Relative Security

Andrei Popescu Jamie Wright

June 5, 2024

Abstract

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

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

Contents

1 Finitary Relative Security

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

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

declare *Let-def*[*simp*]

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

1.1 Finite-trace versions of leakage models and attacker models

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

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

fun *leakVia* :: *'state list* ⇒ *'state list* ⇒ *'secret list* × *'secret list* ⇒ *bool*
where
leakVia tr tr' (*sl*,*sl'*) = (*S tr* = *sl* ∧ *S tr'* = *sl'* ∧ *A tr* = *A tr'* ∧ *O tr* ≠ *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 SV AV OV leakViaV*
+
Opt: *Leakage-Mod-fin istateO validTransO finalO SO AO OO leakViaO*
for *validTransV* :: *'stateV* × *'stateV* ⇒ *bool*
and *istateV* :: *'stateV* ⇒ *bool* **and** *finalV* :: *'stateV* ⇒ *bool*
and *SV* :: *'stateV list* ⇒ *'secret list*
and *AV* :: *'stateV trace* ⇒ *'actV list*
and *OV* :: *'stateV trace* ⇒ *'obsV list*
and *leakViaV* :: *'stateV list* ⇒ *'stateV list* ⇒ *'leak* ⇒ *bool*

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

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

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

end

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

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

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

sublocale *Relative-Security'-fin* $<$ *Relative-Security''-fin*
where $leakViaV = Van.leakVia$ **and** $leakViaO = Opt.leakVia$
⟨*proof*⟩

context *Relative-Security'-fin*
begin

lemma *rsecure-def2*:
$rsecure \longleftrightarrow$
$(\forall\ s1\ tr1\ s2\ tr2.$
$istateO\ s1 \wedge Opt.validFromS\ s1\ tr1 \wedge Opt.completedFrom\ s1\ tr1 \wedge$
$istateO\ s2 \wedge Opt.validFromS\ s2\ tr2 \wedge Opt.completedFrom\ s2\ tr2 \wedge$
$AO\ tr1 = AO\ tr2 \wedge OO\ tr1 \neq OO\ tr2$
$\longrightarrow$
$(\exists\ sv1\ trv1\ sv2\ trv2.$
$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$
$Van.validFromS\ sv1\ trv1 \wedge Van.completedFrom\ sv1\ trv1 \wedge$
$Van.validFromS\ sv2\ trv2 \wedge Van.completedFrom\ sv2\ trv2 \wedge$
$SV\ trv1 = SO\ tr1 \wedge SV\ trv2 = SO\ tr2 \wedge$
$AV\ trv1 = AV\ trv2 \wedge OV\ trv1 \neq OV\ trv2))$
⟨*proof*⟩

end

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

definition *getAct* :: *'state* ⇒ *'act* **where**
getAct = *fst o getInt*

definition *getObs* :: *'state* ⇒ *'obs* **where**
getObs = *snd o getInt*

definition *eqObs trn1 trn2* ≡
(*isInt trn1* ⟷ *isInt trn2*) ∧ (*isInt trn1* ⟶ *getObs trn1* = *getObs trn2*)

definition *eqAct trn1 trn2* ≡
(*isInt trn1* ⟷ *isInt trn2*) ∧ (*isInt trn1* ⟶ *getAct trn1* = *getAct trn2*)

definition *A* :: *'state trace* ⇒ *'act list* **where**
A tr ≡ *filtermap isInt getAct* (*butlast tr*)

sublocale *A*: *FiltermapBL isInt getAct A*
⟨*proof*⟩

definition *O* :: *'state trace* ⇒ *'obs list* **where**
O tr ≡ *filtermap isInt getObs* (*butlast tr*)

sublocale *O*: *FiltermapBL isInt getObs O*
⟨*proof*⟩

definition *S* :: *'state list* ⇒ *'secret list* **where**
S tr ≡ *filtermap isSec getSec* (*butlast tr*)

sublocale *S*: *FiltermapBL isSec getSec S*
⟨*proof*⟩

end

sublocale *Statewise-Attacker-Mod* < *Attacker-Mod-fin*
where *S* = *S* **and** *A* = *A* **and** *O* = *O*
⟨*proof*⟩

locale *Rel-Sec* =
Van: *Statewise-Attacker-Mod istateV validTransV finalV isSecV getSecV isIntV getIntV*

+
Opt: *Statewise-Attacker-Mod istateO validTransO finalO isSecO getSecO isIntO getIntO*
for $validTransV :: 'stateV \times 'stateV \Rightarrow bool$
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-Sec* $<$ *Relative-Security'-fin*
where $SV = Van.S$ **and** $AV = Van.A$ **and** $OV = Van.O$
and $SO = Opt.S$ **and** $AO = Opt.A$ **and** $OO = Opt.O$
⟨*proof*⟩

context *Rel-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-def* $=$ *Van.completedFrom-def*
lemmas *completedFromO-def* $=$ *Opt.completedFrom-def*

lemma *rsecure-def3*:
$rsecure \longleftrightarrow$
$(\forall\ s1\ tr1\ s2\ tr2.$
$\quad istateO\ s1 \wedge Opt.validFromS\ s1\ tr1 \wedge completedFromO\ s1\ tr1 \wedge$
$\quad istateO\ s2 \wedge Opt.validFromS\ s2\ tr2 \wedge completedFromO\ s2\ tr2 \wedge$
$\quad Opt.A\ tr1 = Opt.A\ tr2 \wedge Opt.O\ tr1 \neq Opt.O\ tr2 \wedge$
$\quad (isIntO\ s1 \wedge isIntO\ s2 \longrightarrow getActO\ s1 = getActO\ s2)$
$\quad \longrightarrow$

