

# 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 . . . . .	8
<b>3</b>	<b>Unwinding Proof Method for Finitary Relative Security</b>	<b>12</b>
3.1	The types and operators underlying unwinding: status, matching operators, etc. . . . .	12
3.2	The definition of unwinding . . . . .	17
3.3	The soundness of unwinding . . . . .	18
3.4	Compositional unwinding . . . . .	19
<b>4</b>	<b>Unwinding Proof Method for Relative Security</b>	<b>29</b>
4.1	The types and operators underlying unwinding: status, matching operators, etc. . . . .	29

4.2	The definition of unwinding . . . . .	34
4.3	The soundness of unwinding . . . . .	35
4.4	Compositional unwinding . . . . .	59
<b>5</b>	<b>Relative Security Unwinding Incompleteness example</b>	<b>68</b>
<b>6</b>	<b>Relative Security Unwinding Incompleteness example</b>	<b>71</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*  
 ⟨*proof*⟩

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

**sublocale**  $Relative\text{-}Security'\text{-}fin < Relative\text{-}Security''\text{-}fin$   
**where**  $leakViaV = Van.leakVia$  **and**  $leakViaO = Opt.leakVia$   
 $\langle proof \rangle$

**context**  $Relative\text{-}Security'\text{-}fin$   
**begin**

**lemma**  $rsecure\text{-}def2$ :

$rsecure \longleftrightarrow$

$(\forall s1\ tr1\ s2\ tr2.$

$istateO\ s1 \wedge Opt.validFromS\ s1\ tr1 \wedge Opt.completedFrom\ s1\ tr1 \wedge$   
 $istateO\ s2 \wedge Opt.validFromS\ s2\ tr2 \wedge Opt.completedFrom\ s2\ tr2 \wedge$   
 $AO\ tr1 = AO\ tr2 \wedge OO\ tr1 \neq OO\ tr2$

$\longrightarrow$

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

$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$   
 $Van.validFromS\ sv1\ trv1 \wedge Van.completedFrom\ sv1\ trv1 \wedge$   
 $Van.validFromS\ sv2\ trv2 \wedge Van.completedFrom\ sv2\ trv2 \wedge$   
 $SV\ trv1 = SO\ tr1 \wedge SV\ trv2 = SO\ tr2 \wedge$   
 $AV\ trv1 = AV\ trv2 \wedge OV\ trv1 \neq OV\ trv2))$

$\langle proof \rangle$

**end**

**locale**  $Statewise\text{-}Attacker\text{-}Mod = System\text{-}Mod\ istate\ validTrans\ final$

**for**  $istate :: 'state \Rightarrow bool$  **and**  $validTrans :: 'state \times 'state \Rightarrow bool$  **and**  $final :: 'state \Rightarrow bool$

+

**fixes**

$isSec :: 'state \Rightarrow bool$  **and**  $getSec :: 'state \Rightarrow 'secret$

**and**

$isInt :: 'state \Rightarrow bool$  **and**  $getInt :: 'state \Rightarrow 'act \times 'obs$

**assumes**  $final\text{-}not\text{-}isInt: \bigwedge s. final\ s \Longrightarrow \neg isInt\ s$

**and**  $final\text{-}not\text{-}isSec: \bigwedge s. final\ s \Longrightarrow \neg isSec\ s$

**begin**

**definition**  $getAct :: 'state \Rightarrow 'act$  **where**

$getAct = fst\ o\ getInt$

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

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

**sublocale**  $A: FiltermapBL isInt getAct A$   
 $\langle proof \rangle$

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

**sublocale**  $O: FiltermapBL isInt getObs O$   
 $\langle proof \rangle$

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

**sublocale**  $S: FiltermapBL isSec getSec S$   
 $\langle proof \rangle$

**end**

**sublocale**  $Statewise-Attacker-Mod < Attacker-Mod-fin$   
**where**  $S = S$  **and**  $A = A$  **and**  $O = O$   
 $\langle proof \rangle$

**locale**  $Rel-Sec =$

$Van: Statewise-Attacker-Mod \text{ istateV } validTransV \text{ finalV } isSecV \text{ getSecV } isIntV$   
 $getIntV$

+

$Opt: Statewise-Attacker-Mod \text{ istateO } validTransO \text{ finalO } isSecO \text{ getSecO } isIntO$   
 $getIntO$

**for**  $validTransV :: 'stateV \times 'stateV \Rightarrow bool$

**and**  $istateV :: 'stateV \Rightarrow bool$  **and**  $finalV :: 'stateV \Rightarrow bool$   
**and**  $isSecV :: 'stateV \Rightarrow bool$  **and**  $getSecV :: 'stateV \Rightarrow 'secret$   
**and**  $isIntV :: 'stateV \Rightarrow bool$  **and**  $getIntV :: 'stateV \Rightarrow 'actV \times 'obsV$

**and**  $validTransO :: 'stateO \times 'stateO \Rightarrow bool$   
**and**  $istateO :: 'stateO \Rightarrow bool$  **and**  $finalO :: 'stateO \Rightarrow bool$   
**and**  $isSecO :: 'stateO \Rightarrow bool$  **and**  $getSecO :: 'stateO \Rightarrow 'secret$   
**and**  $isIntO :: 'stateO \Rightarrow bool$  **and**  $getIntO :: 'stateO \Rightarrow 'actO \times 'obsO$

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

**sublocale**  $Rel\text{-}Sec < Relative\text{-}Security'\text{-}fin$   
**where**  $SV = Van.S$  **and**  $AV = Van.A$  **and**  $OV = Van.O$   
**and**  $SO = Opt.S$  **and**  $AO = Opt.A$  **and**  $OO = Opt.O$   
 $\langle proof \rangle$

**context**  $Rel\text{-}Sec$   
**begin**

**abbreviation**  $getObsV :: 'stateV \Rightarrow 'obsV$  **where**  $getObsV \equiv Van.getObs$   
**abbreviation**  $getActV :: 'stateV \Rightarrow 'actV$  **where**  $getActV \equiv Van.getAct$   
**abbreviation**  $getObsO :: 'stateO \Rightarrow 'obsO$  **where**  $getObsO \equiv Opt.getObs$   
**abbreviation**  $getActO :: 'stateO \Rightarrow 'actO$  **where**  $getActO \equiv Opt.getAct$

**abbreviation**  $reachV$  **where**  $reachV \equiv Van.reach$   
**abbreviation**  $reachO$  **where**  $reachO \equiv Opt.reach$

**abbreviation**  $completedFromV :: 'stateV \Rightarrow 'stateV\ list \Rightarrow bool$  **where**  $completedFromV \equiv Van.completedFrom$   
**abbreviation**  $completedFromO :: 'stateO \Rightarrow 'stateO\ list \Rightarrow bool$  **where**  $completedFromO \equiv Opt.completedFrom$

**lemmas**  $completedFromV\text{-}def = Van.completedFrom\text{-}def$   
**lemmas**  $completedFromO\text{-}def = Opt.completedFrom\text{-}def$

**lemma**  $rsecure\text{-}def3$ :

$rsecure \longleftrightarrow$

$(\forall s1\ tr1\ s2\ tr2.$

$istateO\ s1 \wedge Opt.validFromS\ s1\ tr1 \wedge completedFromO\ s1\ tr1 \wedge$   
 $istateO\ s2 \wedge Opt.validFromS\ s2\ tr2 \wedge completedFromO\ s2\ tr2 \wedge$   
 $Opt.A\ tr1 = Opt.A\ tr2 \wedge Opt.O\ tr1 \neq Opt.O\ tr2 \wedge$   
 $(isIntO\ s1 \wedge isIntO\ s2 \longrightarrow getActO\ s1 = getActO\ s2)$

$\longrightarrow$

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

$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$   
 $Van.validFromS\ sv1\ trv1 \wedge completedFromV\ sv1\ trv1 \wedge$   
 $Van.validFromS\ sv2\ trv2 \wedge completedFromV\ sv2\ trv2 \wedge$

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

$$\text{Van.A } trv1 = \text{Van.A } trv2 \wedge \text{Van.O } trv1 \neq \text{Van.O } trv2))$$
 <proof>

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

**lemma**  $eqSec\text{-}S\text{-}Cons'$ :  
 $eqSec\ trnO\ trnA \implies$   
 $(\text{Van.S } (trnO \# trO') = \text{Opt.S } (trnA \# trA')) \implies \text{Van.S } trO' = \text{Opt.S } trA'$   
 <proof>

**lemma**  $eqSec\text{-}S\text{-}Cons[simp]$ :  
 $eqSec\ trnO\ trnA \implies trO' = [] \longleftrightarrow trA' = [] \implies$   
 $(\text{Van.S } (trnO \# trO') = \text{Opt.S } (trnA \# trA')) \longleftrightarrow (\text{Van.S } trO' = \text{Opt.S } trA')$   
 <proof>

end

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

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

```
locale Attacker-Mod = System-Mod istate validTrans final
for istate :: 'state  $\Rightarrow$  bool and validTrans :: 'state  $\times$  'state  $\Rightarrow$  bool and final ::
'state  $\Rightarrow$  bool
+
fixes S :: 'state llist  $\Rightarrow$  'secret llist
and A :: 'state ltrace  $\Rightarrow$  'act llist
and O :: 'state ltrace  $\Rightarrow$  '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
<proof>
```

### 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 ⇒ bool and finalV :: 'stateV ⇒ bool
and leakViaV :: 'stateV llist ⇒ 'stateV llist ⇒ 'leak ⇒ bool

and validTransO :: 'stateO × 'stateO ⇒ bool
and istateO :: 'stateO ⇒ bool and finalO :: 'stateO ⇒ bool
and leakViaO :: 'stateO llist ⇒ 'stateO llist ⇒ 'leak ⇒ bool

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

```

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

```

lrsecure ≡ ∀ l s1 tr1 s2 tr2.
  istateO s1 ∧ Opt.lvalidFromS s1 tr1 ∧ Opt.lcompletedFrom s1 tr1 ∧
  istateO s2 ∧ Opt.lvalidFromS s2 tr2 ∧ Opt.lcompletedFrom s2 tr2 ∧
  leakViaO tr1 tr2 l
  →
  (∃ 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 ∧
    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
⟨proof⟩

```

**context** *Relative-Security'*

**begin**

**lemma** *lrsecure-def2*:

*lrsecure*  $\longleftrightarrow$

( $\forall s1\ tr1\ s2\ tr2.$

$istateO\ s1 \wedge Opt.lvalidFromS\ s1\ tr1 \wedge Opt.lcompletedFrom\ s1\ tr1 \wedge$   
 $istateO\ s2 \wedge Opt.lvalidFromS\ s2\ tr2 \wedge Opt.lcompletedFrom\ s2\ tr2 \wedge$   
 $AO\ tr1 = AO\ tr2 \wedge OO\ tr1 \neq OO\ tr2$

$\longrightarrow$

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

$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$   
 $Van.lvalidFromS\ sv1\ trv1 \wedge Van.lcompletedFrom\ sv1\ trv1 \wedge$   
 $Van.lvalidFromS\ sv2\ trv2 \wedge Van.lcompletedFrom\ sv2\ trv2 \wedge$   
 $SV\ trv1 = SO\ tr1 \wedge SV\ trv2 = SO\ tr2 \wedge$   
 $AV\ trv1 = AV\ trv2 \wedge OV\ trv1 \neq OV\ trv2$ )

*<proof>*

**end**

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

**definition** *lA* :: 'state ltrace  $\Rightarrow$  'act llist **where**

*lA* *tr*  $\equiv$  *lfiltermap isInt getAct (lbutlast tr)*

**sublocale** *lA*: *LfiltermapBL isInt getAct lA*

*<proof>*

**lemma** *lA*: *lcompletedFrom s tr*  $\Longrightarrow$  *lA tr* = *lmap getAct (lfilter isInt tr)*

*<proof>*

**definition** *lO* :: 'state ltrace  $\Rightarrow$  'obs llist **where**

*lO* *tr*  $\equiv$  *lfiltermap isInt getObs (lbutlast tr)*

**sublocale** *lO*: *LfiltermapBL isInt getObs lO*

*<proof>*

**lemma** *lO*: *lcompletedFrom s tr*  $\Longrightarrow$  *lO tr* = *lmap getObs (lfilter isInt tr)*

*<proof>*

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

**sublocale**  $lS: LfiltermapBL\ isSec\ getSec\ lS$   
*<proof>*

**lemma**  $lS: lcompletedFrom\ s\ tr \Longrightarrow lS\ tr = lmap\ getSec\ (lfilter\ isSec\ tr)$   
*<proof>*

**end**

**sublocale**  $Statewise-Attacker-Mod < Attacker-Mod$   
**where**  $S = lS$  **and**  $A = lA$  **and**  $O = lO$   
*<proof>*

**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**  $OO = Opt.lO$   
*<proof>*

**context**  $Rel-Sec$   
**begin**

**abbreviation**  $lcompletedFromV :: 'stateV \Rightarrow 'stateV\ llist \Rightarrow bool$  **where**  $lcompletedFromV \equiv Van.lcompletedFrom$

**abbreviation**  $lcompletedFromO :: 'stateO \Rightarrow 'stateO\ llist \Rightarrow bool$  **where**  $lcompletedFromO \equiv Opt.lcompletedFrom$

**lemma**  $eqSec-lS-Cons'$ :  
 $eqSec\ trnO\ trnA \Longrightarrow$   
 $(Van.lS\ (trnO\ \$\ trO') = Opt.lS\ (trnA\ \$\ trA')) \Longrightarrow Van.lS\ trO' = Opt.lS\ trA'$   
*<proof>*

**lemma**  $eqSec-lS-Cons[simp]$ :  
 $eqSec\ trnO\ trnA \Longrightarrow trO' = [] \longleftrightarrow trA' = [] \Longrightarrow$   
 $(Van.lS\ (trnO\ \$\ trO') = Opt.lS\ (trnA\ \$\ trA')) \longleftrightarrow (Van.lS\ trO' = Opt.lS\ trA')$   
*<proof>*

**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  $\Delta s1 s1' s2 statA sv1 sv2 statO \equiv$ 
```

