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**  $\exists w1\ w2\ s1\ ltr1\ s2\ ltr2\ statA\ sv2\ statO.$   
 $ltrr1 = ltrv1\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) \wedge$   
 $n1 = w1 \wedge$   
 $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$   
 $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$   
 $(statA = Diff \longrightarrow statO = Diff) \wedge$   
 $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge$   
 $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge$   
 $Opt.lA\ ltr1 = Opt.lA\ ltr2$   
**hence** *Van.llvalidFromS* *n1* *sv1* *ltrr1*  
**proof**(*coinduct* rule: *Van.llvalidFromS.coinduct*[*of*  $\lambda n1\ sv1\ ltrr1.$   
 $\exists w1\ w2\ s1\ ltr1\ s2\ ltr2\ statA\ sv2\ statO.$   
 $ltrr1 = ltrv1\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) \wedge$   
 $n1 = w1 \wedge$   
 $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$   
 $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$   
 $(statA = Diff \longrightarrow statO = Diff) \wedge$   
 $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge$   
 $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge$   
 $Opt.lA\ ltr1 = Opt.lA\ ltr2$ ])  
**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**  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**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*

```

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 = ltrv1 (L, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2,
statO)
  unfolding ltrr1 ltrv1-lnever[OF current] by simp
  show ?thesis apply(rule Van.lvalidFromS-selectlvalidFromS)
  unfolding ltrr1 apply simp
  apply(rule lvalidFromS-ltrv1)
  using ln34 Δ lvalidFromS 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  $\varphi'$  unfolding  $\varphi'$ -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  $\varphi'$  unfolding  $\varphi'$ -def
      by (metis Van.validS-append1 Van.validFromS-def append-is-Nil-conv
hd-append2)
    subgoal by fact
    subgoal using  $\varphi'$  unfolding  $\varphi'$ -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  $\varphi'$  unfolding  $\varphi'$ -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 using r(3)  $\varphi'$  unfolding  $\varphi'$ -def
      by (metis Van.reach-validFromS-reach append-is-Nil-conv last-snoc
trv1)
    subgoal using r(4)  $\varphi'$  unfolding  $\varphi'$ -def
      by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
    subgoal using  $\varphi'$  unfolding  $\varphi'$ -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  $\varphi'$  unfolding  $\varphi'$ -def by simp
  show ?thesis
  apply(rule Van.lvalidFromS-selectDelay)
  apply(rule exI[of - w1']) apply(rule exI[of - w1])
  apply(rule exI[of - sv1''']) apply(rule exI[of - lrr1'])
  apply(intro conjI)
  subgoal unfolding n1 .. subgoal by simp
  subgoal unfolding lrr1 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'''] $ lrr1')
  apply(rule exI[of - s2''']) apply(rule exI[of - s2'''] $ lrr2')
  apply(rule exI[of - statAA]) apply(rule exI[of - sv2''']) apply(rule exI[of
- statOO])
  apply(intro conjI)
  subgoal unfolding lrr1' ..
  subgoal ..
  subgoal using  $\varphi'$  unfolding  $\varphi'$ -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)  $\varphi'$  unfolding  $\varphi'$ -def
  by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
  subgoal using  $\varphi'$  unfolding  $\varphi'$ -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.lvalidFromS-imp-lvalidFromS)
using assms by blast
qed

```

```

lemma lvalidFromS-ltrv2:
  assumes unw: unwindCond  $\Delta$ 
  and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
  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**  $ltr14$ :  $Opt.lA\ ltr1 = Opt.lA\ ltr2$   
**shows**  $Van.lvalidFromS\ sv2\ (ltrv2\ (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO))$   
**proof** –  
    **{fix**  $n2\ sv2\ ltrr2$   
    **assume**  $\exists w1\ w2\ s1\ ltr1\ s2\ ltr2\ statA\ sv1\ statO.$   
     $ltrr2 = ltrv2\ (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) \wedge$   
     $n2 = w2 \wedge$   
     $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$   
     $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$   
     $(statA = Diff \longrightarrow statO = Diff) \wedge$   
     $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge$   
     $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge$   
     $Opt.lA\ ltr1 = Opt.lA\ ltr2$   
    **hence**  $Van.llvalidFromS\ n2\ sv2\ ltrr2$   
    **proof**(*coinduct rule: Van.llvalidFromS.coinduct[of  $\lambda n2\ sv2\ ltrr2.$*   
     $\exists w1\ w2\ s1\ ltr1\ s2\ ltr2\ statA\ sv1\ statO.$   
     $ltrr2 = ltrv2\ (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) \wedge$   
     $n2 = w2 \wedge$   
     $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$   
     $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$   
     $(statA = Diff \longrightarrow statO = Diff) \wedge$   
     $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge$   
     $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge$   
     $Opt.lA\ ltr1 = Opt.lA\ ltr2]$ )  
    **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**  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
    **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**  $A34$ :  $Opt.lA\ ltr1 = Opt.lA\ ltr2$   
    **by** *auto*  
  
    **have**  $current$ :  $ltr1 \neq []\ ltr2 \neq []$   
    **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.lvalidFromS-selectlvalidFromS)
unfolding ltrr2 apply simp
apply(rule lvalidFromS-ltrv2)
using ln34  $\Delta$  lvalidFromS 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:  $\neg$  lnever isIntO ltr2
by (metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1(2) A34
ltr2(2))

have ltr2  $\neq$  [[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  $\neg$  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:  $\neg$  finalV sv1  $\wedge$   $\neg$  finalV sv2
using  $\Delta$  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  $\varphi\varphi$ :  $\varphi\varphi$  s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'
and  $\varphi'$ :  $\varphi' \Delta$  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  $\Delta$  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  $\neq$  []  $\vee$  (trv2 = []  $\wedge$  w2' < w2)
using  $\varphi'$  unfolding  $\varphi'$ -def ltrr2 by simp

show ?thesis using ne proof(elim disjE conjE)
assume trv2: trv2  $\neq$  []
show ?thesis
apply(rule Van.lvalidFromS-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  $\varphi'$  unfolding  $\varphi'$ -def
  by (metis Van.validS-append1 Van.validFromS-def append-is-Nil-conv
hd-append2)
subgoal by fact
subgoal using  $\varphi'$  unfolding  $\varphi'$ -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  $\uparrow$ ]) apply(rule exI[of - w2  $\uparrow$ ])
apply(rule exI[of - s1  $\uparrow$ ]) apply(rule exI[of - s1'' $ ltr1  $\uparrow$ ])
apply(rule exI[of - s2  $\uparrow$ ]) apply(rule exI[of - s2'' $ ltr2  $\uparrow$ ])
apply(rule exI[of - statAA]) apply(rule exI[of - sv1  $\uparrow$ ]) apply(rule exI[of
- statOO])
apply(intro conjI)
subgoal unfolding ltrr2' ..
subgoal ..
subgoal using  $\varphi'$  unfolding  $\varphi'$ -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 using r(3)  $\varphi'$  unfolding  $\varphi'$ -def
  by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
subgoal using r(4)  $\varphi'$  unfolding  $\varphi'$ -def
  by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
subgoal using  $\varphi'$  unfolding  $\varphi'$ -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 trv2[simp]: trv2 = [] and MM': w2' < w2
hence sv2''[simp]: sv2'' = sv2 using  $\varphi'$  unfolding  $\varphi'$ -def by simp
show ?thesis
apply(rule Van.lvalidFromS-selectDelay)
apply(rule exI[of - w2  $\uparrow$ ]) apply(rule exI[of - w2])
apply(rule exI[of - sv2  $\uparrow$ ]) apply(rule exI[of - ltrr2  $\uparrow$ ])
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  $\uparrow$ ]) apply(rule exI[of - w2  $\uparrow$ ])
apply(rule exI[of - s1  $\uparrow$ ]) apply(rule exI[of - s1'' $ ltr1  $\uparrow$ ])
apply(rule exI[of - s2  $\uparrow$ ]) apply(rule exI[of - s2'' $ ltr2  $\uparrow$ ])
apply(rule exI[of - statAA]) apply(rule exI[of - sv1  $\uparrow$ ]) apply(rule exI[of

```

```

- statOO])
  apply(intro conjI)
  subgoal unfolding ltrr2' ..
  subgoal ..
  subgoal using  $\varphi'$  unfolding  $\varphi'$ -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 using r(3)  $\varphi'$  unfolding  $\varphi'$ -def
    by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
  subgoal unfolding sv2'' by fact
  subgoal using  $\varphi'$  unfolding  $\varphi'$ -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.lvalidFromS-imp-lvalidFromS)
using assms by blast
qed