$(\exists\, sv1\ trv1\ sv2\ trv2.$
$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$
$Van.validFromS\ sv1\ trv1 \wedge completedFromV\ sv1\ trv1 \wedge$
$Van.validFromS\ sv2\ trv2 \wedge completedFromV\ sv2\ trv2 \wedge$
$Van.S\ trv1 = Opt.S\ tr1 \wedge Van.S\ trv2 = Opt.S\ tr2 \wedge$
$Van.A\ trv1 = Van.A\ trv2 \wedge Van.O\ trv1 \neq Van.O\ trv2))$
$\langle proof \rangle$

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

lemma *eqSec-S-Cons'*:
eqSec trnO trnA $\Longrightarrow$
$(Van.S\ (trnO\ \#\ trO') = Opt.S\ (trnA\ \#\ trA')) \Longrightarrow Van.S\ trO' = Opt.S\ trA'$
$\langle proof \rangle$

lemma *eqSec-S-Cons*[*simp*]:
eqSec trnO trnA $\Longrightarrow trO' = [] \longleftrightarrow trA' = [] \Longrightarrow$
$(Van.S\ (trnO\ \#\ trO') = Opt.S\ (trnA\ \#\ trA')) \longleftrightarrow (Van.S\ trO' = Opt.S\ trA')$
$\langle proof \rangle$

end

locale *Relative-Security-Determ* =
Rel-Sec
validTransV istateV finalV isSecV getSecV isIntV getIntV
validTransO istateO finalO isSecO getSecO isIntO getIntO
corrState
+
System-Mod-Deterministic istateV validTransV finalV nextO
for *validTransV* :: $'stateV \times 'stateV \Rightarrow bool$
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/Trivia*
begin

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

2.1 Leakage models and attacker models

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

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

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

lemmas *lleakVia-def = lleakVia.simps*

end

sublocale *Attacker-Mod < Leakage-Mod*
where *lleakVia = lleakVia*
⟨*proof*⟩

2.2 Locales for increasingly concrete notions of relative security

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

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

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

definition *lrsecure* :: *bool* **where**
lrsecure ≡ ∀ *l s1 tr1 s2 tr2*.
  istateO s1 ∧ *Opt.lvalidFromS s1 tr1* ∧ *Opt.lcompletedFrom s1 tr1* ∧
  istateO s2 ∧ *Opt.lvalidFromS s2 tr2* ∧ *Opt.lcompletedFrom s2 tr2* ∧
  lleakViaO tr1 tr2 l
  ⟶
  (∃ *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* ∧
    lleakViaV trv1 trv2 l)

end

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

and $OV :: 'stateV\ ltrace \Rightarrow 'obsV\ llist$

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

sublocale *Relative-Security'* < *Relative-Security''*
where $lleakViaV = Van.lleakVia$ **and** $lleakViaO = Opt.lleakVia$
⟨*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* ⇒ *'obs llist* **where**
lO tr ≡ *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* ⇒ *'secret llist* **where**
lS tr ≡ *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* ⇒ *'stateV llist* ⇒ *bool* **where** *lcompletedFromV* ≡ *Van.lcompletedFrom*
abbreviation *lcompletedFromO* :: *'stateO* ⇒ *'stateO llist* ⇒ *bool* **where** *lcompletedFromO* ≡ *Opt.lcompletedFrom*

lemma *eqSec-lS-Cons'*:
eqSec trnO trnA $\Longrightarrow$
(*Van.lS* (*trnO* \$ *trO'*) = *Opt.lS* (*trnA* \$ *trA'*)) $\Longrightarrow$ *Van.lS trO'* = *Opt.lS trA'*
⟨*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 *updStat* :: *status* ⇒ *bool* × *'a* ⇒ *bool* × *'a* ⇒ *status* **where**
updStat Eq (*True*,*a*) (*True*,*a'*) = (*if a* = *a' then Eq else Diff*)
|*updStat stat - - = stat*