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

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

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

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

**lemma**  $match1-1-mono:$   
 $\Delta \leq \Delta' \implies match1-1\ \Delta\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \implies match1-1\ \Delta'\ s1\ s1'\ s2$   
 $statA\ sv1\ sv2\ statO$   
 $\langle proof \rangle$

**lemma**  $match1-12-mono:$   
 $\Delta \leq \Delta' \implies match1-12\ \Delta\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \implies match1-12\ \Delta'\ s1\ s1'$   
 $s2\ statA\ sv1\ sv2\ statO$   
 $\langle proof \rangle$

**lemma**  $match1-mono:$   
**assumes**  $\Delta \leq \Delta'$   
**shows**  $match1\ \Delta\ s1\ s2\ statA\ sv1\ sv2\ statO \implies match1\ \Delta'\ s1\ s2\ statA\ sv1\ sv2$   
 $statO$   
 $\langle proof \rangle$

**definition**  $match2-1\ \Delta\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO \equiv$   
 $\exists\ sv2'.\ validTransV\ (sv2,sv2') \wedge$   
 $\Delta \infty\ s1\ s2'\ statA\ sv1\ sv2'\ statO$

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

**definition**  $match2\ \Delta\ s1\ s2\ statA\ sv1\ sv2\ statO \equiv$   
 $\neg\ isIntO\ s2 \longrightarrow$   
 $(\forall\ s2'.\ validTransO\ (s2,s2')$   
 $\longrightarrow$

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

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

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

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

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

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

**lemma** *match2-mono*:

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

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

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

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

$\text{let } \text{statA}' = \text{sstatA}' \ \text{statA } s1 \ s2 \ \text{in}$   
 $\text{validTransO } (s1, s1') \wedge$   
 $\text{validTransO } (s2, s2') \wedge$   
 $\text{Opt.eqAct } s1 \ s2 \wedge$   
 $\text{isIntO } s1 \wedge \text{isIntO } s2$   
 $\longrightarrow$

$$\begin{aligned}
& (\neg \text{isSecO } s1 \wedge \neg \text{isSecO } s2 \wedge (\text{statA} = \text{statA}' \vee \text{statO} = \text{Diff}) \wedge \Delta \infty s1' s2' \\
& \text{statA}' sv1 sv2 \text{statO}) \\
& \vee \\
& (\neg \text{isSecO } s2 \wedge \text{eqSec } sv1 s1 \wedge \neg \text{isIntV } sv1 \wedge \\
& (\text{statA} = \text{statA}' \vee \text{statO} = \text{Diff}) \wedge \\
& \text{match12-1 } \Delta s1' s2' \text{statA}' sv1 sv2 \text{statO}) \\
& \vee \\
& (\neg \text{isSecO } s1 \wedge \text{eqSec } sv2 s2 \wedge \neg \text{isIntV } sv2 \wedge \\
& (\text{statA} = \text{statA}' \vee \text{statO} = \text{Diff}) \wedge \\
& \text{match12-2 } \Delta s1' s2' \text{statA}' sv1 sv2 \text{statO}) \\
& \vee \\
& (\text{eqSec } sv1 s1 \wedge \text{eqSec } sv2 s2 \wedge \text{Van.eqAct } sv1 sv2 \wedge \\
& \text{match12-12 } \Delta s1' s2' \text{statA}' sv1 sv2 \text{statO})
\end{aligned}$$

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

**lemma** *match12-simpleI*:

**assumes**  $\bigwedge s1' s2' \text{statA}'$ .

$\text{statA}' = \text{sstatA}' \text{statA } s1 s2 \implies$

$\text{validTransO } (s1, s1') \implies$

$\text{validTransO } (s2, s2') \implies$

$\text{Opt.eqAct } s1 s2 \implies$

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

$\vee$

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

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

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

*<proof>*

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

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

*<proof>*

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

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

*<proof>*

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

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

*<proof>*

**lemma** *match12-mono*:

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

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

*statO*  
*<proof>*

**definition** *react*  $\Delta$  *s1 s2 statA sv1 sv2 statO  $\equiv$   
*match1*  $\Delta$  *s1 s2 statA sv1 sv2 statO  
 $\wedge$   
*match2*  $\Delta$  *s1 s2 statA sv1 sv2 statO  
 $\wedge$   
*match12*  $\Delta$  *s1 s2 statA sv1 sv2 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** *react*  $\Delta$  *s1 s2 statA sv1 sv2 statO  $\implies$  *react*  $\Delta'$  *s1 s2 statA sv1 sv2 statO  
*<proof>***

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

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

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

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

*<proof>*

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

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

*<proof>*

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

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

*<proof>*

**lemma** *proact-mono*:

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

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

*<proof>*

### 3.2 The definition of unwinding

**definition** *unwindCond* ::

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

**where**

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

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

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

$\longrightarrow$

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

$\wedge$

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

$\wedge$

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

$\vee$

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

)

**lemma** *unwindCond-simpleI*:

**assumes**

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

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

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

$\implies$

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

$s2$ )  
**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**  $unwindCond \Delta$   
 $\langle proof \rangle$

### 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:  $completedFromO s1 tr1 \implies completedFromO s2 tr2 \implies$

$\psi s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2$   
 $\implies completedFromV sv1 trv1 \wedge completedFromV sv2 trv2$   
 $\langle proof \rangle$

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

**lemma**  $rsecure$ -strong:

**assumes**

$\bigwedge s1 tr1 s2 tr2.$   
 $istateO s1 \wedge Opt.validFromS s1 tr1 \wedge completedFromO s1 tr1 \wedge$   
 $istateO s2 \wedge Opt.validFromS s2 tr2 \wedge completedFromO s2 tr2 \wedge$   
 $Opt.A tr1 = Opt.A tr2 \wedge (isIntO s1 \wedge isIntO s2 \longrightarrow getActO s1 = getActO$   
 $s2)$   
 $\implies$   
 $\exists sv1 trv1 sv2 trv2.$

$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$   
 $\psi\ s1\ tr1\ s2\ tr2\ Eq\ sv1\ trv1\ sv2\ trv2$   
**shows** *rsecure*  
 $\langle proof \rangle$

**proposition** *unwindCond-ex-ψ*:

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

**and**  $\Delta$ :  $\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$  **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**  $\exists\ trv1\ trv2. \psi\ s1\ tr1\ s2\ tr2\ statO\ sv1\ trv1\ sv2\ trv2$

$\langle proof \rangle$

**theorem** *unwind-rsecure*:

**assumes** *init*: *initCond*  $\Delta$

**and** *unwind*: *unwindCond*  $\Delta$

**shows** *rsecure*

$\langle proof \rangle$

### 3.4 Compositional unwinding

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

**definition** *unwindIntoCond* ::

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

**where**

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

*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** *unwindIntoCond-simpleI*:

**assumes**

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

*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**  
 $\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$   
 $reachO\ s1 \implies reachO\ s2 \implies reachV\ sv1 \implies reachV\ sv2 \implies$   
 $\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $statA = Eq$   
 $\implies$   
 $isIntO\ s1 \longleftrightarrow isIntO\ s2$   
 $\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$   
 $reachO\ s1 \implies reachO\ s2 \implies reachV\ sv1 \implies reachV\ sv2 \implies$   
 $\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$   
 $\implies$   
 $react\ \Delta'\ s1\ s2\ statA\ sv1\ sv2\ statO$   
**shows** *unwindIntoCond*  $\Delta\ \Delta'$   
*<proof>*

**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*  $\Delta\ \Delta'$

*<proof>*

**lemma** *unwindIntoCond-simpleIB*:

**assumes**

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

$s2)$   
**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*  $\Delta \Delta'$   
 $\langle proof \rangle$

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

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

**definition** *oor where*  
 $oor \Delta \Delta_2 \equiv \lambda w s1 s2 statA sv1 sv2 statO.$   
 $\Delta w s1 s2 statA sv1 sv2 statO \vee \Delta_2 w s1 s2 statA sv1 sv2 statO$

**lemma** *oorI1*:  
 $\Delta w s1 s2 statA sv1 sv2 statO \implies oor \Delta \Delta_2 w s1 s2 statA sv1 sv2 statO$   
 $\langle proof \rangle$

**lemma** *oorI2*:  
 $\Delta_2 w s1 s2 statA sv1 sv2 statO \implies oor \Delta \Delta_2 w s1 s2 statA sv1 sv2 statO$

*<proof>*

**definition oor3 where**

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

$\Delta w s1 s2 statA sv1 sv2 statO \vee \Delta_2 w s1 s2 statA sv1 sv2 statO \vee$   
 $\Delta_3 w s1 s2 statA sv1 sv2 statO$

**lemma oor3I1:**

$\Delta w s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w s1 s2 statA sv1 sv2 statO$

*<proof>*

**lemma oor3I2:**

$\Delta_2 w s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w s1 s2 statA sv1 sv2 statO$

*<proof>*

**lemma oor3I3:**

$\Delta_3 w s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w s1 s2 statA sv1 sv2 statO$

*<proof>*

**definition oor4 where**

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

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

**lemma oor4I1:**

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

*<proof>*

**lemma oor4I2:**

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

*<proof>*

**lemma oor4I3:**

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

*<proof>*

**lemma oor4I4:**

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

*<proof>*

**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:**

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

*<proof>*

**lemma** *oor5I2*:

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

*<proof>*

**lemma** *oor5I3*:

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

*<proof>*

**lemma** *oor5I4*:

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

*<proof>*

**lemma** *oor5I5*:

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

*<proof>*

**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*:

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

*<proof>*

**lemma** *oor6I2*:

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

*<proof>*

**lemma** *oor6I3*:

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

*<proof>*

**lemma** *oor6I4*:

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

*<proof>*

**lemma** *oor6I5*:

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

**lemma** *oor6I6*:

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

**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⟩

**lemma** *isIntO-match1*: *isIntO*  $s1 \implies match1 \Delta s1 s2 statA sv1 sv2 statO$   
 ⟨proof⟩

**lemma** *isIntO-match2*: *isIntO*  $s2 \implies match2 \Delta s1 s2 statA sv1 sv2 statO$   
 ⟨proof⟩

**lemma** *match1-1-oorI1*:

$match1-1 \Delta s1 s1' s2 statA sv1 sv2 statO \implies$   
 $match1-1 (oor \Delta \Delta_2) s1 s1' s2 statA sv1 sv2 statO$   
 ⟨proof⟩