```

```

lemma lcompletedFrom-ltrv1:
  assumes unw: unwindCond  $\Delta$ 
  and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
  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.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  $\neq$  []  $\wedge$  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  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
and  $r$ :  $reachO s1 reachO s2 reachV sv1 reachV sv2$ 
and  $stat$ :  $statA = Diff \longrightarrow statO = Diff$ 
and  $ltr1$ :  $Opt.lvalidFromS s1 ltr1 lcompletedFromO s1 ltr1$ 
and  $ltr2$ :  $Opt.lvalidFromS s2 ltr2 lcompletedFromO s2 ltr2$ 
and  $A34$ :  $Opt.lA ltr1 = Opt.lA ltr2$ 
by auto

have  $current$ :  $ltr1 \neq []$   $ltr2 \neq []$ 
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  $lcompletedFrom-lltrv1[OF unW \Delta r ltr1 ln3 ltr2 ln4, of L]$ 
  using  $lfin[unfolded ltrr1]$ 
  unfolding  $Van.lcompletedFrom-def ltrr1[symmetric]$ 
  using  $l\text{list-of-list-of}$  by fastforce
next
  case False note  $ln3 = False$ 
  hence  $ln4$ :  $\neg lnever isIntO ltr2$ 
  by (metis  $Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1(2) A34$ 
 $ltr2(2)$ )

  have  $ltr1 \neq [s1]$  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  $\neg finalO s1$ 
  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  $nf12$ :  $\neg finalV sv1 \wedge \neg finalV sv2$ 
  using  $\Delta r(1) r(2) r(3) r(4) unW unwindCond-def$  by force

  obtain  $w1' w2' tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statAA$ 
 $statOO$ 
  where  $\varphi\varphi$ :  $\varphi\varphi s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'$ 
  and  $\varphi'$ :  $\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA$ 
 $statO sv1 trv1 sv1'' sv2 trv2 sv2'' statOO$ 
  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 (llist-of ltrr1') < length (llist-of ltrr1) ∨
length (llist-of ltrr1') = length (llist-of ltrr1) ∧ w1' < w1
using ne lfin lfin' by (simp add: llist-of-lappend ltrr1)

have 0: llist-of ltrr1' ≠ [] ∧ finalV (last (llist-of ltrr1'))
using len proof(elim disjE conjE)
assume len: length (llist-of ltrr1') < length (llist-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 (llist-of ltrr1') = length (llist-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)  $\varphi\varphi$  unfolding  $\varphi\varphi$ -def
    by (metis Opt.reach-validFromS-reach append-is-Nil-conv last-snoc
not-Cons-self2)
  subgoal using r(3)  $\varphi'$  unfolding  $\varphi'$ -def
  by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
  subgoal using r(4)  $\varphi'$  unfolding  $\varphi'$ -def
  by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
  subgoal using  $\varphi'$  unfolding  $\varphi'$ -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
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  $\Delta$ 
and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
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.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  $\neq [] \wedge$  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  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
and r: reachO s1 reachO s2 reachV sv1 reachV sv2
and stat: statA = Diff  $\longrightarrow$  statO = Diff
and ltr1: Opt.lvalidFromS s1 ltr1 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 = ltrv2 (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 lrr2 lrr2' ..
have ne: trv2 ≠ [] ∨ (trv2 = [] ∧ w2' < w2)
using  $\varphi'$  unfolding  $\varphi'$ -def lrr2 by simp

have lfin': lfinite lrr2'
using lfin ne unfolding lrr2 by simp
have len: length (list-of lrr2') < length (list-of lrr2) ∨
      length (list-of lrr2') = length (list-of lrr2) ∧ w2' < w2
using ne lfin lfin' by (simp add: list-of-lappend lrr2)

have 0: list-of lrr2' ≠ [] ∧ finalV (last (list-of lrr2'))
using len proof(elim disjE conjE)
  assume len: length (list-of lrr2') < length (list-of lrr2)
  show ?thesis
  apply(rule less(1)[OF - lrr2'])
    subgoal by fact subgoal by fact
    subgoal using  $\varphi'$  unfolding  $\varphi'$ -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)  $\varphi'$  unfolding  $\varphi'$ -def
      by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
    subgoal using r(4)  $\varphi'$  unfolding  $\varphi'$ -def
      by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
    subgoal using  $\varphi'$  unfolding  $\varphi'$ -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 len: length (list-of lrr2') = length (list-of lrr2) w2' < w2
  show ?thesis
  apply(rule less(2)[OF - - lrr2'])
    subgoal by fact subgoal unfolding len ..
    subgoal by fact
    subgoal using  $\varphi'$  unfolding  $\varphi'$ -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)  $\varphi'$  unfolding  $\varphi'$ -def
      by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
    subgoal using r(4)  $\varphi'$  unfolding  $\varphi'$ -def

```

```

    by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
    subgoal using  $\varphi'$  unfolding  $\varphi'$ -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
  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  $\Delta$ 
  and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
  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.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  $\exists w1 w2 s1 s2 ltr2 statA sv1 sv2 statO$ .
    nL = w1  $\wedge$  nR = w1  $\wedge$ 
    ltrr1 = ltrv1 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\wedge$ 
     $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO \wedge$ 
    reachO s1  $\wedge$  reachO s2  $\wedge$  reachV sv1  $\wedge$  reachV sv2  $\wedge$ 
    (statA = Diff  $\longrightarrow$  statO = Diff)  $\wedge$ 
    Opt.lvalidFromS s1 ltr1  $\wedge$  Opt.lcompletedFrom s1 ltr1  $\wedge$ 
    Opt.lvalidFromS s2 ltr2  $\wedge$  Opt.lcompletedFrom s2 ltr2  $\wedge$ 
    Opt.lA ltr1 = Opt.lA ltr2
  hence TwoFuncPred.sameFM isSecV isSecO getSecV getSecO nL nR ltrr1 ltr1
  proof(coinduct rule: TwoFuncPred.sameFM.coinduct[of  $\lambda nL nR ltrr1 ltr1$ .
     $\exists w1 w2 s1 s2 ltr2 statA sv1 sv2 statO$ .
    nL = w1  $\wedge$  nR = w1  $\wedge$ 
    ltrr1 = ltrv1 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\wedge$ 
     $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO \wedge$ 
    reachO s1  $\wedge$  reachO s2  $\wedge$  reachV sv1  $\wedge$  reachV sv2  $\wedge$ 
    (statA = Diff  $\longrightarrow$  statO = Diff)  $\wedge$ 
    Opt.lvalidFromS s1 ltr1  $\wedge$  Opt.lcompletedFrom s1 ltr1  $\wedge$ 


```

```

    Opt.lvalidFromS s2 ltr2 ∧ Opt.lcompletedFrom s2 ltr2 ∧
    Opt.lA ltr1 = Opt.lA ltr2])
  case (2 nL nR ltr1 ltr1)
  then obtain w1 w2 s1 s2 ltr2 statA sv1 sv2 statO
  where nL: nL = w1 and nR: nR = w1
  and ltr1: ltr1 = 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 ltr1: ltr1 = lltrv1 (L, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2,
  statO)
  unfolding ltr1 lltrv1-lnever[OF current] by simp

  have cltrv1: 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 ltr1 apply simp
  using lS-lltrv1-ltr1[OF unW Δ r ltr1 ln3 ltr2 ln4, of L]
  unfolding Van.lS[OF cltrv1] 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:  $\neg$  finalV sv1  $\wedge$   $\neg$  finalV sv2
using  $\Delta$  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  $\varphi\varphi$ :  $\varphi\varphi$  s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'
and  $\varphi'$ :  $\varphi' \Delta$  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  $\Delta$  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  $\neq$  []  $\vee$  w1' < w1
using  $\varphi'$  unfolding  $\varphi'$ -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  $\varphi\varphi$  unfolding  $\varphi\varphi$ -def by simp
subgoal by fact
subgoal by simp
subgoal using  $\varphi'$  unfolding  $\varphi'$ -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  $\varphi'$  unfolding  $\varphi'$ -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)  $\varphi'$  unfolding  $\varphi'$ -def
        by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
      subgoal using r(4)  $\varphi'$  unfolding  $\varphi'$ -def
        by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
      subgoal using  $\varphi'$  unfolding  $\varphi'$ -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 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  $\Delta$ 
and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
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.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  $\exists w1 w2 s1 s2 ltr1 statA sv1 sv2 statO.$ 
  nL = w2  $\wedge$  nR = w2  $\wedge$ 
  ltrr2 = ltrv2 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\wedge$ 
   $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO \wedge$ 
  reachO s1  $\wedge$  reachO s2  $\wedge$  reachV sv1  $\wedge$  reachV sv2  $\wedge$ 
  (statA = Diff  $\longrightarrow$  statO = Diff)  $\wedge$ 
  Opt.lvalidFromS s1 ltr1  $\wedge$  Opt.lcompletedFrom s1 ltr1  $\wedge$ 
  Opt.lvalidFromS s2 ltr2  $\wedge$  Opt.lcompletedFrom s2 ltr2  $\wedge$ 
  Opt.lA ltr1 = Opt.lA ltr2
hence TwoFuncPred.sameFM isSecV isSecO getSecV getSecO nL nR ltrr2 ltr2
proof(coinduct rule: TwoFuncPred.sameFM.coinduct[of  $\lambda nL nR ltrr2 ltr2.$ 
   $\exists w1 w2 s1 s2 ltr1 statA sv1 sv2 statO.$ 
  nL = w2  $\wedge$  nR = w2  $\wedge$ 
  ltrr2 = ltrv2 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\wedge$ 
   $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO \wedge$ 