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

definition *initCond* ::
(*enat* ⇒ *'stateO* ⇒ *'stateO* ⇒ *status* ⇒ *'stateV* ⇒ *'stateV* ⇒ *status* ⇒ *bool*) ⇒ *bool* **where**
initCond Δ ≡ ∀ *s1 s2*.
istateO s1 ∧ *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* $\Delta\ s1\ s2\ statA\ sv1\ sv2\ statO \equiv$
$\neg\ isIntO\ s1 \longrightarrow$
$(\forall\, s1'.\ validTransO\ (s1,s1')$
$\longrightarrow$
$(\neg\ isSecO\ s1 \wedge \Delta \infty\ s1'\ s2\ statA\ sv1\ sv2\ statO) \vee$
$(eqSec\ sv1\ s1 \wedge \neg\ isIntV\ sv1 \wedge match1\text{-}1\ \Delta\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO) \vee$
$(eqSec\ sv1\ s1 \wedge \neg\ isSecV\ sv2 \wedge Van.eqAct\ sv1\ sv2 \wedge match1\text{-}12\ \Delta\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO))$

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

lemma *match1-1-mono*:
$\Delta \leq \Delta' \Longrightarrow match1\text{-}1\ \Delta\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow match1\text{-}1\ \Delta'\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO$
⟨*proof*⟩

lemma *match1-12-mono*:
$\Delta \leq \Delta' \Longrightarrow match1\text{-}12\ \Delta\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow match1\text{-}12\ \Delta'\ s1\ s1'\ s2\ statA\ sv1\ sv2\ statO$
⟨*proof*⟩

lemma *match1-mono*:
assumes $\Delta \leq \Delta'$
shows $match1\ \Delta\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow match1\ \Delta'\ s1\ s2\ statA\ sv1\ sv2\ statO$
⟨*proof*⟩

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

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

lemma *match2-1-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match2-1* Δ *s1 s1′ s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2-1* Δ' *s1 s1′ s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-12-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match2-12* Δ *s1 s1′ s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2-12* Δ' *s1 s1′ s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-mono*:
assumes $\Delta \leq \Delta'$
shows *match2* Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2* Δ' *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

definition *match12-1* Δ *s1′ s2′ statA′ sv1 sv2 statO* $\equiv$
$\exists$ *sv1′. validTransV* (*sv1*,*sv1′*) $\wedge$
$\Delta \infty$ *s1′ s2′ statA′ sv1′ sv2 statO*

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

definition *match12-12* Δ *s1′ s2′ statA′ sv1 sv2 statO* $\equiv$
$\exists$ *sv1′ sv2′.*
let statO′ = *sstatO′ statO sv1 sv2 in*

$validTransV\ (sv1,sv1') \wedge$
$validTransV\ (sv2,sv2') \wedge$
$(statA' = Diff \longrightarrow statO' = Diff) \wedge$
$\Delta \infty\ s1'\ s2'\ statA'\ sv1'\ sv2'\ statO'$

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

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

lemma *match12-simpleI*:
assumes $\bigwedge s1'\ s2'\ statA'.$
$statA' = sstatA'\ statA\ s1\ s2 \Longrightarrow$
$validTransO\ (s1,s1') \Longrightarrow$
$validTransO\ (s2,s2') \Longrightarrow$
$Opt.eqAct\ s1\ s2 \Longrightarrow$
$(\neg\ isSecO\ s1 \wedge \neg\ isSecO\ s2 \wedge (statA = statA' \vee statO = Diff) \wedge \Delta \infty\ s1'\ s2'\ statA'\ sv1\ sv2\ statO)$
$\vee$
$(eqSec\ sv1\ s1 \wedge eqSec\ sv2\ s2 \wedge Van.eqAct\ sv1\ sv2 \wedge$
$match12\text{-}12\ \Delta\ s1'\ s2'\ statA'\ sv1\ sv2\ statO)$
shows *match12* $\Delta\ s1\ s2\ statA\ sv1\ sv2\ statO$
⟨*proof*⟩

lemma *match12-1-mono*:
$\Delta \leq \Delta' \Longrightarrow match12\text{-}1\ \Delta\ s1'\ s2'\ statA'\ sv1\ sv2\ statO \Longrightarrow match12\text{-}1\ \Delta'\ s1'\ s2'\ statA'\ sv1\ sv2\ statO$
⟨*proof*⟩

lemma *match12-2-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match12-2* Δ *s1 s2' statA' sv1 sv2 statO* $\Longrightarrow$ *match12-2* Δ' *s1 s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-12-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *match12-12* Δ *s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$ *match12-12* Δ' *s1' s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-mono*:
assumes $\Delta \leq \Delta'$
shows *match12* Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match12* Δ' *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

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

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

lemma *match-mono*:
assumes $\Delta \leq \Delta'$
shows *match* Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match* Δ' *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

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

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

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

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

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

lemma *move-1-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-1* Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-1* Δ' *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-2-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-2* Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-2* Δ' *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-12-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-12* Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-12* Δ' *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *proact-mono*:
assumes $\Delta \leq \Delta'$
shows *proact* Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *proact* Δ' *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

3.2 The definition of unwinding

definition *unwindCond* ::
(*enat* $\Rightarrow$ *'stateO* $\Rightarrow$ *'stateO* $\Rightarrow$ *status* $\Rightarrow$ *'stateV* $\Rightarrow$ *'stateV* $\Rightarrow$ *status* $\Rightarrow$ *bool*) $\Rightarrow$ *bool*
where
unwindCond $\Delta \equiv \forall$ *w s1 s2 statA sv1 sv2 statO.*
reachO s1 $\wedge$ *reachO s2* $\wedge$ *reachV sv1* $\wedge$ *reachV sv2* $\wedge$
Δ *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*))
$\wedge$
((∃ *v* $<$ *w. proact* Δ *v s1 s2 statA sv1 sv2 statO*)
$\vee$
match Δ *s1 s2 statA sv1 sv2 statO*