**lemma** *match1-1-oorI2*:

$match1-1 \Delta_2 s1 s1' s2 statA sv1 sv2 statO \implies$   
 $match1-1 (oor \Delta \Delta_2) s1 s1' s2 statA sv1 sv2 statO$   
 ⟨proof⟩

**lemma** *match1-oorI1*:

$match1 \Delta s1 s2 statA sv1 sv2 statO \implies$   
 $match1 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
 ⟨proof⟩

**lemma** *match1-oorI2*:

$match1 \Delta_2 s1 s2 statA sv1 sv2 statO \implies$   
 $match1 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
 <proof>

**lemma** *match2-1-oorI1*:  
 $match2-1 \Delta s1 s2 s2' statA sv1 sv2 statO \implies$   
 $match2-1 (oor \Delta \Delta_2) s1 s2 s2' statA sv1 sv2 statO$   
 <proof>

**lemma** *match2-1-oorI2*:  
 $match2-1 \Delta_2 s1 s2 s2' statA sv1 sv2 statO \implies$   
 $match2-1 (oor \Delta \Delta_2) s1 s2 s2' statA sv1 sv2 statO$   
 <proof>

**lemma** *match2-oorI1*:  
 $match2 \Delta s1 s2 statA sv1 sv2 statO \implies$   
 $match2 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
 <proof>

**lemma** *match2-oorI2*:  
 $match2 \Delta_2 s1 s2 statA sv1 sv2 statO \implies$   
 $match2 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
 <proof>

**lemma** *match12-oorI1*:  
 $match12 \Delta s1 s2 statA sv1 sv2 statO \implies$   
 $match12 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
 <proof>

**lemma** *match12-oorI2*:  
 $match12 \Delta_2 s1 s2 statA sv1 sv2 statO \implies$   
 $match12 (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
 <proof>

**lemma** *match12-1-oorI1*:  
 $match12-1 \Delta s1' s2' statA' sv1 sv2 statO \implies$   
 $match12-1 (oor \Delta \Delta_2) s1' s2' statA' sv1 sv2 statO$   
 <proof>

**lemma** *match12-1-oorI2*:  
 $match12-1 \Delta_2 s1' s2' statA' sv1 sv2 statO \implies$   
 $match12-1 (oor \Delta \Delta_2) s1' s2' statA' sv1 sv2 statO$   
 <proof>

**lemma** *match12-2-oorI1*:

$match12-2 \Delta s2 s2' statA' sv1 sv2 statO \implies$   
 $match12-2 (oor \Delta \Delta_2) s2 s2' statA' sv1 sv2 statO$   
 <proof>

**lemma** *match12-2-oorI2*:  
 $match12-2 \Delta_2 s2 s2' statA' sv1 sv2 statO \implies$   
 $match12-2 (oor \Delta \Delta_2) s2 s2' statA' sv1 sv2 statO$   
 <proof>

**lemma** *match12-12-oorI1*:  
 $match12-12 \Delta s1' s2' statA' sv1 sv2 statO \implies$   
 $match12-12 (oor \Delta \Delta_2) s1' s2' statA' sv1 sv2 statO$   
 <proof>

**lemma** *match12-12-oorI2*:  
 $match12-12 \Delta_2 s1' s2' statA' sv1 sv2 statO \implies$   
 $match12-12 (oor \Delta \Delta_2) s1' s2' statA' sv1 sv2 statO$   
 <proof>

**lemma** *match-oorI1*:  
 $react \Delta s1 s2 statA sv1 sv2 statO \implies$   
 $react (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
 <proof>

**lemma** *match-oorI2*:  
 $react \Delta_2 s1 s2 statA sv1 sv2 statO \implies$   
 $react (oor \Delta \Delta_2) s1 s2 statA sv1 sv2 statO$   
 <proof>

**lemma** *proact-oorI1*:  
 $proact \Delta meas s1 s2 statA sv1 sv2 statO \implies$   
 $proact (oor \Delta \Delta_2) meas s1 s2 statA sv1 sv2 statO$   
 <proof>

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

**lemma** *move-1-oorI1*:  
 $move-1 \Delta meas s1 s2 statA sv1 sv2 statO \implies$   
 $move-1 (oor \Delta \Delta_2) meas s1 s2 statA sv1 sv2 statO$   
 <proof>

**lemma** *move-1-oorI2*:

*move-1*  $\Delta_2$  *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
*move-1* (*oor*  $\Delta\ \Delta_2$ ) *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO$   
 ⟨*proof*⟩

**lemma** *move-2-oorI1*:  
*move-2*  $\Delta$  *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
*move-2* (*oor*  $\Delta\ \Delta_2$ ) *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO$   
 ⟨*proof*⟩

**lemma** *move-2-oorI2*:  
*move-2*  $\Delta_2$  *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
*move-2* (*oor*  $\Delta\ \Delta_2$ ) *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO$   
 ⟨*proof*⟩

**lemma** *move-12-oorI1*:  
*move-12*  $\Delta$  *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
*move-12* (*oor*  $\Delta\ \Delta_2$ ) *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO$   
 ⟨*proof*⟩

**lemma** *move-12-oorI2*:  
*move-12*  $\Delta_2$  *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
*move-12* (*oor*  $\Delta\ \Delta_2$ ) *meas*  $s1\ s2\ statA\ sv1\ sv2\ statO$   
 ⟨*proof*⟩

**end**

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

**lemma** *match1-1-defD*: *match1-1*  $\Delta\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \longleftrightarrow$   
 $\neg\ finalV\ sv1 \wedge \Delta \infty\ s1'\ s2\ statA\ (nextO\ sv1)\ sv2\ statO$   
 ⟨*proof*⟩

**lemma** *match1-12-defD*: *match1-12*  $\Delta\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \longleftrightarrow$   
 $\neg\ finalV\ sv1 \wedge \neg\ finalV\ sv2 \wedge$   
 $\Delta \infty\ s1'\ s2\ statA\ (nextO\ sv1)\ (nextO\ sv2)\ (sstatO'\ statO\ sv1\ sv2)$   
 ⟨*proof*⟩

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

**lemma** *match2-1-defD*: *match2-1*  $\Delta\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO \longleftrightarrow$   
 $\neg\ finalV\ sv2 \wedge \Delta \infty\ s1\ s2'\ statA\ sv1\ (nextO\ sv2)\ statO$   
 ⟨*proof*⟩

**lemma** *match2-12-defD*: *match2-12*  $\Delta\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO \longleftrightarrow$   
 $\neg\ finalV\ sv1 \wedge \neg\ finalV\ sv2 \wedge \Delta \infty\ s1\ s2'\ statA\ (nextO\ sv1)\ (nextO\ sv2)\ (sstatO'$

*statO sv1 sv2*)  
{proof}

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

**lemma** *match12-1-defD*: *match12-1 Δ s1' s2' statA' sv1 sv2 statO ↔*  
*¬ finalV sv1 ∧ Δ ∞ s1' s2' statA' (nextO sv1) sv2 statO*  
{proof}

**lemma** *match12-2-defD*: *match12-2 Δ s1' s2' statA' sv1 sv2 statO ↔*  
*¬ finalV sv2 ∧ Δ ∞ s1' s2' statA' sv1 (nextO sv2) statO*  
{proof}

**lemma** *match12-12-defD*: *match12-12 Δ s1' s2' statA' sv1 sv2 statO ↔*  
*(let statO' = sstatO' statO sv1 sv2 in*  
*¬ finalV sv1 ∧ ¬ finalV sv2 ∧*  
*(statA' = Diff → statO' = Diff) ∧*  
*Δ ∞ s1' s2' statA' (nextO sv1) (nextO sv2) statO')*  
{proof}

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

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

**lemma** *move-1-defD*: *move-1 Δ w s1 s2 statA sv1 sv2 statO ↔*  
*¬ finalV sv1 ∧ Δ w s1 s2 statA (nextO sv1) sv2 statO*  
{proof}

**lemma** *move-2-defD*: *move-2 Δ w s1 s2 statA sv1 sv2 statO ↔*  
*¬ finalV sv2 ∧ Δ w s1 s2 statA sv1 (nextO sv2) statO*  
{proof}

**lemma** *move-12-defD*: *move-12 Δ w s1 s2 statA sv1 sv2 statO ↔*  
*(let statO' = sstatO' statO sv1 sv2 in*  
*¬ finalV sv1 ∧ ¬ finalV sv2 ∧*  
*Δ w s1 s2 statA (nextO sv1) (nextO sv2) statO')*  
{proof}

**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  $\langle \text{newStat Eq } (P, R) (Q, S) = \text{Eq} \rangle$ 
   $\langle \text{proof} \rangle$ 
```

```
lemma newStat-diff: newStat stat r r = Diff  $\implies$  stat = Diff
   $\langle \text{proof} \rangle$ 
```