```

```

    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 cltrv2: 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 cltrv2] 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**  $\Delta$  *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**  $\varphi\varphi$ :  $\varphi\varphi$  *s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'*  
**and**  $\varphi'$ :  $\varphi' \Delta$  *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**  $\varphi'$  **unfolding**  $\varphi'$ -*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]*)  
- *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**  $\varphi\varphi$  **unfolding**  $\varphi\varphi$ -*def* **by** *simp*  
**subgoal by fact**  
**subgoal by simp**  
**subgoal using**  $\varphi'$  **unfolding**  $\varphi'$ -*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**  $\varphi'$  **unfolding**  $\varphi'$ -*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)  $\varphi'$  unfolding  $\varphi'$ -def
        by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
      subgoal using r(4)  $\varphi'$  unfolding  $\varphi'$ -def
        by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
      subgoal using  $\varphi'$  unfolding  $\varphi'$ -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 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  $\Delta$ 
and  $\Delta$ :  $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO$ 
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  $\exists w1 w2 s1 ltr1 s2 ltr2 statA sv1 sv2 statO.$ 
    nL = w1  $\wedge$  nR = w2  $\wedge$ 
    ltrr1 = ltrv1 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\wedge$ 
    ltrr2 = ltrv2 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\wedge$ 
     $\Delta \infty w1 w2 s1 s2 statA sv1 sv2 statO \wedge$ 
    reachO s1  $\wedge$  reachO s2  $\wedge$  reachV sv1  $\wedge$  reachV sv2  $\wedge$ 
    (statA = Diff  $\longrightarrow$  statO = Diff)  $\wedge$ 

```

$Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge$   
 $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge$   
 $Opt.lA\ ltr1 = Opt.lA\ ltr2$

**hence**  $TwoFuncPred.sameFM\ isIntV\ isIntV\ getActV\ getActV\ nL\ nR\ ltrr1\ ltrr2$

**proof**(coinduct rule:  $TwoFuncPred.sameFM.coinduct[of\ \lambda nL\ nR\ ltrr1\ ltrr2.$

$\exists\ w1\ w2\ s1\ ltr1\ s2\ ltr2\ statA\ sv1\ sv2\ statO.$   
 $nL = w1 \wedge nR = w2 \wedge$   
 $ltrr1 = ltrv1\ (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) \wedge$   
 $ltrr2 = ltrv2\ (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) \wedge$   
 $\Delta \infty\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$   
 $reachO\ s1 \wedge reachO\ s2 \wedge reachV\ sv1 \wedge reachV\ sv2 \wedge$   
 $(statA = Diff \longrightarrow statO = Diff) \wedge$   
 $Opt.lvalidFromS\ s1\ ltr1 \wedge Opt.lcompletedFrom\ s1\ ltr1 \wedge$   
 $Opt.lvalidFromS\ s2\ ltr2 \wedge Opt.lcompletedFrom\ s2\ ltr2 \wedge$   
 $Opt.lA\ ltr1 = Opt.lA\ ltr2])$

**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**  $\Delta: \Delta \infty\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$

**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**  $A34: Opt.lA\ ltr1 = Opt.lA\ ltr2$

**by auto**

**have current:**  $ltr1 \neq [] \ ltr2 \neq []$

**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**  $cllrv1: Van.lcompletedFrom\ sv1\ (lltrv1\ (L, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO))$

**using**  $lcompletedFrom-lltrv1[OF\ unW\ \Delta\ r\ ltr1\ ln3\ ltr2\ ln4]$  .

**have**  $cllrv2: Van.lcompletedFrom\ sv2\ (lltrv2\ (L, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO))$

**using**  $lcompletedFrom-lltrv2[OF\ unW\ \Delta\ r\ ltr1\ ln3\ ltr2\ ln4]$  .

```

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

```



**lemma** *lO-ltrv1-ltrv2*:

**assumes** *unw*: *unwindCond*  $\Delta$

**and**  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$

**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*

**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**  $\exists$  *ltrr1* *ltrr2* *w1* *w2* *s1* *s2* *statA* *sv1* *sv2* *statO*.

*ltrr1* = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)  $\wedge$

*ltrr2* = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)  $\wedge$

$\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$

*reachO* *s1*  $\wedge$  *reachO* *s2*  $\wedge$  *reachV* *sv1*  $\wedge$  *reachV* *sv2*  $\wedge$

(*statA* = *Diff*  $\longrightarrow$  *statO* = *Diff*)  $\wedge$

*Opt.lvalidFromS* *s1* *ltr1*  $\wedge$  *Opt.lcompletedFrom* *s1* *ltr1*  $\wedge$

*Opt.lvalidFromS* *s2* *ltr2*  $\wedge$  *Opt.lcompletedFrom* *s2* *ltr2*  $\wedge$

*Opt.lA* *ltr1* = *Opt.lA* *ltr2*  $\wedge$

*Van.lO* *ltrr1* = *Van.lO* *ltrr2*  $\wedge$

*statO* = *Eq*

**hence** *TwoFuncPred.sameFM* *isIntO* *isIntO* *getObsO* *getObsO* *nL* *nR* *ltr1* *ltr2*

**proof**(*coinduct* rule: *TwoFuncPred.sameFM.coinduct*[*of*  $\lambda nL\ nR\ ltr1\ ltr2$  .

$\exists$  *ltrr1* *ltrr2* *w1* *w2* *s1* *s2* *statA* *sv1* *sv2* *statO* .

*ltrr1* = *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)  $\wedge$

*ltrr2* = *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)  $\wedge$

$\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \wedge$

*reachO* *s1*  $\wedge$  *reachO* *s2*  $\wedge$  *reachV* *sv1*  $\wedge$  *reachV* *sv2*  $\wedge$

(*statA* = *Diff*  $\longrightarrow$  *statO* = *Diff*)  $\wedge$

*Opt.lvalidFromS* *s1* *ltr1*  $\wedge$  *Opt.lcompletedFrom* *s1* *ltr1*  $\wedge$

*Opt.lvalidFromS* *s2* *ltr2*  $\wedge$  *Opt.lcompletedFrom* *s2* *ltr2*  $\wedge$

*Opt.lA* *ltr1* = *Opt.lA* *ltr2*  $\wedge$

*Van.lO* *ltrr1* = *Van.lO* *ltrr2*  $\wedge$

*statO* = *Eq*])

**case** ( $\exists nL\ nR\ ltr1\ ltr2$ )

**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**  $\Delta$ :  $\Delta \infty w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$

**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
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  $\neq$   $[[\ ]]$  ltr2  $\neq$   $[[\ ]]$ 
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  $\Delta$  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  $\Delta$  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:  $\neg$  lnever isIntO ltr2
by (metis Opt.lA lfiltermap-LNil-never lfiltermap-lmap-lfilter ltr1(2) A34
ltr2(2))

have ltr1  $\neq$   $[[s1]]$  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-LNil llist-eq-cong ltr1(1) ltr1(2) nle-le not-less-zero)
hence  $\neg$  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*:  $\neg \text{finalV } sv1 \wedge \neg \text{finalV } sv2$   
**using**  $\Delta$  *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**  $\varphi\varphi$ :  $\varphi\varphi$  *s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'*  
**and**  $\varphi'$ :  $\varphi' \Delta$  *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**  $\varphi'$  **unfolding**  $\varphi'$ -*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**  $\varphi'$  **unfolding**  $\varphi'$ -*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**  $\varphi\varphi$   
**unfolding**  $\varphi\varphi$ -*def* **by** *auto*

**have** *sst*: *statOO = Diff*  $\longleftrightarrow$  *Van.O (trv1 ## sv1'') ≠ Van.O (trv2 ##*  
*sv2'')*  
*statA = Eq*  $\implies$  *statAA = Diff*  $\longleftrightarrow$  *Opt.O ((tr1 ## s1') ## s1'') ≠ Opt.O*  
*((tr2 ## s2') ## s2'')*  
*statO = Diff*  $\implies$  *statOO = Diff*  
*statAA = Diff*  $\implies$  *statOO = Diff*  
**using**  $\varphi'$  *stO* **unfolding**  $\varphi'$ -*def* **by** *auto*

**have** *Atrv12'*: *Van.A (trv1 ## sv1'') = Van.A (trv2 ## sv2'')*  
**using**  $\varphi'$  **unfolding**  $\varphi'$ -*def* **by** *auto*