)

lemma *unwindCond-simpleI*:
assumes
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(finalO\ s1 \longleftrightarrow finalO\ s2) \wedge (finalV\ sv1 \longleftrightarrow finalO\ s1) \wedge (finalV\ sv2 \longleftrightarrow finalO\ s2)$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow statA = Eq$
$\Longrightarrow$
$isIntO\ s1 \longleftrightarrow isIntO\ s2$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$match\ \Delta\ s1\ s2\ statA\ sv1\ sv2\ statO$
shows $unwindCond\ \Delta$
⟨*proof*⟩

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

$\psi\ s1\ tr1\ s2\ tr2\ statO\ sv1\ trv1\ sv2\ trv2$
$\Longrightarrow completedFromV\ sv1\ trv1 \wedge completedFromV\ sv2\ trv2$
⟨*proof*⟩

lemma *completedFromO-lastt*: $completedFromO\ s1\ tr1 \Longrightarrow finalO\ (lastt\ s1\ tr1)$
⟨*proof*⟩

lemma *rsecure-strong*:
assumes
$\bigwedge s1\ tr1\ s2\ tr2.$
$istateO\ s1 \wedge Opt.validFromS\ s1\ tr1 \wedge completedFromO\ s1\ tr1 \wedge$
$istateO\ s2 \wedge Opt.validFromS\ s2\ tr2 \wedge completedFromO\ s2\ tr2 \wedge$
$Opt.A\ tr1 = Opt.A\ tr2 \wedge (isIntO\ s1 \wedge isIntO\ s2 \longrightarrow getActO\ s1 = getActO\ s2)$
$\Longrightarrow$
$\exists\, sv1\ trv1\ sv2\ trv2.$
$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$
$\psi\ s1\ tr1\ s2\ tr2\ Eq\ sv1\ trv1\ sv2\ trv2$
shows *rsecure*
⟨*proof*⟩

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

and *v*: *Opt.validFromS s1 tr1 Opt.completedFrom s1 tr1 Opt.validFromS s2 tr2 Opt.completedFrom s2 tr2*
and *tr14*: *Opt.A tr1* = *Opt.A tr2*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
shows ∃ *trv1 trv2*. *ψ s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2*
⟨*proof*⟩

theorem *unwind-rsecure*:
assumes *init*: *initCond* Δ
and *unwind*: *unwindCond* Δ
shows *rsecure*
⟨*proof*⟩

3.4 Compositional unwinding

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

definition *unwindIntoCond* ::
(*enat* ⇒ *'stateO* ⇒ *'stateO* ⇒ *status* ⇒ *'stateV* ⇒ *'stateV* ⇒ *status* ⇒ *bool*) ⇒
(*enat* ⇒ *'stateO* ⇒ *'stateO* ⇒ *status* ⇒ *'stateV* ⇒ *'stateV* ⇒ *status* ⇒ *bool*)
⇒ *bool*
where
unwindIntoCond Δ Δ′ ≡ ∀ *w s1 s2 statA sv1 sv2 statO*.
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Δ *w s1 s2 statA sv1 sv2 statO* ⟶
(*finalO s1* ⟷ *finalO s2*) ∧ (*finalV sv1* ⟷ *finalO s1*) ∧ (*finalV sv2* ⟷ *finalO s2*)

$\wedge$
$(statA = Eq \longrightarrow (isIntO\ s1 \longleftrightarrow isIntO\ s2))$
$\wedge$
$((\exists\, v{<}w.\ proact\ \Delta'\ v\ s1\ s2\ statA\ sv1\ sv2\ statO)$
$\vee$
$match\ \Delta'\ s1\ s2\ statA\ sv1\ sv2\ statO)$

lemma *unwindIntoCond-simpleI*:
assumes
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(finalO\ s1 \longleftrightarrow finalO\ s2) \wedge (finalV\ sv1 \longleftrightarrow finalO\ s1) \wedge (finalV\ sv2 \longleftrightarrow finalO\ s2)$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$
$statA = Eq$
$\Longrightarrow$
$isIntO\ s1 \longleftrightarrow isIntO\ s2$
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$match\ \Delta'\ s1\ s2\ statA\ sv1\ sv2\ statO$
shows $unwindIntoCond\ \Delta\ \Delta'$
$\langle proof \rangle$

lemma *unwindIntoCond-simpleI2*:
assumes
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(finalO\ s1 \longleftrightarrow finalO\ s2) \wedge (finalV\ sv1 \longleftrightarrow finalO\ s1) \wedge (finalV\ sv2 \longleftrightarrow finalO\ s2)$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$
$statA = Eq$
$\Longrightarrow$
$isIntO\ s1 \longleftrightarrow isIntO\ s2$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$