```
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
```

```
   $\longrightarrow$ 
```

```
( $\exists sv1 sv2. \text{istateV } sv1 \wedge \text{istateV } sv2 \wedge \text{corrState } sv1 s1 \wedge \text{corrState } sv2 s2$ 
 $\wedge \Delta \infty \infty \infty s1 s2 \text{Eq } sv1 sv2 \text{Eq}$ )
```

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

**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**  $match1-1-mono:$   
 $\Delta \leq \Delta' \implies match1-1 \Delta w1 w2 s1 s1' s2 statA sv1 sv2 statO \implies$   
 $match1-1 \Delta' w1 w2 s1 s1' s2 statA sv1 sv2 statO$   
 $\langle proof \rangle$

**lemma**  $match1-12-mono:$   
 $\Delta \leq \Delta' \implies match1-12 \Delta w1 w2 s1 s1' s2 statA sv1 sv2 statO \implies$   
 $match1-12 \Delta' w1 w2 s1 s1' s2 statA sv1 sv2 statO$   
 $\langle proof \rangle$

**lemma**  $match1-mono:$   
**assumes**  $\Delta \leq \Delta'$   
**shows**  $match1 \Delta w1 w2 s1 s2 statA sv1 sv2 statO \implies match1 \Delta' w1 w2 s1 s2$   
 $statA sv1 sv2 statO$   
 $\langle proof \rangle$

**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**  $match2-12 \Delta w1 w2 s1 s2 s2' statA sv1 sv2 statO \equiv$   
 $\exists sv1' sv2'.$   
 $let statO' = sstatO' statO sv1 sv2 in$

$$\begin{aligned} & \text{validTransV } (sv1, sv1') \wedge \\ & \text{validTransV } (sv2, sv2') \wedge \\ & \Delta \infty w1 \ w2 \ s1 \ s2' \ \text{statA } sv1' \ sv2' \ \text{statO}' \end{aligned}$$

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

$$\begin{aligned} & \neg \text{isIntO } s2 \longrightarrow \\ & (\forall s2'. \text{validTransO } (s2, s2')) \\ & \longrightarrow \\ & (\exists w1' < w1. \exists w2' < w2. \neg \text{isSecO } s2 \wedge \Delta \infty w1' \ w2' \ s1 \ s2' \ \text{statA } sv1 \ sv2 \\ & \text{statO}) \vee \\ & (\exists w1' < w1. \text{eqSec } sv2 \ s2 \wedge \neg \text{isIntV } sv2 \wedge \text{match2-1 } \Delta \ w1' \ \infty \ s1 \ s2 \ s2' \ \text{statA} \\ & sv1 \ sv2 \ \text{statO}) \vee \\ & (\neg \text{isSecV } sv1 \wedge \text{eqSec } sv2 \ s2 \wedge \text{Van.eqAct } sv1 \ sv2 \wedge \text{match2-12 } \Delta \ \infty \ \infty \ s1 \\ & s2 \ s2' \ \text{statA } sv1 \ sv2 \ \text{statO})) \end{aligned}$$

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

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

$$\Delta \leq \Delta' \implies \text{match2-1 } \Delta \ w1 \ w2 \ s1 \ s1' \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO} \implies \text{match2-1 } \Delta' \ w1 \ w2 \ s1 \ s1' \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO}$$

*<proof>*

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

$$\Delta \leq \Delta' \implies \text{match2-12 } \Delta \ w1 \ w2 \ s1 \ s1' \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO} \implies \text{match2-12 } \Delta' \ w1 \ w2 \ s1 \ s1' \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO}$$

*<proof>*

**lemma** *match2-mono*:

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

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

*<proof>*

**definition** *match12-1*  $\Delta \ w1 \ w2 \ s1' \ s2' \ \text{statA}' \ sv1 \ sv2 \ \text{statO} \equiv$

$$\begin{aligned} & \exists sv1'. \text{validTransV } (sv1, sv1') \wedge \\ & \Delta \infty w1 \ w2 \ s1' \ s2' \ \text{statA}' \ sv1' \ sv2 \ \text{statO} \end{aligned}$$

**definition** *match12-2*  $\Delta \ w1 \ w2 \ s1' \ s2' \ \text{statA}' \ sv1 \ sv2 \ \text{statO} \equiv$

$$\begin{aligned} & \exists sv2'. \text{validTransV } (sv2, sv2') \wedge \\ & \Delta \infty w1 \ w2 \ s1' \ s2' \ \text{statA}' \ sv1 \ sv2' \ \text{statO} \end{aligned}$$

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

$$\begin{aligned} & \exists sv1' \ sv2'. \\ & \text{let } \text{statO}' = \text{sstatO}' \ \text{statO } sv1 \ sv2 \ \text{in} \\ & \text{validTransV } (sv1, sv1') \wedge \\ & \text{validTransV } (sv2, sv2') \wedge \\ & (\text{statA}' = \text{Diff} \longrightarrow \text{statO}' = \text{Diff}) \wedge \end{aligned}$$

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

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

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

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

**lemma** *match12-simpleI*:

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

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

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

*<proof>*

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

$$\begin{aligned} & \Delta \leq \Delta' \implies \text{match12-1 } \Delta w1 w2 s1' s2' statA' sv1 sv2 statO \implies \text{match12-1 } \Delta' \\ & w1 w2 s1' s2' statA' sv1 sv2 statO \\ & \text{<proof>} \end{aligned}$$

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

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

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

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

**lemma** *match12-mono*:

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

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

**definition** *react*  $\Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \equiv$   
 $\text{match1 } \Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$   
 $\wedge$   
 $\text{match2 } \Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$   
 $\wedge$   
 $\text{match12 } \Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$

**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**  $\text{react } \Delta \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \implies \text{react } \Delta' \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}$   
 ⟨proof⟩

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

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

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

**lemmas**  $\text{proact-defs} = \text{proact-def move-1-def move-2-def move-12-def}$

**lemma**  $\text{move-1-mono}$ :

$\Delta \leq \Delta' \implies \text{move-1 } \Delta w w1 w2 s1 s2 \text{ statA sv1 sv2 statO} \implies \text{move-1 } \Delta' w w1$   
 $w2 s1 s2 \text{ statA sv1 sv2 statO}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{move-2-mono}$ :

$\Delta \leq \Delta' \implies \text{move-2 } \Delta w w1 w2 s1 s2 \text{ statA sv1 sv2 statO} \implies \text{move-2 } \Delta' w w1$   
 $w2 s1 s2 \text{ statA sv1 sv2 statO}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{move-12-mono}$ :

$\Delta \leq \Delta' \implies \text{move-12 } \Delta w w1 w2 s1 s2 \text{ statA sv1 sv2 statO} \implies \text{move-12 } \Delta' w w1$   
 $w2 s1 s2 \text{ statA sv1 sv2 statO}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{proact-mono}$ :

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

**shows**  $\text{proact } \Delta w w1 w2 s1 s2 \text{ statA sv1 sv2 statO} \implies \text{proact } \Delta' w w1 w2 s1 s2$   
 $\text{statA sv1 sv2 statO}$   
 $\langle \text{proof} \rangle$

## 4.2 The definition of unwinding

**definition**  $\text{unwindCond} ::$

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

**where**

$\text{unwindCond } \Delta \equiv \forall w w1 w2 s1 s2 \text{ statA sv1 sv2 statO.}$

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

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

$\longrightarrow$

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

$\wedge$

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

$\wedge$

$((\exists v < w. \text{proact } \Delta v w1 w2 s1 s2 \text{ statA sv1 sv2 statO})$

$\vee$

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

)

**lemma** *unwindCond-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**

$\bigwedge w w1 w2 s1 s2 statA sv1 sv2 statO.$

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

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies statA = Eq$

$\implies$

$isIntO s1 \longleftrightarrow isIntO s2$

**and**

$\bigwedge w w1 w2 s1 s2 statA sv1 sv2 statO.$

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

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

$\implies$

$react \Delta w1 w2 s1 s2 statA sv1 sv2 statO$

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

$\langle proof \rangle$

### 4.3 The soundness of unwinding

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

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

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

$Van.validFromS sv1 trv1 \wedge$

$Van.validFromS sv2 trv2 \wedge$

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

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

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

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

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

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

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

$\langle proof \rangle$

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

$\langle proof \rangle$

**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*

$\langle proof \rangle$

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

$\langle proof \rangle$

**lemma** *unwindCond-final*:

$unwindCond\ \Delta \implies reachO\ s1 \implies reachO\ s2 \implies reachV\ sv1 \implies reachV\ sv2 \implies$   
 $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \implies$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

**lemma**  $\varphi$ -completedFrom:  $unwindCond \Delta \implies$

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

$Opt.validFromS s1 tr1 \implies completedFromO s1 tr1 \implies$

$Opt.validFromS s2 tr2 \implies completedFromO s2 tr2 \implies$

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

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

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

$(length trv1 > Suc 0 \vee w1' < w1) \wedge (length trv2 > Suc 0 \vee w2' < w2) \wedge$

$Van.validFromS sv1 trv1 \wedge$

$Van.validFromS sv2 trv2 \wedge$

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

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

$(statO = Eq \longrightarrow (statOO = Diff \longleftrightarrow Van.O trv1 \neq Van.O trv2)) \wedge$

$(statA = Eq \longrightarrow (statAA = Diff \longleftrightarrow Opt.O tr1 \neq Opt.O tr2)) \wedge$

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

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

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

**lemma** *unwindCond-ex- $\varphi$ a-getActO*:

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

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

**and**  $r34$ : *reachO*  $s1 \text{ reachO } s2$  **and**  $r12$ : *reachV*  $sv1 \text{ reachV } sv2$

**and** *stat*: (*statA* = *Diff*  $\longrightarrow$  *statO* = *Diff*)

**and**  $v$ : *validTransO* ( $s1, s1'$ ) *validTransO* ( $s2, s2'$ )

**and**  $i34$ : *isIntO*  $s1 \text{ isIntO } s2$  *getActO*  $s1 = \text{getActO } s2$

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

$\varphi_a \Delta \infty w1 w2 w1' w2' \text{ statA } s1 [s1, s1'] s2 [s2, s2'] (\text{sstatA}' \text{ statA } s1 s2)$   
 $\text{statO } sv1 \text{ trv1 } sv2 \text{ trv2 } \text{statOO}$

*<proof>*

**lemma** *unwindCond-ex- $\varphi$ a'-aux*:

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

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

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

**and** *stat*: (*statA* = *Diff*  $\longrightarrow$  *statO* = *Diff*)

**and**  $tr14NE$ :  $tr1 \neq [] \text{ } tr2 \neq []$

**and**  $v3'$ : *Opt.validFromS*  $s1 (tr1 \text{ ## } s1')$  **and**  $v4'$ : *Opt.validFromS*  $s2 (tr2 \text{ ## } s2')$

**and**  $i$ : *isIntO* ( $\text{lastt } s1 \text{ } tr1$ ) *isIntO* ( $\text{lastt } s2 \text{ } tr2$ )

**and**  $A34$ : *getActO* ( $\text{lastt } s1 \text{ } tr1$ ) = *getActO* ( $\text{lastt } s2 \text{ } tr2$ )

**and**  $nev$ : *never isIntO* (*butlast*  $tr1$ ) *never isIntO* (*butlast*  $tr2$ )

**shows**  $\exists w1' w2' \text{ trv1}' \text{ } \text{trv2}' \text{ statAA}' \text{ } \text{statOO}'$ .

$\varphi_a \Delta \infty w1 w2 w1' w2' \text{ statA } s1 (tr1 \text{ ## } s1') s2 (tr2 \text{ ## } s2') \text{ statAA}' \text{ } \text{statO}$   
 $sv1 \text{ trv1}' sv2 \text{ trv2}' \text{ statOO}'$

*<proof>*

**lemma** *unwindCond-ex- $\varphi$ a-aux2*:

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

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

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

**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*  $s1' \text{ isIntO } s2'$

**and**  $A34$ : *getActO*  $s1' = \text{getActO } s2'$

**and**  $nev$ : *never isIntO*  $tr1$  *never isIntO*  $tr2$

**shows**  $\exists w1' w2' \text{ trv1 } \text{trv2 } \text{statAA } \text{statOO}$ .

$\varphi_a \Delta \infty w1 w2 w1' w2' \text{ statA } s1 (tr1 @ [s1',s1'']) s2 (tr2 @ [s2',s2'']) \text{ statAA}$   
 $\text{statO } sv1 \text{ trv1 } sv2 \text{ trv2 } \text{statOO}$

*<proof>*

**lemma** *lastt-snoc[simp]*: *lastt*  $s1 (tr1 @ [s1'']) = s1''$

*<proof>*

**lemma** *lastt-snoc2[simp]*:  $\text{lastt } s1 \ (tr1 \ @ \ [s1', s1'']) = s1''$   
 ⟨proof⟩

**lemma** *append-snoc2*:  $tr1 \ @ \ [s1', s1''] = (tr1 \ ## \ s1') \ ## \ s1''$   
 ⟨proof⟩

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