**have**  $\Delta'$ :  $\Delta \infty$  *w1' w2' s1'' s2'' statAA sv1'' sv2'' statOO*  
**using**  $\varphi'$  **unfolding**  $\varphi'$ -*def* **by** *auto*

```

have ultrv1: 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 ultrv2: 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  $\varphi\varphi$  unfolding  $\varphi\varphi$ -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  $\varphi\varphi$  unfolding  $\varphi\varphi$ -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  $\varphi'$  unfolding  $\varphi'$ -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)  $\varphi'$  unfolding  $\varphi'$ -def
      by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
    subgoal using r(4)  $\varphi'$  unfolding  $\varphi'$ -def

```

```

    by (metis Van.reach-validFromS-reach snoc-eq-iff-butlast)
    subgoal using  $\varphi'$  unfolding  $\varphi'$ -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 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  $\Delta$  and unwind: unwindCond  $\Delta$ 
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  $\neq$  Opt.lO tr2
  obtain sv1 sv2 where
  isv12: istateV sv1 istateV sv2 and c12: corrState sv1 s1 corrState sv2 s2
  and  $\Delta$ :  $\Delta \infty \infty \infty$  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  $\Delta$  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  $\Delta$  ( $\infty$ ,  $\infty$ , s1, tr1, s2, tr2, Eq, sv1, sv2, Eq)])
  apply(rule exI[of - sv2])
  apply(rule exI[of - ltrv2  $\Delta$  ( $\infty$ ,  $\infty$ , 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 ⇒ enat ⇒ enat ⇒ 'stateO ⇒ 'stateO ⇒ status ⇒ 'stateV ⇒ 'stateV ⇒ status ⇒ bool) ⇒*  
*(enat ⇒ enat ⇒ enat ⇒ 'stateO ⇒ 'stateO ⇒ status ⇒ 'stateV ⇒ 'stateV ⇒ status ⇒ bool)*  
 $\Rightarrow$  *bool*

**where**

*unwindIntoCond*  $\Delta$   $\Delta'$   $\equiv \forall w w1 w2 s1 s2 statA sv1 sv2 statO.$   
*reachO*  $s1 \wedge$  *reachO*  $s2 \wedge$  *reachV*  $sv1 \wedge$  *reachV*  $sv2 \wedge$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \longrightarrow$   
*(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*  $\Delta' v w1 w2 s1 s2 statA sv1 sv2 statO)$   
 $\vee$   
*react*  $\Delta' w1 w2 s1 s2 statA sv1 sv2 statO)$

**theorem** *distrib-unwind-lrsecure*:

**assumes**  $m: 0 < m$  **and**  $\text{next}: \bigwedge i. i < (m::\text{nat}) \implies \text{next } i \subseteq \{0..<m\}$

**and** *init*: *initCond* ( $\Delta s 0$ )

**and** *step*:  $\bigwedge i. i < m \implies$

*unwindIntoCond* ( $\Delta s i$ ) ( $\lambda w w1 w2 s1 s2 statA sv1 sv2 statO.$

$\exists j \in \text{next } i. \Delta s j w w1 w2 s1 s2 statA sv1 sv2 statO)$

**shows** *lrsecure*

**proof** –

**define**  $\Delta$  **where**  $D: \Delta \equiv \lambda w w1 w2 s1 s2 statA sv1 sv2 statO. \exists i < m. \Delta s i w w1 w2 s1 s2 statA sv1 sv2 statO$

**have**  $i: \text{initCond } \Delta$  **using** *init m unfolding initCond-def D by meson*

**have**  $c: \text{unwindCond } \Delta$  **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  $\lambda w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO.\ \exists j \in next\ i.$   
 $\Delta s\ j\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$ ])  
**subgoal** **unfolding** *le-fun-def D* **by** *simp* (*meson atLeastLessThan-iff next subsetD*)  
**subgoal** . . . .  
**subgoal** **apply**(*rule disjI2*)  
**apply**(*rule match-mono*[of  $\lambda w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO.\ \exists j \in next\ i.$   
 $\Delta s\ j\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$ ])  
**subgoal** **unfolding** *le-fun-def D* **by** *simp* (*meson atLeastLessThan-iff next subsetD*)  
**subgoal** . . . .  
**show** *?thesis* **using** *unwind-lrsecure[OF i c]* .  
**qed**

**lemma** *unwindIntoCond-simpleI*:

**assumes**

$\bigwedge w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO.$   
 $reachO\ s1 \implies reachO\ s2 \implies reachV\ sv1 \implies reachV\ sv2 \implies$   
 $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
 $\implies$   
 $(finalO\ s1 \longleftrightarrow finalO\ s2) \wedge (finalV\ sv1 \longleftrightarrow finalO\ s1) \wedge (finalV\ sv2 \longleftrightarrow finalO\ s2)$

**and**

$\bigwedge w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO.$   
 $reachO\ s1 \implies reachO\ s2 \implies reachV\ sv1 \implies reachV\ sv2 \implies$   
 $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $statA = Eq$   
 $\implies$   
 $isIntO\ s1 \longleftrightarrow isIntO\ s2$   
 $\bigwedge w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO.$   
 $reachO\ s1 \implies reachO\ s2 \implies reachV\ sv1 \implies reachV\ sv2 \implies$   
 $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
 $\implies$   
 $react\ \Delta'\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**shows** *unwindIntoCond*  $\Delta\ \Delta'$

using *assms unfolding unwindIntoCond-def by auto*

**lemma** *unwindIntoCond-simpleI2*:

**assumes**

$\bigwedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
 $\implies$   
 $(finalO s1 \longleftrightarrow finalO s2) \wedge (finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO$   
 $s2)$

**and**

$\bigwedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $statA = Eq$   
 $\implies$   
 $isIntO s1 \longleftrightarrow isIntO s2$

**and**

$\bigwedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
 $\implies$   
 $(\exists v < w. proact \Delta' v w1 w2 s1 s2 statA sv1 sv2 statO)$

**shows** *unwindIntoCond*  $\Delta \Delta'$

using *assms unfolding unwindIntoCond-def by auto*

**lemma** *unwindIntoCond-simpleIB*:

**assumes**

$\bigwedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
 $\implies$   
 $(finalO s1 \longleftrightarrow finalO s2) \wedge (finalV sv1 \longleftrightarrow finalO s1) \wedge (finalV sv2 \longleftrightarrow finalO$   
 $s2)$

**and**

$\bigwedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $statA = Eq$   
 $\implies$   
 $isIntO s1 \longleftrightarrow isIntO s2$

**and**

$\bigwedge w w1 w2 s1 s2 statA sv1 sv2 statO.$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $\Delta w w1 w2 s1 s2 statA sv1 sv2 statO$   
 $\implies$   
 $(\exists v < w. proact \Delta' v w1 w2 s1 s2 statA sv1 sv2 statO) \vee react \Delta' w1 w2 s1 s2$   
 $statA sv1 sv2 statO$

**shows** *unwindIntoCond*  $\Delta \Delta'$

using *assms* **unfolding** *unwindIntoCond-def* by *auto*

**definition** *oor* where

$oor \Delta \Delta_2 \equiv \lambda w w1 w2 s1 s2 statA sv1 sv2 statO.$

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \vee \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO$

**lemma** *oorI1*:

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor \Delta \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor-def* by *simp*

**lemma** *oorI2*:

$\Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor \Delta \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor-def* by *simp*

**definition** *oor3* where

$oor3 \Delta \Delta_2 \Delta_3 \equiv \lambda w w1 w2 s1 s2 statA sv1 sv2 statO.$

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \vee \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \vee$

$\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO$

**lemma** *oor3I1*:

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor3-def* by *simp*

**lemma** *oor3I2*:

$\Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor3-def* by *simp*

**lemma** *oor3I3*:

$\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor3-def* by *simp*

**definition** *oor4* where

$oor4 \Delta \Delta_2 \Delta_3 \Delta_4 \equiv \lambda w w1 w2 s1 s2 statA sv1 sv2 statO.$

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \vee \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \vee$

$\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO \vee \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$

**lemma** *oor4I1*:

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor4-def* by *simp*

**lemma** *oor4I2*:

$\Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor4-def* by *simp*

**lemma** *oor4I3*:

$\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor4-def* by *simp*

**lemma** *oor4I4*:

$\Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor4-def* by *simp*

**definition** *oor5* where

$oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 \equiv \lambda w w1 w2 s1 s2 statA sv1 sv2 statO.$

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \vee \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \vee$   
 $\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO \vee \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 $\vee$

$\Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$

**lemma** *oor5I1*:

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *oor5I2*:

$\Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *oor5I3*:

$\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *oor5I4*:

$\Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *oor5I5*:

$\Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$

**unfolding** *oor5-def* by *simp*

**lemma** *isIntO-match1*:  $isIntO\ s1 \implies match1\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**unfolding** *match1-def* **by** *auto*

**lemma** *isIntO-match2*:  $isIntO\ s2 \implies match2\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**unfolding** *match2-def* **by** *auto*

**lemma** *isIntO-match*:  
**assumes**  $\langle isIntO\ s1 \rangle$  **and**  $\langle isIntO\ s2 \rangle$   
**and**  $\langle match12\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \rangle$   
**shows**  $\langle react\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \rangle$   
**unfolding** *react-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*:  
 $match1-1\ \Delta\ w1\ w2\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $match1-1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO$   
**apply**(*rule match1-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match1-1-oorI2*:  
 $match1-1\ \Delta_2\ w1\ w2\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $match1-1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO$   
**apply**(*rule match1-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match1-oorI1*:  
 $match1\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $match1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**apply**(*rule match1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match1-oorI2*:  
 $match1\ \Delta_2\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $match1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**apply**(*rule match1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match2-1-oorI1*:  
 $match2-1\ \Delta\ w1\ w2\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO \implies$   
 $match2-1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO$   
**apply**(*rule match2-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match2-1-oorI2*:  
 $match2-1 \Delta_2 w1 w2 s1 s2 s2' statA sv1 sv2 statO \implies$   
 $match2-1 (oor \Delta \Delta_2) w1 w2 s1 s2 s2' statA sv1 sv2 statO$   
**apply**(rule *match2-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match2-oorI1*:  
 $match2 \Delta w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $match2 (oor \Delta \Delta_2) w1 w2 s1 s2 statA sv1 sv2 statO$   
**apply**(rule *match2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match2-oorI2*:  
 $match2 \Delta_2 w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $match2 (oor \Delta \Delta_2) w1 w2 s1 s2 statA sv1 sv2 statO$   
**apply**(rule *match2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-oorI1*:  
 $match12 \Delta w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $match12 (oor \Delta \Delta_2) w1 w2 s1 s2 statA sv1 sv2 statO$   
**apply**(rule *match12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-oorI2*:  
 $match12 \Delta_2 w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $match12 (oor \Delta \Delta_2) w1 w2 s1 s2 statA sv1 sv2 statO$   
**apply**(rule *match12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-1-oorI1*:  
 $match12-1 \Delta w1 w2 s1' s2' statA' sv1 sv2 statO \implies$   
 $match12-1 (oor \Delta \Delta_2) w1 w2 s1' s2' statA' sv1 sv2 statO$   
**apply**(rule *match12-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-1-oorI2*:  
 $match12-1 \Delta_2 w1 w2 s1' s2' statA' sv1 sv2 statO \implies$   
 $match12-1 (oor \Delta \Delta_2) w1 w2 s1' s2' statA' sv1 sv2 statO$   
**apply**(rule *match12-1-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-2-oorI1*:  
 $match12-2 \Delta w1 w2 s2 s2' statA' sv1 sv2 statO \implies$   
 $match12-2 (oor \Delta \Delta_2) w1 w2 s2 s2' statA' sv1 sv2 statO$   
**apply**(rule *match12-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-2-oorI2*:  
 $match12-2 \Delta_2 w1 w2 s2 s2' statA' sv1 sv2 statO \implies$   
 $match12-2 (oor \Delta \Delta_2) w1 w2 s2 s2' statA' sv1 sv2 statO$   
**apply**(rule *match12-2-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match12-12-oorI1*:  
 $match12-12 \Delta w1 w2 s1' s2' statA' sv1 sv2 statO \implies$

*match12-12 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta_2$  w1 w2 s1' s2' statA' sv1 sv2 statO  $\implies$*   
*match12-12 (oor  $\Delta$   $\Delta_2$ ) w1 w2 s1' s2' statA' sv1 sv2 statO*  
**apply**(rule *match12-12-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match-oorI1*:  
*react  $\Delta$  w1 w2 s1 s2 statA sv1 sv2 statO  $\implies$*   
*react (oor  $\Delta$   $\Delta_2$ ) w1 w2 s1 s2 statA sv1 sv2 statO*  
**apply**(rule *match-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *match-oorI2*:  
*react  $\Delta_2$  w1 w2 s1 s2 statA sv1 sv2 statO  $\implies$*   
*react (oor  $\Delta$   $\Delta_2$ ) w1 w2 s1 s2 statA sv1 sv2 statO*  
**apply**(rule *match-mono*) **unfolding** *le-fun-def oor-def* **by** *auto*

**lemma** *proact-oorI1*:  
*proact  $\Delta$  w w1 w2 s1 s2 statA sv1 sv2 statO  $\implies$*   
*proact (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta_2$  w w1 w2 s1 s2 statA sv1 sv2 statO  $\implies$*   
*proact (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta$  w w1 w2 s1 s2 statA sv1 sv2 statO  $\implies$*   
*move-1 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta_2$  w w1 w2 s1 s2 statA sv1 sv2 statO  $\implies$*   
*move-1 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta$  w w1 w2 s1 s2 statA sv1 sv2 statO  $\implies$*   
*move-2 (oor  $\Delta$   $\Delta_2$ ) 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  $\Delta_2$  w w1 w2 s1 s2 statA sv1 sv2 statO  $\implies$*

*move-2* (*oor*  $\Delta$   $\Delta_2$ ) *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*  $\Delta$  *w w1 w2 s1 s2 statA sv1 sv2 statO*  $\implies$   
*move-12* (*oor*  $\Delta$   $\Delta_2$ ) *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*  $\Delta_2$  *w w1 w2 s1 s2 statA sv1 sv2 statO*  $\implies$   
*move-12* (*oor*  $\Delta$   $\Delta_2$ ) *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*  $\Delta$  *w1 w2 s1 s1' s2 statA sv1 sv2 statO*  $\longleftrightarrow$   
 $\neg$  *finalV* *sv1*  $\wedge$   $\Delta \infty$  *w1 w2 s1' s2 statA* (*nextO* *sv1*) *sv2 statO*  
**unfolding** *match1-1-def validTrans-iff-next* **by** *simp*

**lemma** *match1-12-defD*: *match1-12*  $\Delta$  *w1 w2 s1 s1' s2 statA sv1 sv2 statO*  $\longleftrightarrow$   
 $\neg$  *finalV* *sv1*  $\wedge$   $\neg$  *finalV* *sv2*  $\wedge$   
 $\Delta \infty$  *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*  $\Delta$  *w1 w2 s1 s2 s2' statA sv1 sv2 statO*  $\longleftrightarrow$   
 $\neg$  *finalV* *sv2*  $\wedge$   $\Delta \infty$  *w1 w2 s1 s2' statA* *sv1* (*nextO* *sv2*) *statO*  
**unfolding** *match2-1-def validTrans-iff-next* **by** *simp*

**lemma** *match2-12-defD*: *match2-12*  $\Delta$  *w1 w2 s1 s2 s2' statA sv1 sv2 statO*  $\longleftrightarrow$   
 $\neg$  *finalV* *sv1*  $\wedge$   $\neg$  *finalV* *sv2*  $\wedge$   $\Delta \infty$  *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*  $\Delta$  *w1 w2 s1' s2' statA' sv1 sv2 statO*  $\longleftrightarrow$   
 $\neg$  *finalV* *sv1*  $\wedge$   $\Delta \infty$  *w1 w2 s1' s2' statA'* (*nextO* *sv1*) *sv2 statO*  
**unfolding** *match12-1-def validTrans-iff-next* **by** *simp*

**lemma** *match12-2-defD*:  $match12-2 \Delta w1 w2 s1' s2' statA' sv1 sv2 statO \longleftrightarrow$   
 $\neg finalV sv2 \wedge \Delta \infty w1 w2 s1' s2' statA' sv1 (nextO sv2) statO$   
**unfolding** *match12-2-def validTrans-iff-next* **by** *simp*

**lemma** *match12-12-defD*:  $match12-12 \Delta w1 w2 s1' s2' statA' sv1 sv2 statO \longleftrightarrow$

(*let*  $statO' = sstatO' statO sv1 sv2$  *in*  
 $\neg finalV sv1 \wedge \neg finalV sv2 \wedge$   
 $(statA' = Diff \longrightarrow statO' = Diff) \wedge$   
 $\Delta \infty w1 w2 s1' s2' statA' (nextO sv1) (nextO sv2) statO'$ )

**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*:  $move-1 \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \longleftrightarrow$   
 $\neg finalV sv1 \wedge \Delta w w1 w2 s1 s2 statA (nextO sv1) sv2 statO$   
**unfolding** *move-1-def validTrans-iff-next* **by** *simp*

**lemma** *move-2-defD*:  $move-2 \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \longleftrightarrow$   
 $\neg finalV sv2 \wedge \Delta w w1 w2 s1 s2 statA sv1 (nextO sv2) statO$   
**unfolding** *move-2-def validTrans-iff-next* **by** *simp*

**lemma** *move-12-defD*:  $move-12 \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \longleftrightarrow$   
(*let*  $statO' = sstatO' statO sv1 sv2$  *in*  
 $\neg finalV sv1 \wedge \neg finalV sv2 \wedge$   
 $\Delta w w1 w2 s1 s2 statA (nextO sv1) (nextO sv2) statO'$ )  
**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**

## 5 Relative Security Unwinding Incompleteness example

Demonstrating a counterexample which is secure but fails in the finitary unwinding

**theory** *Incomplete-fin*