$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(\exists\, v{<}w.\ proact\ \Delta'\ v\ s1\ s2\ statA\ sv1\ sv2\ statO)$
shows *unwindIntoCond* $\Delta\ \Delta'$
⟨*proof*⟩

lemma *unwindIntoCond-simpleIB*:
assumes
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(finalO\ s1 \longleftrightarrow finalO\ s2) \wedge (finalV\ sv1 \longleftrightarrow finalO\ s1) \wedge (finalV\ sv2 \longleftrightarrow finalO\ s2)$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$
$statA = Eq$
$\Longrightarrow$
$isIntO\ s1 \longleftrightarrow isIntO\ s2$
and
$\bigwedge w\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(\exists\, v{<}w.\ proact\ \Delta'\ v\ s1\ s2\ statA\ sv1\ sv2\ statO) \vee match\ \Delta'\ s1\ s2\ statA\ sv1\ sv2\ statO$
shows *unwindIntoCond* $\Delta\ \Delta'$
⟨*proof*⟩

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

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

definition *oor* **where**
oor Δ $\Delta_2 \equiv \lambda w$ *s1 s2 statA sv1 sv2 statO*.
Δ *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w s1 s2 statA sv1 sv2 statO*

lemma *oorI1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor* Δ Δ_2 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oorI2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor* Δ Δ_2 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

definition *oor3* **where**
oor3 Δ Δ_2 $\Delta_3 \equiv \lambda w$ *s1 s2 statA sv1 sv2 statO*.
Δ *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w s1 s2 statA sv1 sv2 statO*

lemma *oor3I1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor3I2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor3I3*:
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor3* Δ Δ_2 Δ_3 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

definition *oor4* **where**
oor4 Δ Δ_2 Δ_3 $\Delta_4 \equiv \lambda w$ *s1 s2 statA sv1 sv2 statO*.
Δ *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_2 *w s1 s2 statA sv1 sv2 statO* $\vee$
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\vee$ Δ_4 *w s1 s2 statA sv1 sv2 statO*

lemma *oor4I1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor4I2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor4I3*:
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor4I4*:
Δ_4 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor4* Δ Δ_2 Δ_3 Δ_4 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

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

lemma *oor5I1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor5I2*:
Δ_2 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor5I3*:
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor5I4*:
Δ_4 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor5I5*:
Δ_5 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

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

lemma *oor6I1*:
Δ *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *oor6I3*:
Δ_3 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor6I4*:
Δ_4 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor6I5*:
Δ_5 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor6I6*:
Δ_6 *w s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor6* Δ Δ_2 Δ_3 Δ_4 Δ_5 Δ_6 *w s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *unwind-rsecure-foo*:
assumes *init*: *initCond* Δ_0
and *step0*: *unwindIntoCond* Δ_0 ΔNN
and *stepNN*: *unwindIntoCond* ΔNN (*oor5* ΔNN ΔSN ΔNS ΔSS $\Delta nonspec$)
and *stepNS*: *unwindIntoCond* ΔNS (*oor4* ΔNN ΔSN ΔNS ΔSS)
and *stepSN*: *unwindIntoCond* ΔSN (*oor4* ΔNN ΔSN ΔNS ΔSS)
and *stepSS*: *unwindIntoCond* ΔSS (*oor4* ΔNN ΔSN ΔNS ΔSS)
and *stepNonspec*: *unwindIntoCond* $\Delta nonspec$ $\Delta nonspec$
shows *rsecure*
⟨*proof*⟩

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

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

lemma *match1-1-oorI1*:
match1-1 Δ *s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$
match1-1 (*oor* Δ Δ_2) *s1 s1' s2 statA sv1 sv2 statO*

⟨*proof*⟩

lemma *match1-1-oorI2*:
match1-1 Δ_2 *s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$
match1-1 (*oor* Δ Δ_2) *s1 s1' s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match1-oorI1*:
match1 Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match1 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match1-oorI2*:
match1 Δ_2 *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match1 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-1-oorI1*:
match2-1 Δ *s1 s2 s2' statA sv1 sv2 statO* $\Longrightarrow$
match2-1 (*oor* Δ Δ_2) *s1 s2 s2' statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-1-oorI2*:
match2-1 Δ_2 *s1 s2 s2' statA sv1 sv2 statO* $\Longrightarrow$
match2-1 (*oor* Δ Δ_2) *s1 s2 s2' statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-oorI1*:
match2 Δ *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match2 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-oorI2*:
match2 Δ_2 *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match2 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

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

lemma *match12-oorI2*:
match12 Δ_2 *s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match12 (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*