**proposition** *unwindCond-ex-φ'*:

**assumes** *unwind*:  $\text{unwindCond } \Delta$  **and**  $\Delta$ :  $\Delta \ w \ w1 \ w2 \ s1 \ s2 \ \text{statA } sv1 \ sv2 \ \text{statO}$   
**and** *r*:  $\text{reachO } s1 \ \text{reachO } s2 \ \text{reachV } sv1 \ \text{reachV } sv2$   
**and** *stat*:  $\text{statA} = \text{Diff} \longrightarrow \text{statO} = \text{Diff}$   
**and** *v3'*:  $\text{Opt.validFromS } s1 \ ((tr1 \ ## \ s1') \ ## \ s1'')$  **and** *v4'*:  $\text{Opt.validFromS } s2 \ ((tr2 \ ## \ s2') \ ## \ s2'')$   
**and** *i*:  $\text{isIntO } s1' \ \text{isIntO } s2'$   
**and** *A34*:  $\text{getActO } s1' = \text{getActO } s2'$   
**and** *nev*:  $\text{never isIntO } tr1 \ \text{never isIntO } tr2$   
**shows**  $\exists \ w1' \ w2' \ trv1 \ sv1'' \ trv2 \ sv2'' \ \text{statAA } \text{statOO}$ .  
 $\varphi' \Delta \ w1 \ w2 \ w1' \ w2' \ \text{statA } s1 \ tr1 \ s1' \ s1'' \ s2 \ tr2 \ s2' \ s2'' \ \text{statAA } \text{statO } sv1 \ trv1$   
 $sv1'' \ sv2 \ trv2 \ sv2'' \ \text{statOO}$   
 ⟨proof⟩

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

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

**assumes**  $unw: unwindCond \Delta$   
**and**  $r: reachO s1 reachO s2 reachV sv1 reachV sv2$   
**and**  $utr1: Opt.validFromS s1 tr1$   
**and**  $\chi3: \chi3 \Delta w w1 w2 w1' w2' s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO$   
**shows**  $(finalV (lastt sv1 trv1) \longleftrightarrow finalO (lastt s1 tr1)) \wedge (finalV (lastt sv2 trv2) \longleftrightarrow finalO s2)$   
 <proof>

**lemma**  $\chi3-completedFrom: unwindCond \Delta \implies$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $Opt.validFromS s1 tr1 \implies completedFromO s1 tr1 \implies$   
 $\chi3 \Delta w w1 w2 w1' w2' s1 tr1 s2 statAA sv1 trv1 sv2 trv2 statOO$   
 $\implies completedFromV sv1 trv1 \wedge completedFromV sv2 trv2$   
 <proof>

**lemma**  $unwindCond-ex-\chi3:$   
**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**  $utr1: Opt.validFromS s1 tr1$   
**and**  $nis1: \neg isIntO s1$  **and**  $nis2: \neg isIntO s2$   
**and**  $inter3: never isIntO tr1$   
**and**  $sec: never isSecO (butlast tr1) isSecO (lastt s1 tr1)$   
**shows**  $\exists w' w1' w2' trv1 trv2 statOO. \chi3 \Delta w' w1 w2 w1' w2' s1 tr1 s2 statA sv1 trv1 sv2 trv2 statOO$   
 <proof>

**definition**  $\chi3a$  **where**  $\chi3a \Delta w (w1::enat) w2 w1' w2' s1 s1' s2 statAA sv1 trv1 sv2 trv2 statOO \equiv$   
 $trv1 \neq [] \wedge trv2 \neq [] \wedge (length trv2 > Suc 0 \vee w2' < w2) \wedge$   
 $Van.validFromS sv1 trv1 \wedge Van.validFromS sv2 trv2 \wedge$   
 $Van.S trv1 = [getSecO s1] \wedge$   
 $never isSecV (butlast trv2) \wedge$   
 $Van.A trv1 = Van.A trv2 \wedge$   
 $\Delta w w1' w2' s1' s2 statAA (lastt sv1 trv1) (lastt sv2 trv2) statOO$

**lemma**  $unwindCond-ex-\chi3a-getSec:$   
**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**  $v: validTransO (s1, s1')$   
**and**  $ii3: \neg isIntO s1$   
**and**  $is1: isSecO s1$  **and**  $isv13: isSecV sv1 getSecO s1 = getSecV sv1$   
**shows**  $\exists w1' w2' trv1 trv2 statOO.$   
 $\chi3a \Delta \infty w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv2 trv2 statOO$   
 <proof>

**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>*

**lemma** *unwindCond-ex- $\chi^3b$ -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**  $\exists w1'\ w2'\ trv1\ trv2\ statOO. \chi^3b \Delta \infty w1\ w2\ w1'\ w2'\ s1\ (tr1 @ [s1',s1'])$   
 $s2\ statA\ sv1\ trv1\ sv2\ trv2\ statOO$   
*<proof>*

**definition**  $\chi^3' \Delta w1\ w2\ w1'\ w2'\ s1\ tr1\ s1'\ s1''\ s2\ statAA\ sv1\ trv1\ sv1''\ sv2\ trv2$   
 $sv2''\ statOO \equiv$   
 $Van.validFromS\ sv1\ (trv1 \## sv1'') \wedge Van.validFromS\ sv2\ (trv2 \## sv2'') \wedge$   
 $Van.S\ (trv1 \## sv1'') = Opt.S\ ((tr1 \## s1') \## s1'') \wedge never\ isSecV\ trv2 \wedge$   
 $Van.A\ (trv1 \## sv1'') = Van.A\ (trv2 \## sv2'') \wedge$   
 $trv1 \neq [] \wedge (trv2 \neq [] \vee w2' < w2) \wedge$   
 $\Delta \infty w1'\ w2'\ s1''\ s2\ statAA\ sv1''\ sv2''\ statOO$

**proposition** *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** *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**  $\exists w1' w2' trv1 sv1'' trv2 sv2'' statOO. \chi3' \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO$   
*<proof>*

**definition**  $\omega3 \Delta w1 w2 w1' w2' s1 s1' s2 statAA sv1 trv1 sv1' sv2 trv2 sv2' statOO \equiv$   
*Van.validFromS sv1 (trv1 ## sv1')  $\wedge$  Van.validFromS sv2 (trv2 ## sv2')  $\wedge$*   
*never isSecV trv1  $\wedge$  never isSecV trv2  $\wedge$*   
*Van.A (trv1 ## sv1') = Van.A (trv2 ## sv2')  $\wedge$*   
*(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- $\omega3$* :  
**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**  $\exists w1' w2' trv1 sv1' trv2 sv2' statOO. \omega3 \Delta w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO$   
*<proof>*

**definition**  $\chi4 \Delta w w1 (w2::enat) w1' w2' s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO \equiv$   
*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**  *$\chi4$ -final*:  
**assumes** *unw*: *unwindCond  $\Delta$*   
**and** *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*  
**and** *vtr2*: *Opt.validFromS s2 tr2*  
**and**  $\chi4$ :  $\chi4 \Delta w w1 w2 w1' w2' s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO$   
**shows** (*finalV (lastt sv1 trv1)*  $\longleftrightarrow$  *finalO s1*)  $\wedge$  (*finalV (lastt sv2 trv2)*  $\longleftrightarrow$  *finalO (lastt s2 tr2)*)

*<proof>*

**lemma**  $\chi_4$ -completedFrom:  $unwindCond \Delta \implies$   
 $reachO s1 \implies reachO s2 \implies reachV sv1 \implies reachV sv2 \implies$   
 $Opt.validFromS s2 tr2 \implies completedFromO s2 tr2 \implies$   
 $\chi_4 \Delta w w1 w2 w1' w2' s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO$   
 $\implies completedFromV sv1 trv1 \wedge completedFromV sv2 trv2$   
*<proof>*

**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**  $\exists w' w1' w2' trv1 trv2 statOO. \chi_4 \Delta w' w1 w2 w1' w2' s1 s2 tr2 statA sv1$   
 $trv1 sv2 trv2 statOO$   
*<proof>*

**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**  $r3_4$ :  $reachO s1 reachO s2$  **and**  $r1_2$ :  $reachV sv1 reachV sv2$   
**and**  $v$ :  $validTransO (s2, s2')$   
**and**  $ii_4$ :  $\neg isIntO s2$   
**and**  $is2$ :  $isSecO s2$  **and**  $isv2_4$ :  $isSecV sv2 getSecO s2 = getSecV sv2$   
**shows**  $\exists w1' w2' trv1 trv2 statOO.$   
 $\chi_{4a} \Delta \infty w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv2 trv2 statOO$   
*<proof>*

**definition**  $\chi_{4b} \Delta w w1 w2 w1' (w2'::enat) s1 s2 tr2 statAA sv1 trv1 sv2 trv2$   
 $statOO \equiv$   
 $trv1 \neq [] \wedge trv2 \neq [] \wedge (length trv1 > Suc 0 \vee w1' < w1) \wedge$   
 $Van.validFromS sv1 trv1 \wedge Van.validFromS sv2 trv2 \wedge$   
 $never isSecV (butlast trv1) \wedge$   
 $Van.S trv2 = Opt.S tr2 \wedge$   
 $Van.A trv1 = Van.A trv2 \wedge$   
 $\Delta w w1' w2' s1 (lastt s2 tr2) statAA (lastt sv1 trv1) (lastt sv2 trv2) statOO$

**lemma** *unwindCond-ex- $\chi_4^b$ -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** *tr2NE*: *tr2*  $\neq$   $\square$   
**and** *v4'*: *Opt.validFromS s2 (tr2 ## s2')*  
**and** *nis1*:  $\neg$  *isIntO s1* **and** *nis2*:  $\neg$  *isIntO s2*  
**and** *ninter4'*: *never isIntO (tr2 ## s2')*  
**and** *sec*: *never isSecO (butlast tr2) isSecO (lastt s2 tr2)*  
**shows**  $\exists w1' w2' trv1 trv2 statOO. \chi_4^b \Delta \infty w1 w2 w1' w2' s1 s2 (tr2 ## s2')$   
*statA sv1 trv1 sv2 trv2 statOO*  
*<proof>*

**lemma** *unwindCond-ex- $\chi_4^b$ -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** *v4'*: *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**  $\exists w1' w2' trv1 trv2 statOO. \chi_4^b \Delta \infty w1 w2 w1' w2' s1 s2 (tr2 @ [s2',s2'])$   
*statA sv1 trv1 sv2 trv2 statOO*  
*<proof>*

**definition**  $\chi_4' \Delta w1 w2 w1' (w2'::enat) s1 s2 tr2 s2' s2'' statAA sv1 trv1 sv1''$   
 $sv2 trv2 sv2'' statOO \equiv$   
*Van.validFromS sv1 (trv1 ## sv1'')  $\wedge$  Van.validFromS sv2 (trv2 ## sv2'')  $\wedge$*   
*never isSecV (butlast (trv1 ## sv1''))  $\wedge$*   
*Van.S (trv2 ## sv2'') = Opt.S ((tr2 ## s2') ## s2'')  $\wedge$*   
*Van.A (trv1 ## sv1'') = Van.A (trv2 ## sv2'')  $\wedge$*   
*trv2  $\neq$   $\square$   $\wedge$  (trv1  $\neq$   $\square$   $\vee$   $w1' < w1$ )  $\wedge$*   
 $\Delta \infty w1' w2' s1 s2'' statAA sv1'' sv2'' statOO$

**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** *v4'*: *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**  $\exists w1' w2' trv1 sv1'' trv2 sv2'' statOO. \chi_4' \Delta w1 w2 w1' w2' s1 s2 tr2 s2'$   
 $s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO$   
*<proof>*

**definition**  $\omega_4 \Delta w_1 w_2 w_1' (w_2'::\text{enat}) s_1 s_2 s_2' \text{statAA } sv_1 \text{trv1 } sv_1' sv_2 \text{trv2 } sv_2' \text{statOO} \equiv$

$\text{Van.validFromS } sv_1 (\text{trv1} \#\# sv_1') \wedge \text{Van.validFromS } sv_2 (\text{trv2} \#\# sv_2') \wedge$   
 $\text{never isSecV } \text{trv1} \wedge \text{never isSecV } \text{trv2} \wedge$   
 $\text{Van.A } (\text{trv1} \#\# sv_1') = \text{Van.A } (\text{trv2} \#\# sv_2') \wedge$   
 $(\text{trv1} \neq \square \vee w_1' < w_1) \wedge (\text{trv2} \neq \square \vee w_2' < w_2) \wedge$   
 $\Delta \infty w_1' w_2' s_1 s_2' \text{statAA } sv_1' sv_2' \text{statOO}$

**proposition** *unwindCond-ex- $\omega_4$* :

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

**and**  $\Delta$ :  $\Delta w_1 w_2 s_1 s_2 \text{statA } sv_1 sv_2 \text{statO}$

**and**  $r3_4$ : *reachO*  $s_1 \text{reachO } s_2$  **and**  $r1_2$ : *reachV*  $sv_1 \text{reachV } sv_2$

**and**  $nis_1$ :  $\neg \text{isIntO } s_1$

**and**  $v_4$ : *validTransO*  $(s_2, s_2')$

**and**  $nis_2$ :  $\neg \text{isIntO } s_2 \neg \text{isIntO } s_2' \neg \text{isSecO } s_2$

**shows**  $\exists w_1' w_2' \text{trv1 } sv_1' \text{trv2 } sv_2' \text{statOO}. \omega_4 \Delta w_1 w_2 w_1' w_2' s_1 s_2 s_2' \text{statA } sv_1 \text{trv1 } sv_1' sv_2 \text{trv2 } sv_2' \text{statOO}$

*<proof>*

**definition**  $\varphi\varphi s_1 \text{ltr1 } s_2 \text{ltr2 } tr_1 s_1' s_1'' \text{ltr1}' tr_2 s_2' s_2'' \text{ltr2}' \equiv$

$\text{ltr1} = \text{lappend } (\text{llist-of } (\text{tr1} \#\# s_1')) (s_1'' \$ \text{ltr1}') \wedge$

$\text{ltr2} = \text{lappend } (\text{llist-of } (\text{tr2} \#\# s_2')) (s_2'' \$ \text{ltr2}') \wedge$

$\text{Opt.validFromS } s_1 ((\text{tr1} \#\# s_1') \#\# s_1'') \wedge \text{Opt.validFromS } s_2 ((\text{tr2} \#\# s_2') \#\# s_2'') \wedge$

$\text{never isIntO } tr_1 \wedge \text{never isIntO } tr_2 \wedge$

$\text{isIntO } s_1' \wedge \text{isIntO } s_2' \wedge \text{getActO } s_1' = \text{getActO } s_2' \wedge$

$\text{Opt.lvalidFromS } s_1'' (s_1'' \$ \text{ltr1}') \wedge \text{Opt.lcompletedFrom } s_1'' (s_1'' \$ \text{ltr1}') \wedge$

$\text{Opt.lvalidFromS } s_2'' (s_2'' \$ \text{ltr2}') \wedge \text{Opt.lcompletedFrom } s_2'' (s_2'' \$ \text{ltr2}') \wedge$

$\text{Opt.lA } (s_1'' \$ \text{ltr1}') = \text{Opt.lA } (s_2'' \$ \text{ltr2}')$

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

**assumes**  $vltr_1$ : *Opt.lvalidFromS*  $s_1 \text{ltr1 } \text{Opt.lcompletedFrom } s_1 \text{ltr1}$

**and**  $vltr_2$ : *Opt.lvalidFromS*  $s_2 \text{ltr2 } \text{Opt.lcompletedFrom } s_2 \text{ltr2}$

**and**  $A$ : *Opt.lA*  $\text{ltr1} = \text{Opt.lA } \text{ltr2}$  **and**  $\text{inter3}$ :  $\neg \text{never isIntO } \text{ltr1}$

**shows**  $\exists tr_1 s_1' s_1'' \text{ltr1}' tr_2 s_2' s_2'' \text{ltr2}' . \varphi\varphi s_1 \text{ltr1 } s_2 \text{ltr2 } tr_1 s_1' s_1'' \text{ltr1}' tr_2 s_2' s_2'' \text{ltr2}'$

*<proof>*

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

**lemma**  $isSecO$ - $\chi\chi$ :

**assumes**  $vltr1$ :  $Opt.lvalidFromS$   $s1$   $ltr1$   $Opt.lcompletedFrom$   $s1$   $ltr1$

**and**  $inter$ :  $lnever$   $isIntO$   $ltr1$  **and**  $isec$ :  $\neg$   $lnever$   $isSecO$   $ltr1$

**shows**  $\exists$   $tr1$   $s1'$   $s1''$   $ltr1'$ .  $\chi\chi$   $s1$   $ltr1$   $tr1$   $s1'$   $s1''$   $ltr1'$

*<proof>*

**type-synonym** ( $'stA$ ,  $'stO$ )  $tuple34 =$

$enat \times enat \times$   
 $'stA \times 'stA$   $l$ list  $\times$   
 $'stA \times 'stA$   $l$ list  $\times$   
 $status \times$   
 $'stO \times 'stO \times status$

**type-synonym** ( $'stA$ ,  $'stO$ )  $tuple12 =$

$'stO$   $list \times 'stO \times 'stO$   $list \times 'stO \times status \times status$

**context**

**fixes**  $\Delta$  ::  $enat \Rightarrow enat \Rightarrow enat \Rightarrow 'stateO \Rightarrow 'stateO \Rightarrow status \Rightarrow 'stateV \Rightarrow 'stateV \Rightarrow status \Rightarrow bool$

**begin**

**fun**  $isn$  ::  $turn \times ('stateO, 'stateV) tuple34 \Rightarrow bool$

**where**

$isn$  ( $trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO$ )  $\longleftrightarrow$   $ltr1 = [] \wedge ltr2 = []$

**fun**  $h-t$  ::

$turn \times ('stateO, 'stateV) tuple34 \Rightarrow$

$('stateO, 'stateV) tuple12 \times$

$turn \times ('stateO, 'stateV) tuple34$

**where**

$h-t$  ( $trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO$ ) =

(if  $trn = L$

then if  $lnever$   $isSecO$   $ltr1$

then let ( $s1', ltr1'$ ) = ( $lhd$  ( $l$ tl  $ltr1$ ),  $l$ tl  $ltr1$ )

in let ( $w1', w2', trv1, sv1', trv2, sv2', statOO$ ) =

( $SOME$   $k$ . case  $k$  of ( $w1', w2', trv1, sv1', trv2, sv2', statOO$ )  $\Rightarrow$