```

imports Unwinding-fin
begin

```

```

no-notation bot ( $\perp$ )

```

```

abbreviation noninform ( $\perp$ ) where  $\perp \equiv \text{undefined}$ 

```

Demonstrating a counterexample which is secure but fails in the unwinding

```

datatype State = ss | ss'
type-synonym secret = State

```

```

fun transit::State  $\Rightarrow$  State  $\Rightarrow$  bool(infix  $\rightarrow I$  55) where
  transit s s' = (if (s = ss  $\wedge$  s' = ss') then True else False)

```

```

lemma transit-singlv::ss  $\rightarrow I$  ss' by auto

```

```

lemma transit-iff::s  $\rightarrow I$  s'  $\longleftrightarrow$  (s = ss  $\wedge$  s' = ss') by auto

```

```

definition final x  $\equiv \forall y. \neg (\rightarrow I) x y$ 

```

```

lemma final-sv[simp]:final ss' unfolding final-def transit-iff by auto

```

```

lemma final-iff: final s  $\longleftrightarrow$  s = ss' unfolding final-def transit-iff by (auto,metis
State.exhaust)

```

Vanilla-semantics system model

```

type-synonym stateV = State
fun validTransV where validTransV (s,s') = s  $\rightarrow I$  s'

```

No secrets or interaction

```

fun isIntV :: stateV  $\Rightarrow$  bool where isIntV s = False
fun getIntV::stateV  $\Rightarrow$  nat  $\times$  nat where getIntV s = ( $\perp$ , $\perp$ )

```

```

definition isSecV :: stateV  $\Rightarrow$  bool where isSecV s = False
fun getSecV :: stateV  $\Rightarrow$  secret where getSecV s = s

```

The optimization-enhanced system model

```

type-synonym stateO = State
fun validTransO where validTransO (s,s') = s  $\rightarrow I$  s'

```

No interaction, only isSec at starting state

```

fun isIntO :: stateO  $\Rightarrow$  bool where isIntO s = False
fun getIntO::stateO  $\Rightarrow$  nat  $\times$  nat where getIntO s = ( $\perp$ , $\perp$ )

```

**definition**  $isSecO :: stateO \Rightarrow bool$  **where**  $isSecO\ s = (if\ s = ss\ then\ True\ else\ False)$   
**fun**  $getSecO :: stateO \Rightarrow secret$  **where**  $getSecO\ s = s$

$corrState$

**fun**  $corrState :: stateV \Rightarrow stateO \Rightarrow bool$  **where**  
 $corrState\ cfgO\ cfgA = True$

**interpretation**  $Rel-Sec$

**where**  $validTransV = validTransV$  **and**  $istateV = \lambda s. s = ss$   
**and**  $finalV = final$   
**and**  $isSecV = isSecV$  **and**  $getSecV = getSecV$   
**and**  $isIntV = isIntV$  **and**  $getIntV = getIntV$

**and**  $validTransO = validTransO$  **and**  $istateO = \lambda s. s = ss$   
**and**  $finalO = final$   
**and**  $isSecO = isSecO$  **and**  $getSecO = getSecO$   
**and**  $isIntO = isIntO$  **and**  $getIntO = getIntV$   
**and**  $corrState = corrState$

**apply**( $unfold-locales$ )  
**subgoal by** ( $auto\ simp: final-def$ )  
**subgoal by** ( $auto\ simp: final-def$ )  
**subgoal by** ( $auto\ simp: final-def\ isSecV-def$ )  
**subgoal by** ( $auto\ simp: final-def$ )  
**subgoal by** ( $auto\ simp: final-def$ )  
**subgoal by** ( $auto\ simp: final-iff\ isSecO-def$ ) .

**lemma**  $validFromV: Van.validFromS\ ss\ [ss, ss']$  **unfolding**  $Van.validFromS-def$   
 $Van.validS-def$  **by**  $auto$

**lemma**  $tr1-shape: Opt.validFromS\ ss\ tr1 \Longrightarrow completedFromO\ ss\ tr1 \Longrightarrow tr1 = [ss, ss']$

**unfolding**  $completedFromO-def$  **apply**( $erule\ disjE$ )  
**subgoal by**( $simp\ add: final-iff$ )  
**unfolding**  $Opt.validFromS-def\ Opt.validS-def\ final-iff$   
**apply**( $cases\ tr1, auto\ split: if-splits$ )  
**by** ( $metis\ One-nat-def\ State.distinct\ append-Nil\ append-butlast-last-id\ diff-is-0-eq$   
 $length-butlast\ length-greater-0-conv$   
 $list.size(3)\ nth-Cons-0\ nth-Cons-Suc\ verit-comp-simplify1(3)$ )

**lemma**  $tr1-shape': s1 = ss \Longrightarrow Opt.validFromS\ s1\ tr1 \Longrightarrow completedFromO\ s1\ tr1$   
 $\Longrightarrow tr1 = [ss, ss']$   
**using**  $tr1-shape$  **by**  $auto$

**proposition** *rsecure*

**unfolding** *rsecure-def3*  
**apply**(*intro allI impI, elim conjE*)  
**apply**(*rule exI[of - ss], rule exI[of - [ss, ss']]*)  
**apply**(*rule exI[of - ss], rule exI[of - [ss, ss']]*)  
**by**(*frule tr1-shape', simp-all*)**+**

**lemma** *validSS:Opt.validS [ss, ss'] unfolding Opt.validS-def by auto*

**lemma** *validSS-van:Van.validS [ss, ss'] unfolding Van.validS-def by auto*

**lemma** *reachOs:reachO ss using Opt.reach.Istate by auto*

**lemma** *reachVs:reachV ss using Van.reach.Istate by auto*

**lemma** *reachO':reachO ss' using Opt.validS-reach[of [ss, ss'], OF - validSS] by auto*

**lemma** *reachV':reachV ss' using Van.validS-reach[of [ss, ss'], OF - validSS-van] by auto*

**lemma** *impE-eq:x = x  $\rightarrow$  Q  $\implies$  (Q  $\implies$  Rs)  $\implies$  Rs by auto*

**lemma** *isSecOs:isSecO ss unfolding isSecO-def by auto*

**lemma** *neq-Sec: $\neg$ eqSec ss ss unfolding eqSec-def isSecO-def isSecV-def by auto*

**lemma** *statOs: (sstatO' Eq ss ss) = Eq unfolding sstatO'-def by auto*

**lemma** *noUnwind: shows init:  $\Delta \infty$  ss ss Eq ss ss Eq  $\implies$  unwindCond  $\Delta \implies$  False*

**subgoal premises** *unwind using unwind(2)[unfolded unwindCond-def]*

**apply**–**apply**(*erule allE[of -  $\infty$ ]*)  
**apply**(*erule allE[of - ss], erule allE[of - ss]*)  
**apply**(*erule allE[of - Eq]*)  
**apply**(*erule allE[of - ss], erule allE[of - ss]*)  
**apply**(*erule allE[of - Eq], erule impE*)  
**subgoal using** *reachOs reachVs unwind(1) by auto*  
**apply**(*elim conjE disjE impE-eq*)

**subgoal apply**(*elim exE conjE, unfold proact-def, elim disjE*)

**subgoal for** *v unfolding move-1-def apply (simp split: if-splits)*  
**using** *unwind(2)[unfolded unwindCond-def]*  
**apply**–**apply**(*erule allE[of - v]*)  
**apply**(*erule allE[of - ss], erule allE[of - ss]*)  
**apply**(*erule allE[of - Eq]*)  
**apply**(*erule allE[of - ss'], erule allE[of - ss]*)  
**apply**(*erule allE[of - Eq], erule impE*)  
**subgoal using** *reachOs reachVs reachV' unwind(1) by auto*

```

apply(elim conjE disjE impE-eq)

subgoal apply(elim exE conjE, unfold proact-def, elim disjE)

  subgoal for v unfolding move-1-def by (simp split: if-splits)

    subgoal for v' unfolding move-2-def apply (simp split: if-splits)
      using unwind(2)[unfolded unwindCond-def]
      apply–apply(erule allE[of - v'])
      apply(erule allE[of - ss], erule allE[of - ss])
      apply(erule allE[of - Eq])
      apply(erule allE[of - ss'], erule allE[of - ss'])
      apply(erule allE[of - Eq], erule impE)
      subgoal using reachOs reachVs reachV' reachO' by auto
      apply(elim conjE disjE impE-eq)

    subgoal by(unfold proact-def, auto simp: move-1-def move-2-def move-12-def)

      subgoal unfolding react-def match1-def match1-1-def match1-12-def
        apply(elim conjE impE, simp)
        by(erule allE[of - ss'], auto simp: neq-Sec isSecOs) .