⟨*proof*⟩

lemma *match12-1-oorI1*:
match12-1 Δ *s1′ s2′ statA′ sv1 sv2 statO* ⟹
match12-1 (*oor* Δ Δ_2) *s1′ s2′ statA′ sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-1-oorI2*:
match12-1 Δ_2 *s1′ s2′ statA′ sv1 sv2 statO* ⟹
match12-1 (*oor* Δ Δ_2) *s1′ s2′ statA′ sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-2-oorI1*:
match12-2 Δ *s2 s2′ statA′ sv1 sv2 statO* ⟹
match12-2 (*oor* Δ Δ_2) *s2 s2′ statA′ sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-2-oorI2*:
match12-2 Δ_2 *s2 s2′ statA′ sv1 sv2 statO* ⟹
match12-2 (*oor* Δ Δ_2) *s2 s2′ statA′ sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-12-oorI1*:
match12-12 Δ *s1′ s2′ statA′ sv1 sv2 statO* ⟹
match12-12 (*oor* Δ Δ_2) *s1′ s2′ statA′ sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-12-oorI2*:
match12-12 Δ_2 *s1′ s2′ statA′ sv1 sv2 statO* ⟹
match12-12 (*oor* Δ Δ_2) *s1′ s2′ statA′ sv1 sv2 statO*
⟨*proof*⟩

lemma *match-oorI1*:
match Δ *s1 s2 statA sv1 sv2 statO* ⟹
match (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match-oorI2*:
match Δ_2 *s1 s2 statA sv1 sv2 statO* ⟹
match (*oor* Δ Δ_2) *s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *proact-oorI1*:
proact Δ *meas s1 s2 statA sv1 sv2 statO* ⟹
proact (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*

⟨*proof*⟩

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

lemma *move-1-oorI1*:
move-1 Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-1 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-1-oorI2*:
move-1 Δ_2 *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-1 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-2-oorI1*:
move-2 Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-2 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-2-oorI2*:
move-2 Δ_2 *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-2 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-12-oorI1*:
move-12 Δ *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-12 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-12-oorI2*:
move-12 Δ_2 *meas s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-12 (*oor* Δ Δ_2) *meas s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

end

context *Relative-Security-Determ*
begin

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

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

lemma *match12-2-defD*: *match12-2* Δ *s1' s2' statA' sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ $\Delta \infty$ *s1' s2' statA' sv1* (*nextO sv2*) *statO*
⟨*proof*⟩

lemma *match12-12-defD*: *match12-12* Δ *s1' s2' statA' sv1 sv2 statO* $\longleftrightarrow$
(*let statO'* = *sstatO' statO sv1 sv2 in*
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$
(*statA'* = *Diff* $\longrightarrow$ *statO'* = *Diff*) $\wedge$
$\Delta \infty$ *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* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ Δ *w s1 s2 statA* (*nextO sv1*) *sv2 statO*
⟨*proof*⟩

lemma *move-2-defD*: *move-2* Δ *w s1 s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ Δ *w s1 s2 statA sv1* (*nextO sv2*) *statO*
⟨*proof*⟩

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

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

lemma *updStat-EqI*:
assumes ‹*R* = *S*›
shows ‹*updStat Eq* (*P*, *R*) (*Q*, *S*) = *Eq*›
⟨*proof*⟩

lemma *updStat-diff*:*updStat stat r r* = *Diff* ⟹ *stat* = *Diff*
⟨*proof*⟩

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

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

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

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

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

lemma *match1-1-mono*:
Δ ≤ Δ' ⟹ *match1-1* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* ⟹
match1-1 Δ' *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match1-12-mono*:
Δ ≤ Δ' ⟹ *match1-12* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* ⟹
match1-12 Δ' *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match1-mono*:

assumes $\Delta \leq \Delta'$
shows *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match1* Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

definition *match2-1* Δ *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* Δ *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\equiv$
$\exists$ *sv1' sv2'.*
let statO' = *sstatO' statO sv1 sv2 in*
validTransV (*sv1*,*sv1'*) $\wedge$
validTransV (*sv2*,*sv2'*) $\wedge$
$\Delta \infty$ *w1 w2 s1 s2' statA sv1' sv2' statO'*

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

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

lemma *match2-1-mono*:
$\Delta \leq \Delta'$ $\Longrightarrow$ *match2-1* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2-1* Δ' *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-12-mono*:
$\Delta \leq \Delta'$ $\Longrightarrow$ *match2-12* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2-12* Δ' *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-mono*:
assumes $\Delta \leq \Delta'$
shows *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match2* Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

definition *match12-1* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* $\equiv$
$\exists$ *sv1′. validTransV* (*sv1*,*sv1′*) $\wedge$
$\Delta \infty$ *w1 w2 s1′ s2′ statA′ sv1′ sv2 statO*

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

definition *match12-12* Δ *w1 w2 s1′ s2′ statA′ sv1 sv2 statO* $\equiv$
$\exists$ *sv1′ sv2′.*
let statO′ = *sstatO′ statO sv1 sv2 in*
validTransV (*sv1*,*sv1′*) $\wedge$
validTransV (*sv2*,*sv2′*) $\wedge$
(*statA′* = *Diff* $\longrightarrow$ *statO′* = *Diff*) $\wedge$
$\Delta \infty$ *w1 w2 s1′ s2′ statA′ sv1′ sv2′ statO′*