```

      ω3 Δ w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO)
in ((trv1,sv1',trv2,sv2',statA,statOO),
  (if trv1 = [] then L else R,
    w1',w2',s1',ltr1',s2,ltr2,statA,sv1',sv2',statOO))
else
let (tr1,s1',s1'',ltr1') =
  (SOME k. case k of (tr1,s1',s1'',ltr1') ⇒
    χχ s1 ltr1 tr1 s1' s1'' ltr1')
in let (w1',w2',trv1,sv1'',trv2,sv2'',statOO) =
  (SOME k'. case k' of (w1',w2',trv1,sv1'',trv2,sv2'',statOO) ⇒
    χ3' Δ w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2
sv2'' statOO)
  in ((trv1,sv1'',trv2,sv2'',statA,statOO),
    (R,w1',w2',s1'',s1'' $ ltr1',s2,ltr2,statA,sv1'',sv2'',statOO))
—
else if lnever isSecO ltr2
then let (s2',ltr2') = (lhd (ltl ltr2), ltl ltr2)
in let (w1',w2',trv1,sv1',trv2,sv2',statOO) =
  (SOME k. case k of (w1',w2',trv1,sv1',trv2,sv2',statOO) ⇒
    ω4 Δ w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO)
in ((trv1,sv1',trv2,sv2',statA,statOO),
  (if trv2 = [] then R else L,
    w1',w2',s1,ltr1,s2',ltr2',statA,sv1',sv2',statOO))
else
let (tr2,s2',s2'',ltr2') =
  (SOME k. case k of (tr2,s2',s2'',ltr2') ⇒
    χχ s2 ltr2 tr2 s2' s2'' ltr2')
in let (w1',w2',trv1,sv1'',trv2,sv2'',statOO) =
  (SOME k'. case k' of (w1',w2',trv1,sv1'',trv2,sv2'',statOO) ⇒
    χ4' Δ w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2
sv2'' statOO)
  in ((trv1,sv1'',trv2,sv2'',statA,statOO),
    (L,w1',w2',s1, ltr1, s2'',s2'' $ ltr2',statA,sv1'',sv2'',statOO))
)

```

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

**definition** *h* ≡ *fst* o *h-t*

**definition** *t* ≡ *snd* o *h-t*

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

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

**definition** *f* :: *turn* × ('*stateO*','*stateV*)*tuple34* ⇒ ('*stateO*','*stateV*)*tuple12* *llist*

**where** *f* ≡ *ccorec-llist isn h econd e t*

**lemma** *f-LNil*:

$ltr1 = [] \implies ltr2 = [] \implies f (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) = []$   
(proof)

**lemma** *f-length-1*:

**assumes**  $ltr1 \neq [] \vee ltr2 \neq []$   $llength\ ltr1 \leq Suc\ 0 \vee llength\ ltr2 \leq Suc\ 0$   
**shows**  $f (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) = [[(sv1), sv1], [sv2], sv2, statA, statO]]$

(proof)

**lemma** *f-length-ge1*:

**assumes**  $llength\ ltr1 > Suc\ 0$   $llength\ ltr2 > Suc\ 0$

**shows**  $f (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$

$LCons (h (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)) (f (t (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)))$

(proof)

**definition** *lltrv1* ::  $turn \times ('stateO, 'stateV)tuple34 \Rightarrow 'stateV\ llist$  **where**

$lltrv1\ trn-tp = lconcat (lmap (\lambda(trv1, sv1'', trv2, sv2'', statAA, statOO). llist-of\ trv1) (f\ trn-tp))$

**definition** *then1* ::  $turn \times ('stateO, 'stateV)tuple34 \Rightarrow nat$  **where**

$then1\ trn-tp = firstNC (lmap (\lambda(trv1, sv1'', trv2, sv2'', statAA, statOO). trv1) (f\ trn-tp))$

**definition** *lltrv2* ::  $turn \times ('stateO, 'stateV)tuple34 \Rightarrow 'stateV\ llist$  **where**

$lltrv2\ trn-tp = lconcat (lmap (\lambda(trv1, sv1'', trv2, sv2'', statAA, statOO). llist-of\ trv2) (f\ trn-tp))$

**definition** *then2* ::  $turn \times ('stateO, 'stateV)tuple34 \Rightarrow nat$  **where**

$then2\ trn-tp = firstNC (lmap (\lambda(trv1, sv1'', trv2, sv2'', statAA, statOO). trv2) (f\ trn-tp))$

**lemma** *lltrv1-ne-imp*:

**assumes**  $lltrv1\ trn-tp \neq []$

**shows**  $\exists trv1\ sv1''\ trv2\ sv2''\ statAA\ statOO. (trv1, sv1'', trv2, sv2'', statAA, statOO) \in lset (f\ trn-tp) \wedge trv1 \neq []$

(proof)

**lemma** *lltrv2-ne-imp*:

**assumes**  $lltrv2\ trn-tp \neq []$

**shows**  $\exists trv1\ sv1''\ trv2\ sv2''\ statAA\ statOO. (trv1, sv1'', trv2, sv2'', statAA, statOO) \in lset (f\ trn-tp) \wedge trv2 \neq []$

(proof)

**lemma** *lltrv1-LNil[simp]*:  
 $ltr1 = [] \implies ltr2 = [] \implies lltrv1 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)$   
 $= []$   
 $\langle proof \rangle$

**lemma** *lltrv2-LNil[simp]*:  
 $ltr1 = [] \implies ltr2 = [] \implies lltrv2 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)$   
 $= []$   
 $\langle proof \rangle$

**lemma** *lltrv1-lnever[simp]*:  
**assumes**  $ltr1 \neq [] \vee ltr2 \neq []$   $llength\ ltr1 \leq Suc\ 0 \vee llength\ ltr2 \leq Suc\ 0$   
**shows**  $lltrv1 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) = [[sv1]]$   
 $\langle proof \rangle$

**lemma** *lltrv2-lnever[simp]*:  
**assumes**  $ltr1 \neq [] \vee ltr2 \neq []$   $llength\ ltr1 \leq Suc\ 0 \vee llength\ ltr2 \leq Suc\ 0$   
**shows**  $lltrv2 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) = [[sv2]]$   
 $\langle proof \rangle$

**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**  $\exists\ w1'\ w2'\ s1'\ ltr1'\ trv1\ sv1'\ trv2\ sv2'\ statOO.$   
 $ltr1 = s1\ \$\ ltr1' \wedge validTransO\ (s1, s1') \wedge$   
 $Opt.lvalidFromS\ s1'\ ltr1' \wedge Opt.lcompletedFrom\ s1'\ ltr1' \wedge lnever\ isIntO\ ltr1' \wedge$   
 $w3\ \Delta\ w1\ w2\ w1'\ w2'\ s1\ s1'\ s2\ statA\ sv1\ trv1\ sv1'\ sv2\ trv2\ sv2'\ statOO \wedge$   
 $h-t\ (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $((trv1, sv1', trv2, sv2', statA, statOO),$   
 $(if\ trv1 = []\ then\ L\ else\ R,$   
 $w1', w2', s1', ltr1', s2, ltr2, statA, sv1', sv2', statOO))$   
 $\langle proof \rangle$

**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**  $\exists w1' w2' s1' ltr1' trv1 sv1' trv2 sv2' statOO.$   
 $ltr1 = s1 \$ ltr1' \wedge validTransO (s1, s1') \wedge$   
 $Opt.lvalidFromS s1' ltr1' \wedge Opt.lcompletedFrom s1' ltr1' \wedge lnever isIntO ltr1' \wedge$   
  
 $\omega3 \Delta w1 w2 w1' w2' s1 s1' s2 statA sv1 trv1 sv1' sv2 trv2 sv2' statOO \wedge$   
 $lltrv1 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\text{list-of } trv1) (lltrv1 (if trv1 = [] \text{ then } L \text{ else } R,$   
 $w1', w2', s1', ltr1', s2, ltr2, statA, sv1', sv2', statOO)) \wedge$   
 $lltrv2 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\text{list-of } trv2) (lltrv2 (if trv1 = [] \text{ then } L \text{ else } R,$   
 $w1', w2', s1', ltr1', s2, ltr2, statA, sv1', sv2', statOO))$   
 $\langle proof \rangle$

**lemma**  $h\text{-}t\text{-not-}lnever\text{-}L$ :

**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**  $\exists w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO.$   
 $\chi\chi s1 ltr1 tr1 s1' s1'' ltr1' \wedge$   
 $\chi3' \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO$   
 $\wedge$   
 $h\text{-}t (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $((trv1, sv1'', trv2, sv2'', statA, statOO),$   
 $(R, w1', w2', s1'', s1'' \$ ltr1', s2, ltr2, statA, sv1'', sv2'', statOO))$   
 $\langle proof \rangle$

**lemma**  $lltrv1\text{-}lltrv2\text{-not-}lnever\text{-}L$ :

**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**  $\exists w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO.$   
 $\chi\chi s1 ltr1 tr1 s1' s1'' ltr1' \wedge$

$\chi^3 \Delta w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO$   
 $\wedge$   
 $lltrv1 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\text{list-of } trv1) (lltrv1 (R, w1', w2', s1'', s1'' \$ ltr1', s2, ltr2, statA, sv1'', sv2'', statOO))$   
 $\wedge$   
 $lltrv2 (trn, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\text{list-of } trv2) (lltrv2 (R, w1', w2', s1'', s1'' \$ ltr1', s2, ltr2, statA, sv1'', sv2'', statOO))$   
 $\langle proof \rangle$

**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**  $\exists w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO.$

$ltr2 = s2 \$ ltr2' \wedge validTransO (s2, s2') \wedge$

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

$w4 \Delta w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO \wedge$

*h-t* (*trn*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*) =

((*trv1*, *sv1'*, *trv2*, *sv2'*, *statA*, *statOO*),

(*if* *trv2* =  $\square$  *then* *R* *else* *L*,

$w1', w2', s1, ltr1, s2', ltr2', statA, sv1', sv2', statOO$ ))

$\langle proof \rangle$

**lemma** *lltrv1-lltrv2-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**  $\exists w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO.$

$ltr2 = s2 \$ ltr2' \wedge validTransO (s2, s2') \wedge$

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

$w4 \Delta w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO \wedge$

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

*lappend* (*l\text{list-of}* *trv1*) (*lltrv1* (*if* *trv2* =  $\square$  *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\text{list-of}* *trv2*) (*lltrv2* (*if* *trv2* =  $\square$  *then* *R* *else* *L*,

$w1', w2', s1, ltr1, s2', ltr2', statA, sv1', sv2', statOO$ )

$\langle proof \rangle$

**lemma** *h-t-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'*  $\wedge$

$\chi4'$   $\Delta$  *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*))

$\langle proof \rangle$

**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'*  $\wedge$

$\chi4'$   $\Delta$  *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* (*l**list-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* (*l**list-of* *trv2*) (*lltrv2* (*L*, *w1'*, *w2'*, *s1*, *ltr1*, *s2''*, *s2''*  $\$$  *ltr2'*, *statA*, *sv1''*, *sv2''*, *statOO*))

$\langle proof \rangle$

**lemma** *f-not-LNil*: *ltr1*  $\neq$   $[\ ]$   $\vee$  *ltr2*  $\neq$   $[\ ]$   $\implies$

*f* (*w1*, *w2*, *trn*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)  $\neq$   $[\ ]$

$\langle proof \rangle$

**lemma** *lvalidFromS-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**  $Van.lvalidFromS sv1 (lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))$   
 $\langle proof \rangle$

**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**  $Van.lvalidFromS sv2 (lltrv2 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))$   
 $\langle proof \rangle$

**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**  $Van.lcompletedFrom sv1 (lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))$   
 $\langle proof \rangle$

**lemma**  $lcompletedFrom$ - $lltrv2$ :  
**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**  $Van.lcompletedFrom sv2 (lltrv2 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))$   
 $\langle proof \rangle$

**lemma**  $lS$ - $lltrv1$ - $ltr1$ :  
**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**  $ltr1$ :  $Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1$   
**and**  $ltr2$ :  $Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2$   
**shows**  $Van.lS (lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)) = Opt.lS ltr1$   
 $\langle proof \rangle$

**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>