      subgoal unfolding move-12-def by auto .

      unfolding react-def match1-def match1-1-def match1-12-def
      apply(elim conjE impE, simp) by(erule allE[of - ss'], auto simp: neq-Sec
isSecOs)

subgoal for v unfolding move-2-def apply (simp split: if-splits)
  using unwind(2)[unfolded unwindCond-def]
  apply–apply(erule allE[of - v])
  apply(erule allE[of - ss], erule allE[of - ss])
  apply(erule allE[of - Eq])
  apply(erule allE[of - ss], erule allE[of - ss'])
  apply(erule allE[of - Eq], erule impE)
  subgoal using reachOs reachVs reachV' unwind(1) by auto
  apply(elim conjE disjE impE-eq)

subgoal apply(elim exE conjE, unfold proact-def, elim disjE)

  subgoal for v' unfolding move-1-def apply (simp split: if-splits)
    using unwind(2)[unfolded unwindCond-def]
    apply–apply(erule allE[of - v'])
    apply(erule allE[of - ss], erule allE[of - ss])
    apply(erule allE[of - Eq])
    apply(erule allE[of - ss'], erule allE[of - ss'])

```

```

apply(erule allE[of - Eq], erule impE)
subgoal using reachOs reachVs reachV' reachO' by auto
apply(elim conjE disjE impE-eq)

subgoal by(auto simp: proact-def move-1-def move-2-def move-12-def)

subgoal unfolding react-def match1-def match1-1-def match1-12-def
apply(elim conjE impE, simp)
by(erule allE[of - ss^], auto simp: neq-Sec isSecOs) .

subgoal for v unfolding move-2-def by (simp split: if-splits)

subgoal unfolding move-12-def by auto .

unfolding react-def match1-def match1-1-def match1-12-def
apply(elim conjE impE, simp) by(erule allE[of - ss^], auto simp: neq-Sec
isSecOs)

```

```

subgoal for v unfolding move-12-def apply (simp split: if-splits)
using unwind(2)[unfolded unwindCond-def]
apply—apply(erule allE[of - v])
apply(erule allE[of - ss], erule allE[of - ss])
apply(erule allE[of - Eq])
apply(erule allE[of - ss^], erule allE[of - ss^])
apply(erule allE[of - Eq], erule impE)
subgoal using reachOs reachVs reachV' reachO' by (auto simp: statOs)
apply(elim conjE disjE impE-eq)

```

```

subgoal by(auto simp: proact-def move-1-def move-2-def move-12-def)

```

```

subgoal unfolding react-def match1-def match1-12-def match1-1-def ap-
ply(elim conjE impE, simp)
by(erule allE[of - ss^], auto simp: neq-Sec isSecOs) . .

```

```

subgoal unfolding react-def match1-def apply(elim conjE impE, simp)
by(erule allE[of - ss^], auto simp: neq-Sec isSecOs) . .

```

```

lemma incomplete-fin:
  assumes init: initCond Δ
  and unwind: unwindCond Δ
  shows False
  using noUnwind assms[unfolded initCond-def] by auto

```

```

end

```

## 6 Relative Security Unwinding Incompleteness example

Demonstrating a counterexample which is secure but fails in the infintary unwinding

```
theory Incomplete
  imports Unwinding
begin
```

```
no-notation bot ( $\perp$ )
```

```
abbreviation noninform ( $\perp$ ) where  $\perp \equiv \text{undefined}$ 
```

Demonstrating a counterexample which is secure but fails in the unwinding

```
datatype State = ss | ss'
type-synonym secret = State
```

```
fun transit::State  $\Rightarrow$  State  $\Rightarrow$  bool(infix  $\rightarrow I$  55) where
  transit s s' = (if (s = ss  $\wedge$  s' = ss') then True else False)
```

```
lemma transit-singlv:ss  $\rightarrow I$  ss' by auto
```

```
lemma transit-iff:s  $\rightarrow I$  s'  $\longleftrightarrow$  (s = ss  $\wedge$  s' = ss') by auto
```

```
definition final x  $\equiv \forall y. \neg (\rightarrow I) x y$ 
```

```
lemma final-sv'[simp]:final ss' unfolding final-def transit-iff by auto
```

```
lemma final-iff: final s  $\longleftrightarrow$  s = ss' unfolding final-def transit-iff by (auto,metis State.exhaust)
```

Vanilla-semantics system model

```
type-synonym stateV = State
fun validTransV where validTransV (s,s') = s  $\rightarrow I$  s'
```

No secrets or interaction

```
fun isIntV :: stateV  $\Rightarrow$  bool where isIntV s = False
fun getIntV::stateV  $\Rightarrow$  nat  $\times$  nat where getIntV s = ( $\perp$ , $\perp$ )
```

```
definition isSecV :: stateV  $\Rightarrow$  bool where isSecV s = False
fun getSecV :: stateV  $\Rightarrow$  secret where getSecV s = s
```

The optimization-enhanced system model

**type-synonym**  $stateO = State$   
**fun**  $validTransO$  **where**  $validTransO (s,s') = s \rightarrow I s'$

No interaction, only isSec at starting state

**fun**  $isIntO :: stateO \Rightarrow bool$  **where**  $isIntO s = False$   
**fun**  $getIntO :: stateO \Rightarrow nat \times nat$  **where**  $getIntO s = (\perp, \perp)$

**definition**  $isSecO :: stateO \Rightarrow bool$  **where**  $isSecO s = (if s = ss then True else False)$

**fun**  $getSecO :: stateO \Rightarrow secret$  **where**  $getSecO s = s$

$corrState$

**fun**  $corrState :: stateV \Rightarrow stateO \Rightarrow bool$  **where**  
 $corrState\ cfGO\ cfGA = True$

**interpretation**  $Rel\text{-}Sec$

**where**  $validTransV = validTransV$  **and**  $istateV = \lambda s. s = ss$   
**and**  $finalV = final$   
**and**  $isSecV = isSecV$  **and**  $getSecV = getSecV$   
**and**  $isIntV = isIntV$  **and**  $getIntV = getIntV$

**and**  $validTransO = validTransO$  **and**  $istateO = \lambda s. s = ss$   
**and**  $finalO = final$   
**and**  $isSecO = isSecO$  **and**  $getSecO = getSecO$   
**and**  $isIntO = isIntO$  **and**  $getIntO = getIntO$   
**and**  $corrState = corrState$

**apply**( $unfold\text{-}locales$ )  
**subgoal** **by** ( $auto\ simp: final\text{-}def$ )  
**subgoal** **by** ( $auto\ simp: final\text{-}def$ )  
**subgoal** **by** ( $auto\ simp: final\text{-}def\ isSecV\text{-}def$ )  
**subgoal** **by** ( $auto\ simp: final\text{-}def$ )  
**subgoal** **by** ( $auto\ simp: final\text{-}def$ )  
**subgoal** **by** ( $auto\ simp: final\text{-}iff\ isSecO\text{-}def$ ) .

**lemma**  $validTrFinite: Opt.lvalidFromS\ ss\ tr1 \Longrightarrow lfinite\ tr1$

**unfolding**  $Opt.lvalidFromS\text{-}def\ Opt.lvalidS\text{-}def$   
**by** ( $auto,metis\ State.distinct(1)\ enat\text{-}ord\text{-}code(4)\ idiff\text{-}infinity\ llength\text{-}eq\text{-}infty\text{-}conv\text{-}lfinite$ )

**lemma**  $tr1\text{-}shape: Opt.lvalidFromS\ ss\ tr1 \Longrightarrow lcompletedFromO\ ss\ tr1 \Longrightarrow tr1 = [[ss, ss]]$

**unfolding**  $Opt.lcompletedFrom\text{-}def$  **apply**( $erule\ impE$ )  
**subgoal** **by**( $simp\ add: validTrFinite$ )  
**apply**( $frule\ validTrFinite$ )  
**unfolding**  $Opt.lvalidFromS\text{-}def\ Opt.lvalidS\text{-}def\ final\text{-}iff$   
**apply**( $cases\ tr1, auto\ split: if\text{-}splits$ )  
**subgoal** **for**  $tr1'$

```

apply(unfold nth-list-of[symmetric, of tr1'])
apply(unfold nth-list-of[symmetric, of (ss $ tr1')], unfolded lfinite-LCons)
apply(unfold length-list-of[symmetric])
apply(cases tr1', simp)
subgoal premises p
using p apply-apply(erule allE[of - 0], simp split: if-splits, metis llist-of.simps(1))
llist-of-list-of)
using p apply-by(erule allE[of - 1], simp split: if-splits) . .

```