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

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

lemma *match12-simpleI*:
assumes $\bigwedge$*s1′ s2′ statA′.*
statA′ = *sstatA′ statA s1 s2* $\Longrightarrow$
validTransO (*s1*,*s1′*) $\Longrightarrow$
validTransO (*s2*,*s2′*) $\Longrightarrow$
Opt.eqAct s1 s2 $\Longrightarrow$
($\exists$ *w1′*$<$ *w1*. $\exists$ *w2′*$<$ *w2*. $\neg$ *isSecO s1* $\wedge$ $\neg$ *isSecO s2* $\wedge$ (*statA* = *statA′* $\vee$ *statO*

$= Diff) \wedge$
$\Delta \infty\ w1'\ w2'\ s1'\ s2'\ statA'\ sv1\ sv2\ statO)$
$\vee$
$(eqSec\ sv1\ s1 \wedge eqSec\ sv2\ s2 \wedge Van.eqAct\ sv1\ sv2 \wedge$
$match12\text{-}12\ \Delta \infty \infty\ s1'\ s2'\ statA'\ sv1\ sv2\ statO)$
shows *match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-1-mono*:
$\Delta \leq \Delta' \Longrightarrow match12\text{-}1\ \Delta\ w1\ w2\ s1'\ s2'\ statA'\ sv1\ sv2\ statO \Longrightarrow match12\text{-}1\ \Delta'$ *w1 w2 s1' s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-2-mono*:
$\Delta \leq \Delta' \Longrightarrow match12\text{-}2\ \Delta\ w1\ w2\ s1\ s2'\ statA'\ sv1\ sv2\ statO \Longrightarrow match12\text{-}2\ \Delta'$ *w1 w2 s1 s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-12-mono*:
$\Delta \leq \Delta' \Longrightarrow match12\text{-}12\ \Delta\ w1\ w2\ s1'\ s2'\ statA'\ sv1\ sv2\ statO \Longrightarrow match12\text{-}12$ Δ' *w1 w2 s1' s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-mono*:
assumes $\Delta \leq \Delta'$
shows *match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match12* Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

definition *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\equiv$
match1 Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
$\wedge$
match2 Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
$\wedge$
match12 Δ *w1 w2 s1 s2 statA sv1 sv2 statO*

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

lemma *match-mono*:
assumes $\Delta \leq \Delta'$
shows *match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *match* Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

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

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

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

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

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

lemma *move-1-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-1* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-1* Δ' *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-2-mono*:
$\Delta \leq \Delta' \Longrightarrow$ *move-2* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *move-2* Δ' *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

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

lemma *proact-mono*:
assumes $\Delta \leq \Delta'$
shows *proact* Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *proact* Δ' *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

4.2 The definition of unwinding

definition *unwindCond* ::
(*enat* $\Rightarrow$ *enat* $\Rightarrow$ *enat* $\Rightarrow$ *'stateO* $\Rightarrow$ *'stateO* $\Rightarrow$ *status* $\Rightarrow$ *'stateV* $\Rightarrow$ *'stateV* $\Rightarrow$

status ⇒ *bool*) ⇒ *bool*
where
unwindCond Δ ≡ ∀ *w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟶
(*finalO s1* ⟷ *finalO s2*) ∧ (*finalV sv1* ⟷ *finalO s1*) ∧ (*finalV sv2* ⟷ *finalO s2*)
∧
(*statA* = *Eq* ⟶ (*isIntO s1* ⟷ *isIntO s2*))
∧
((∃ *v* < *w*. *proact* Δ *v w1 w2 s1 s2 statA sv1 sv2 statO*)
∨
match Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
)

lemma *unwindCond-simpleI*:
assumes
⋀*w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 ⟹ *reachO s2* ⟹ *reachV sv1* ⟹ *reachV sv2* ⟹
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟹
(*finalO s1* ⟷ *finalO s2*) ∧ (*finalV sv1* ⟷ *finalO s1*) ∧ (*finalV sv2* ⟷ *finalO s2*)
and
⋀*w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 ⟹ *reachO s2* ⟹ *reachV sv1* ⟹ *reachV sv2* ⟹
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* ⟹ *statA* = *Eq*
⟹
isIntO s1 ⟷ *isIntO s2*
and
⋀*w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 ⟹ *reachO s2* ⟹ *reachV sv1* ⟹ *reachV sv2* ⟹
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟹
match Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
shows *unwindCond* Δ
⟨*proof*⟩

4.3 The soundness of unwinding

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

definition *ψ s1 tr1 s2 tr2 statO sv1 trv1 sv2 trv2* ≡
trv1 ≠ [] ∧ *trv2* ≠ [] ∧
Van.validFromS sv1 trv1 ∧