```

```

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>

```

```

fun isN :: ('stateO,'stateV) tuple34  $\Rightarrow$  bool
where
isN (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)  $\longleftrightarrow$  ltr1 = []  $\vee$  ltr2 = []

```

```

fun H-T ::
('stateO,'stateV) tuple34  $\Rightarrow$ 
('stateO,'stateV) tuple12  $\times$ 
('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')  $\Rightarrow$ 
 $\varphi\varphi$  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)  $\Rightarrow$ 
 $\varphi'$   $\Delta$  w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO
sv1 trv1 sv1'' sv2 trv2 sv2'' statOO)
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**  $H \equiv fst \circ H-T$   
**definition**  $T \equiv snd \circ H-T$

**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**  $F :: ('stateO,'stateV)tuple34 \Rightarrow ('stateO,'stateV)tuple12 llist$   
**where**  $F \equiv ccorec-llist isN H Econd E T$

**lemma**  $F-LNil$ :

$ltr1 = [] \vee ltr2 = [] \implies F (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) = []$   
 $\langle proof \rangle$

**lemma**  $F-lnever$ :

**assumes**  $ltr1 \neq []$   $ltr2 \neq []$   $lnever isIntO ltr1$   
**shows**  $F (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) = f (L, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)$   
 $\langle proof \rangle$

**lemma**  $F-not-lnever$ :

**assumes**  $ltr1 \neq []$   $ltr2 \neq []$   $\neg lnever isIntO ltr1$   
**shows**  $F (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =$   
 $LCons (H (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)) (F (T (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)))$   
 $\langle proof \rangle$

**definition**  $ltrv1 :: ('stateO,'stateV)tuple34 \Rightarrow 'stateV llist$  **where**

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

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

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

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

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

**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**  $\exists\ trv1\ sv1''\ trv2\ sv2''\ statAA\ statOO.\ (trv1,sv1'',trv2,sv2'',statAA,statOO) \in\ lset\ (F\ tp) \wedge$   
 $trv1 \neq []$   
 $\langle proof \rangle$

**lemma** *ltrv2-ne-imp*:  
**assumes**  $ltrv2\ tp \neq []$   
**shows**  $\exists\ trv1\ sv1''\ trv2\ sv2''\ statAA\ statOO.\ (trv1,sv1'',trv2,sv2'',statAA,statOO) \in\ lset\ (F\ tp) \wedge$   
 $trv2 \neq []$   
 $\langle proof \rangle$

**lemma** *ltrv1-LNil[simp]*:  
 $ltr1 = [] \vee ltr2 = [] \implies ltrv1\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) = []$   
 $\langle proof \rangle$

**lemma** *ltrv2-LNil[simp]*:  
 $ltr1 = [] \vee ltr2 = [] \implies ltrv2\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) = []$   
 $\langle proof \rangle$

**lemma** *ltrv1-lnever*:  
**assumes**  $ltr1 \neq []\ ltr2 \neq []\ lnever\ isIntO\ ltr1$   
**shows**  $ltrv1\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) = lltrv1\ (L,\ w1,\ w2,\ s1,\ ltr1,\ s2,\ ltr2,\ statA,\ sv1,\ sv2,\ statO)$   
 $\langle proof \rangle$

**lemma** *ltrv2-lnever*:  
**assumes**  $ltr1 \neq []\ ltr2 \neq []\ lnever\ isIntO\ ltr1$   
**shows**  $ltrv2\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) = lltrv2\ (L,\ w1,\ w2,\ s1,\ ltr1,\ s2,\ ltr2,\ statA,\ sv1,\ sv2,\ statO)$   
 $\langle proof \rangle$

**lemma** *H-T-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**  $\exists w1' w2' tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statAA statOO$ .

$\varphi\varphi s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2' \wedge$   
 $\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO sv1 trv1$   
 $sv1'' sv2 trv2 sv2'' statOO \wedge$   
 $H-T (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $((trv1, sv1'', trv2, sv2'', statAA, statOO),$   
 $(w1', w2', s1'', s1'' \$ ltr1', s2'', s2'' \$ ltr2', statAA, sv1'', sv2'', statOO))$   
 $\langle proof \rangle$

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

$\varphi\varphi s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2' \wedge$   
 $\varphi' \Delta w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO sv1 trv1$   
 $sv1'' sv2 trv2 sv2'' statOO \wedge$   
 $ltrv1 (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\text{-list-of } trv1) (ltrv1 (w1', w2', s1'', s1'' \$ ltr1', s2'', s2'' \$ ltr2', statAA, sv1'', sv2'', statOO))$   
 $\wedge$   
 $ltrv2 (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO) =$   
 $lappend (l\text{-list-of } trv2) (ltrv2 (w1', w2', s1'', s1'' \$ ltr1', s2'', s2'' \$ ltr2', statAA, sv1'', sv2'', statOO))$   
 $\langle proof \rangle$

**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))*

$\langle proof \rangle$

**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**  $Van.lvalidFromS\ sv2\ (ltrv2\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))$   
 $\langle proof \rangle$

**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**  $Van.lcompletedFrom\ sv1\ (ltrv1\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))$   
 $\langle proof \rangle$

**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**  $Van.lcompletedFrom\ sv2\ (ltrv2\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO))$   
 $\langle proof \rangle$

**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**  $Van.lS\ (ltrv1\ (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO)) = Opt.lS\ ltr1$   
 $\langle proof \rangle$

**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**  $Van.lS\ (ltrv2\ (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)) = Opt.lS\ ltr2$   
 $\langle proof \rangle$

**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**  $Van.lA\ (ltrv1\ (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)) =$   
 $Van.lA\ (ltrv2\ (w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)) =$   
 $\langle proof \rangle$

**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**  $Opt.lO\ ltr1 = Opt.lO\ ltr2$   
 $\langle proof \rangle$

**end**

**theorem**  $unwind-lrsecure$ :  
**assumes**  $init$ :  $initCond\ \Delta$  **and**  $unwind$ :  $unwindCond\ \Delta$   
**shows**  $lrsecure$   
 $\langle proof \rangle$

#### 4.4 Compositional unwinding

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

**definition**  $unwindIntoCond$  ::

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

**where**

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

**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**  $\text{init}: \text{initCond } (\Delta s \ 0)$

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

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

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

**shows** *lrsecure*

$\langle \text{proof} \rangle$

**lemma** *unwindIntoCond-simpleI*:

**assumes**

$\bigwedge w \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}.$

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

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

$\implies$

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

**and**

$\bigwedge w \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}.$

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

$\Delta \ w \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO} \implies$

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

$\implies$

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

$\bigwedge w \ w1 \ w2 \ s1 \ s2 \ \text{statA} \ sv1 \ sv2 \ \text{statO}.$

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

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

$\implies$

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

**shows**  $\text{unwindIntoCond } \Delta \ \Delta'$

*<proof>*

**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*  $\Delta \Delta'$

*<proof>*

**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*  $\Delta \Delta'$

*<proof>*

**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:**

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor \Delta \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO$

*<proof>*

**lemma oorI2:**

$\Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor \Delta \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO$

*<proof>*

**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:**

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO$

*<proof>*

**lemma oor3I2:**

$\Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO$

*<proof>*

**lemma oor3I3:**

$\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor3 \Delta \Delta_2 \Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO$

*<proof>*

**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:**

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$

*<proof>*

**lemma oor4I2:**

$\Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 \langle proof \rangle

**lemma** *oor4I3*:

$\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 \langle proof \rangle

**lemma** *oor4I4*:

$\Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor4 \Delta \Delta_2 \Delta_3 \Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 \langle proof \rangle

**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*:

$\Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 \langle proof \rangle

**lemma** *oor5I2*:

$\Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 \langle proof \rangle

**lemma** *oor5I3*:

$\Delta_3 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 \langle proof \rangle

**lemma** *oor5I4*:

$\Delta_4 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 \langle proof \rangle

**lemma** *oor5I5*:

$\Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO \implies oor5 \Delta \Delta_2 \Delta_3 \Delta_4 \Delta_5 w w1 w2 s1 s2 statA sv1 sv2 statO$   
 \langle proof \rangle

**lemma** *isIntO-match1*:  $isIntO\ s1 \implies match1\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
*<proof>*

**lemma** *isIntO-match2*:  $isIntO\ s2 \implies match2\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
*<proof>*

**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**  $\langle react\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \rangle$   
*<proof>*

**lemma** *match1-1-oorI1*:  
 $match1-1\ \Delta\ w1\ w2\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $match1-1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO$   
*<proof>*

**lemma** *match1-1-oorI2*:  
 $match1-1\ \Delta_2\ w1\ w2\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $match1-1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO$   
*<proof>*

**lemma** *match1-oorI1*:  
 $match1\ \Delta\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $match1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
*<proof>*

**lemma** *match1-oorI2*:  
 $match1\ \Delta_2\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \implies$   
 $match1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$   
*<proof>*

**lemma** *match2-1-oorI1*:  
 $match2-1\ \Delta\ w1\ w2\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO \implies$   
 $match2-1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO$   
*<proof>*

**lemma** *match2-1-oorI2*:  
 $match2-1\ \Delta_2\ w1\ w2\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO \implies$   
 $match2-1\ (oor\ \Delta\ \Delta_2)\ w1\ w2\ s1\ s2\ s2'\ statA\ sv1\ sv2\ statO$   
*<proof>*

**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*  
<proof>

**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*  
<proof>

**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*  
<proof>

**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*  
<proof>

**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*  
<proof>

**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*  
<proof>

**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*  
<proof>

**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*  
<proof>

**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*  
<proof>

**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*  
<proof>

**lemma** *match-oorI1*:

$react \Delta w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $react (oor \Delta \Delta_2) w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *match-oorI2*:

$react \Delta_2 w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $react (oor \Delta \Delta_2) w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *proact-oorI1*:

$proact \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $proact (oor \Delta \Delta_2) w w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *proact-oorI2*:

$proact \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $proact (oor \Delta \Delta_2) w w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *move-1-oorI1*:

$move-1 \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $move-1 (oor \Delta \Delta_2) w w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *move-1-oorI2*:

$move-1 \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $move-1 (oor \Delta \Delta_2) w w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *move-2-oorI1*:

$move-2 \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $move-2 (oor \Delta \Delta_2) w w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *move-2-oorI2*:

$move-2 \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $move-2 (oor \Delta \Delta_2) w w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *move-12-oorI1*:

$move-12 \Delta w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $move-12 (oor \Delta \Delta_2) w w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**lemma** *move-12-oorI2*:

$move-12 \Delta_2 w w1 w2 s1 s2 statA sv1 sv2 statO \implies$   
 $move-12 (oor \Delta \Delta_2) w w1 w2 s1 s2 statA sv1 sv2 statO$   
*<proof>*

**end**

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

**begin**

**lemma** *match1-1-defD*:  $match1-1 \Delta w1 w2 s1 s1' s2 statA sv1 sv2 statO \longleftrightarrow$

$\neg finalV sv1 \wedge \Delta \infty w1 w2 s1' s2 statA (nextO sv1) sv2 statO$   
*<proof>*

**lemma** *match1-12-defD*:  $match1-12 \Delta w1 w2 s1 s1' s2 statA sv1 sv2 statO \longleftrightarrow$

$\neg finalV sv1 \wedge \neg finalV sv2 \wedge$   
 $\Delta \infty w1 w2 s1' s2 statA (nextO sv1) (nextO sv2) (sstatO' statO sv1 sv2)$   
*<proof>*

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

**lemma** *match2-1-defD*:  $match2-1 \Delta w1 w2 s1 s2 s2' statA sv1 sv2 statO \longleftrightarrow$

$\neg finalV sv2 \wedge \Delta \infty w1 w2 s1 s2' statA sv1 (nextO sv2) statO$   
*<proof>*

**lemma** *match2-12-defD*:  $match2-12 \Delta w1 w2 s1 s2 s2' statA sv1 sv2 statO \longleftrightarrow$

$\neg finalV sv1 \wedge \neg finalV sv2 \wedge \Delta \infty w1 w2 s1 s2' statA (nextO sv1) (nextO sv2)$   
 $(sstatO' statO sv1 sv2)$   
*<proof>*

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

**lemma** *match12-1-defD*:  $match12-1 \Delta w1 w2 s1' s2' statA' sv1 sv2 statO \longleftrightarrow$

$\neg finalV sv1 \wedge \Delta \infty w1 w2 s1' s2' statA' (nextO sv1) sv2 statO$   
*<proof>*

**lemma** *match12-2-defD*:  $match12-2 \Delta w1 w2 s1' s2' statA' sv1 sv2 statO \longleftrightarrow$

$\neg finalV sv2 \wedge \Delta \infty w1 w2 s1' s2' statA' sv1 (nextO sv2) statO$   
*<proof>*

**lemma** *match12-12-defD*:  $match12-12 \Delta w1 w2 s1' s2' statA' sv1 sv2 statO \longleftrightarrow$