```

lemma tr1-shape':s1 = ss  $\implies$  Opt.lvalidFromS s1 tr1  $\implies$  lcompletedFromO s1
tr1  $\implies$  tr1 = [[ss, ss']]
using tr1-shape by auto

```

```

proposition lrsecure
unfolding lrsecure-def2
apply(intro allI impI, elim conjE)
apply(rule exI[of - ss],rule exI[of - [[ss, ss']]])
apply(rule exI[of - ss],rule exI[of - [[ss, ss']]])
by(frule tr1-shape', simp-all)+

```

```

lemma validSS:Opt.validS [ss, ss'] unfolding Opt.validS-def by auto
lemma validSS-van:Van.validS [ss, ss'] unfolding Van.validS-def by auto

```

```

lemma reachOs:reachO ss using Opt.reach.Istate by auto
lemma reachVs:reachV ss using Van.reach.Istate by auto
lemma reachO':reachO ss' using Opt.validS-reach[of [ss, ss'], OF - validSS] by
auto
lemma reachV':reachV ss' using Van.validS-reach[of [ss, ss'], OF - validSS-van]
by auto

```

```

lemma impE-eq:x = x  $\longrightarrow$  Q  $\implies$  (Q  $\implies$  Rs)  $\implies$  Rs by auto

```

```

lemma isSecOs:isSecO ss unfolding isSecO-def by auto
lemma neq-Sec:¬eqSec ss ss unfolding eqSec-def isSecO-def isSecV-def by auto

```

```

lemma statOs: (sstatO' Eq ss ss) = Eq unfolding sstatO'-def by auto

```

```

lemma noUnwind: shows init: Δ ∞ ∞ ∞ ss ss Eq ss ss Eq  $\implies$  unwindCond Δ
 $\implies$  False
subgoal premises unwind using unwind(2)[unfolded unwindCond-def]

```

**apply**–**apply**(*erule allE*[of -  $\infty$ ],*erule allE*[of -  $\infty$ ],*erule allE*[of -  $\infty$ ])  
**apply**(*erule allE*[of - *ss*], *erule allE*[of - *ss*])  
**apply**(*erule allE*[of - *Eq*])  
**apply**(*erule allE*[of - *ss*], *erule allE*[of - *ss*])  
**apply**(*erule allE*[of - *Eq*], *erule impE*)  
**subgoal using** *reachOs reachVs unwind(1)* **by auto**  
**apply**(*elim conjE disjE impE-eq*)

**subgoal apply**(*elim exE conjE, unfold proact-def, elim disjE*)

**subgoal for** *v* **unfolding** *move-1-def* **apply** (*simp split: if-splits*)  
**using** *unwind(2)[unfolded unwindCond-def]*  
**apply**–**apply**(*erule allE*[of - *v*],*erule allE*[of -  $\infty$ ],*erule allE*[of -  $\infty$ ])  
**apply**(*erule allE*[of - *ss*], *erule allE*[of - *ss*])  
**apply**(*erule allE*[of - *Eq*])  
**apply**(*erule allE*[of - *ss*'], *erule allE*[of - *ss*])  
**apply**(*erule allE*[of - *Eq*], *erule impE*)  
**subgoal using** *reachOs reachVs reachV' unwind(1)* **by auto**  
**apply**(*elim conjE disjE impE-eq*)

**subgoal apply**(*elim exE conjE, unfold proact-def, elim disjE*)

**subgoal for** *v* **unfolding** *move-1-def* **by** (*simp split: if-splits*)

**subgoal for** *v'* **unfolding** *move-2-def* **apply** (*simp split: if-splits*)  
**using** *unwind(2)[unfolded unwindCond-def]*  
**apply**–**apply**(*erule allE*[of - *v'*],*erule allE*[of -  $\infty$ ],*erule allE*[of -  $\infty$ ])  
**apply**(*erule allE*[of - *ss*], *erule allE*[of - *ss*])  
**apply**(*erule allE*[of - *Eq*])  
**apply**(*erule allE*[of - *ss*'], *erule allE*[of - *ss*'])  
**apply**(*erule allE*[of - *Eq*], *erule impE*)  
**subgoal using** *reachOs reachVs reachV' reachO'* **by auto**  
**apply**(*elim conjE disjE impE-eq*)

**subgoal by**(*unfold proact-def, auto simp: move-1-def move-2-def move-12-def*)

**subgoal unfolding** *react-def match1-def match1-1-def match1-12-def*  
**apply**(*elim conjE impE, simp*)  
**by**(*erule allE*[of - *ss*'], *auto simp: neq-Sec isSecOs*) .

**subgoal unfolding** *move-12-def* **by auto** .

**unfolding** *react-def match1-def match1-1-def match1-12-def*  
**apply**(*elim conjE impE, simp*) **by**(*erule allE*[of - *ss*'], *auto simp: neq-Sec isSecOs*)

**subgoal for  $v$  unfolding  $move-2-def$  apply** (*simp split: if-splits*)  
**using**  $unwind(2)[unfolded\ unwindCond-def]$   
**apply–apply**( $erule\ allE[of - v], erule\ allE[of - \infty], erule\ allE[of - \infty]$ )  
**apply**( $erule\ allE[of - ss], erule\ allE[of - ss]$ )  
**apply**( $erule\ allE[of - Eq]$ )  
**apply**( $erule\ allE[of - ss], erule\ allE[of - ss']$ )  
**apply**( $erule\ allE[of - Eq], erule\ impE$ )  
**subgoal using**  $reachOs\ reachVs\ reachV'\ unwind(1)$  **by auto**  
**apply**( $elim\ conjE\ disjE\ impE-eq$ )

**subgoal apply**( $elim\ exE\ conjE, unfold\ proact-def, elim\ disjE$ )

**subgoal for  $v'$  unfolding  $move-1-def$  apply** (*simp split: if-splits*)  
**using**  $unwind(2)[unfolded\ unwindCond-def]$   
**apply–apply**( $erule\ allE[of - v'], erule\ allE[of - \infty], erule\ allE[of - \infty]$ )  
**apply**( $erule\ allE[of - ss], erule\ allE[of - ss]$ )  
**apply**( $erule\ allE[of - Eq]$ )  
**apply**( $erule\ allE[of - ss'], erule\ allE[of - ss']$ )  
**apply**( $erule\ allE[of - Eq], erule\ impE$ )  
**subgoal using**  $reachOs\ reachVs\ reachV'\ reachO'$  **by auto**  
**apply**( $elim\ conjE\ disjE\ impE-eq$ )

**subgoal by**(*auto simp: proact-def move-1-def move-2-def move-12-def*)

**subgoal unfolding  $react-def\ match1-def\ match1-1-def\ match1-12-def$**   
**apply**( $elim\ conjE\ impE, simp$ )  
**by**( $erule\ allE[of - ss'], auto\ simp: neq-Sec\ isSecOs$ ) .

**subgoal for  $v$  unfolding  $move-2-def$  by** (*simp split: if-splits*)

**subgoal unfolding  $move-12-def$  by auto** .

**unfolding  $react-def\ match1-def\ match1-1-def\ match1-12-def$**   
**apply**( $elim\ conjE\ impE, simp$ ) **by**( $erule\ allE[of - ss'], auto\ simp: neq-Sec\ isSecOs$ )

**subgoal for  $v$  unfolding  $move-12-def$  apply** (*simp split: if-splits*)  
**using**  $unwind(2)[unfolded\ unwindCond-def]$   
**apply–apply**( $erule\ allE[of - v], erule\ allE[of - \infty], erule\ allE[of - \infty]$ )  
**apply**( $erule\ allE[of - ss], erule\ allE[of - ss]$ )  
**apply**( $erule\ allE[of - Eq]$ )  
**apply**( $erule\ allE[of - ss'], erule\ allE[of - ss']$ )  
**apply**( $erule\ allE[of - Eq], erule\ impE$ )  
**subgoal using**  $reachOs\ reachVs\ reachV'\ reachO'$  **by** (*auto simp: statOs*)

```

apply(elim conjE disjE impE-eq)

subgoal by(auto simp: proact-def move-1-def move-2-def move-12-def)

  subgoal unfolding react-def match1-def match1-12-def match1-1-def ap-
ply(elim conjE impE, simp)
    by(erule allE[of - ss^], auto simp: neq-Sec isSecOs) . .

subgoal unfolding react-def match1-def apply(elim conjE impE, simp)
  by(erule allE[of - ss^], auto simp: neq-Sec isSecOs) . .

lemma incomplete-inf:
  assumes init: initCond Δ
  and unwind: unwindCond Δ
  shows False
  using noUnwind assms[unfolded initCond-def] by auto

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.