```

    (let statO' = sstatO' statO sv1 sv2 in
      ¬ finalV sv1 ∧ ¬ finalV sv2 ∧
      (statA' = Diff → statO' = Diff) ∧
      Δ ∞ w1 w2 s1' s2' statA' (nextO sv1) (nextO sv2) statO')
  ⟨proof⟩

```

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

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

**lemma** *move-1-defD*: *move-1 Δ w w1 w2 s1 s2 statA sv1 sv2 statO ↔*  
 ¬ *finalV sv1* ∧ Δ *w w1 w2 s1 s2 statA (nextO sv1) sv2 statO*  
 ⟨proof⟩

**lemma** *move-2-defD*: *move-2 Δ w w1 w2 s1 s2 statA sv1 sv2 statO ↔*  
 ¬ *finalV sv2* ∧ Δ *w w1 w2 s1 s2 statA sv1 (nextO sv2) statO*  
 ⟨proof⟩

**lemma** *move-12-defD*: *move-12 Δ w w1 w2 s1 s2 statA sv1 sv2 statO ↔*  
 (let *statO' = sstatO' statO sv1 sv2 in*  
 ¬ *finalV sv1* ∧ ¬ *finalV sv2* ∧  
 Δ *w w1 w2 s1 s2 statA (nextO sv1) (nextO sv2) statO'*)  
 ⟨proof⟩

**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'*  $\langle$ *proof* $\rangle$

**lemma** *transit-iff*::*s*  $\rightarrow I$  *s'*  $\longleftrightarrow$  (*s* = *ss*  $\wedge$  *s'* = *ss'*)  $\langle$ *proof* $\rangle$

**definition** *final* *x*  $\equiv$   $\forall y. \neg (\rightarrow I) x y$

**lemma** *final-so'*[*simp*]:*final* *ss'*  $\langle$ *proof* $\rangle$

**lemma** *final-iff*: *final* *s*  $\longleftrightarrow$  *s* = *ss'*  $\langle$ *proof* $\rangle$

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**  $corrState = corrState$   
*<proof>*

**lemma**  $validFromV: Van.validFromS\ ss\ [ss, ss']\ \langle proof \rangle$

**lemma**  $tr1\text{-shape}: Opt.validFromS\ ss\ tr1 \implies completedFromO\ ss\ tr1 \implies tr1 = [ss, ss']\ \langle proof \rangle$

**lemma**  $tr1\text{-shape}' : s1 = ss \implies Opt.validFromS\ s1\ tr1 \implies completedFromO\ s1\ tr1 \implies tr1 = [ss, ss']\ \langle proof \rangle$

**proposition** *rsecure*  
*<proof>*

**lemma**  $validSS: Opt.validS\ [ss, ss']\ \langle proof \rangle$

**lemma**  $validSS\text{-van}: Van.validS\ [ss, ss']\ \langle proof \rangle$

**lemma**  $reachOs: reachO\ ss\ \langle proof \rangle$

**lemma**  $reachVs: reachV\ ss\ \langle proof \rangle$

**lemma**  $reachO': reachO\ ss'\ \langle proof \rangle$

**lemma**  $reachV': reachV\ ss'\ \langle proof \rangle$

**lemma**  $impE\text{-eq}: x = x \implies Q \implies (Q \implies Rs) \implies Rs\ \langle proof \rangle$

**lemma**  $isSecOs: isSecO\ ss\ \langle proof \rangle$

**lemma**  $neq\text{-Sec}: \neg eqSec\ ss\ ss\ \langle proof \rangle$

**lemma**  $statOs: (sstatO'\ Eq\ ss\ ss) = Eq\ \langle proof \rangle$

**lemma**  $noUnwind: \text{shows } init: \Delta \infty\ ss\ ss\ Eq\ ss\ ss\ Eq \implies unwindCond\ \Delta \implies False$

*<proof>*

**lemma** *incomplete-fin*:  
  **assumes** *init*: *initCond*  $\Delta$   
  **and** *unwind*: *unwindCond*  $\Delta$   
  **shows** *False*  
  *<proof>*

**end**

## 6 Relative Security Unwinding Incompleteness example

Demonstrating a counterexample which is secure but fails in the infinitary 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'* *<proof>*

**lemma** *transit-iff*:*s*  $\rightarrow I$  *s'*  $\longleftrightarrow$  (*s* = *ss*  $\wedge$  *s'* = *ss'*) *<proof>*

**definition** *final* *x*  $\equiv \forall y. \neg (\rightarrow I) x y$

**lemma** *final-sv*[*simp*]:*final* *ss'* *<proof>*

**lemma** *final-iff*: *final* *s*  $\longleftrightarrow$  *s* = *ss'* *<proof>*

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* = *getIntO*  
**and** *corrState* = *corrState*  
*<proof>*

**lemma** *validTrFinite:Opt.lvalidFromS* *ss* *tr1*  $\Longrightarrow$  *lfinite* *tr1*  
*<proof>*

**lemma** *tr1-shape:Opt.lvalidFromS* *ss* *tr1*  $\Longrightarrow$  *lcompletedFromO* *ss* *tr1*  $\Longrightarrow$  *tr1* =  
[[*ss*, *ss*]]  
*<proof>*

**lemma**  $tr1\text{-shape}' : s1 = ss \implies Opt.\text{lvalidFromS } s1 \text{ } tr1 \implies l\text{completedFromO } s1$   
 $tr1 \implies tr1 = [[ss, ss']]$   
 $\langle proof \rangle$

**proposition**  $lrsecure$   
 $\langle proof \rangle$

**lemma**  $validSS : Opt.\text{validS } [ss, ss'] \langle proof \rangle$   
**lemma**  $validSS\text{-van} : Van.\text{validS } [ss, ss'] \langle proof \rangle$

**lemma**  $reachOs : reachO \ ss \langle proof \rangle$   
**lemma**  $reachVs : reachV \ ss \langle proof \rangle$   
**lemma**  $reachO' : reachO \ ss' \langle proof \rangle$   
**lemma**  $reachV' : reachV \ ss' \langle proof \rangle$

**lemma**  $impE\text{-eq} : x = x \longrightarrow Q \implies (Q \implies Rs) \implies Rs \langle proof \rangle$

**lemma**  $isSecOs : isSecO \ ss \langle proof \rangle$   
**lemma**  $neq\text{-Sec} : \neg eqSec \ ss \ ss \langle proof \rangle$

**lemma**  $statOs : (sstatO' \ Eq \ ss \ ss) = Eq \langle proof \rangle$

**lemma**  $noUnwind : \text{shows } init : \Delta \infty \infty \infty \ ss \ ss \ Eq \ ss \ ss \ Eq \implies unwindCond \ \Delta$   
 $\implies False$   
 $\langle proof \rangle$

**lemma**  $incomplete\text{-inf} :$   
**assumes**  $init : initCond \ \Delta$   
**and**  $unwind : unwindCond \ \Delta$   
**shows**  $False$   
 $\langle proof \rangle$

**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.