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

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

$\psi\ s1\ tr1\ s2\ tr2\ statO\ sv1\ trv1\ sv2\ trv2$
$\Longrightarrow completedFromV\ sv1\ trv1 \wedge completedFromV\ sv2\ trv2$
⟨*proof*⟩

lemma *completedFromO-lastt*: $completedFromO\ s1\ tr1 \Longrightarrow finalO\ (lastt\ s1\ tr1)$
⟨*proof*⟩

lemma *rsecure-strong*:
assumes
$\bigwedge s1\ tr1\ s2\ tr2.$
$istateO\ s1 \wedge Opt.validFromS\ s1\ tr1 \wedge completedFromO\ s1\ tr1 \wedge$
$istateO\ s2 \wedge Opt.validFromS\ s2\ tr2 \wedge completedFromO\ s2\ tr2 \wedge$
$Opt.A\ tr1 = Opt.A\ tr2$
$\Longrightarrow$
$\exists\, sv1\ trv1\ sv2\ trv2.$
$istateV\ sv1 \wedge istateV\ sv2 \wedge corrState\ sv1\ s1 \wedge corrState\ sv2\ s2 \wedge$
$\psi\ s1\ tr1\ s2\ tr2\ Eq\ sv1\ trv1\ sv2\ trv2$
shows *rsecure*
⟨*proof*⟩

proposition *unwindCond-ex-ψ*:
assumes *unwind*: $unwindCond\ \Delta$
and Δ: $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$ **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 $\exists\, trv1\ trv2.\ \psi\ s1\ tr1\ s2\ tr2\ statO\ sv1\ trv1\ sv2\ trv2$
⟨*proof*⟩

lemma *unwindCond-final*:
$unwindCond\ \Delta \Longrightarrow reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow \Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$
$(finalV\ sv1 \longleftrightarrow finalO\ s1) \wedge (finalV\ sv2 \longleftrightarrow finalO\ s2)$

⟨*proof*⟩

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 *φ-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 *φ*: $\varphi\ \Delta\ w\ w1\ w2\ w1'\ w2'\ statA\ s1\ tr1\ s2\ tr2\ statAA\ statO\ sv1\ trv1\ sv2\ trv2\ statOO$
shows $(finalV\ (lastt\ sv1\ trv1) \longleftrightarrow finalO\ (lastt\ s1\ tr1)) \wedge (finalV\ (lastt\ sv2\ trv2) \longleftrightarrow finalO\ (lastt\ s2\ tr2))$
⟨*proof*⟩

lemma *φ-completedFrom*: $unwindCond\ \Delta \Longrightarrow$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$Opt.validFromS\ s1\ tr1 \Longrightarrow completedFromO\ s1\ tr1 \Longrightarrow$
$Opt.validFromS\ s2\ tr2 \Longrightarrow completedFromO\ s2\ tr2 \Longrightarrow$
$\varphi\ \Delta\ statA\ w\ w1\ w2\ w1'\ w2'\ s1\ tr1\ s2\ tr2\ statAA\ statO\ sv1\ trv1\ sv2\ trv2\ statOO$
$\Longrightarrow completedFromV\ sv1\ trv1 \wedge completedFromV\ sv2\ trv2$
⟨*proof*⟩

lemma *unwindCond-ex-φ*:
assumes *unwind*: $unwindCond\ \Delta$
and *Δ*: $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
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 $\exists w'\ w1'\ w2'\ trv1\ trv2\ statAA\ statOO.\ \varphi\ \Delta\ w'\ w1\ w2\ w1'\ w2'\ statA\ s1\ tr1\ s2\ tr2\ statAA\ statO\ sv1\ trv1\ sv2\ trv2\ statOO$
⟨*proof*⟩

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

lemma *unwindCond-ex-φa-getActO*:
assumes *unwind*: $unwindCond\ \Delta$
and Δ: $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
and *r34*: $reachO\ s1\ reachO\ s2$ **and** *r12*: $reachV\ sv1\ reachV\ sv2$
and *stat*: $(statA = Diff \longrightarrow statO = Diff)$
and *v*: $validTransO\ (s1,\ s1')\ validTransO\ (s2,\ s2')$
and *i34*: $isIntO\ s1\ isIntO\ s2\ getActO\ s1 = getActO\ s2$
shows $\exists\, w1'\ w2'\ trv1\ trv2\ statOO.$
$\varphi a\ \Delta\ \infty\ w1\ w2\ w1'\ w2'\ statA\ s1\ [s1,\ s1']\ s2\ [s2,\ s2']\ (sstatA'\ statA\ s1\ s2)\ statO\ sv1\ trv1\ sv2\ trv2\ statOO$
⟨*proof*⟩

lemma *unwindCond-ex-φa'-aux*:
assumes *unwind*: $unwindCond\ \Delta$
and Δ: $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
and *r*: $reachO\ s1\ reachO\ s2\ reachV\ sv1\ reachV\ sv2$
and *stat*: $(statA = Diff \longrightarrow statO = Diff)$
and *tr14NE*: $tr1 \neq []\ tr2 \neq []$
and *v3'*: $Opt.validFromS\ s1\ (tr1\ \#\#\ s1')$ **and** *v4'*: $Opt.validFromS\ s2\ (tr2\ \#\#\ s2')$
and *i*: $isIntO\ (lastt\ s1\ tr1)\ isIntO\ (lastt\ s2\ tr2)$
and *A34*: $getActO\ (lastt\ s1\ tr1) = getActO\ (lastt\ s2\ tr2)$
and *nev*: $never\ isIntO\ (butlast\ tr1)\ never\ isIntO\ (butlast\ tr2)$
shows $\exists\, w1'\ w2'\ trv1'\ trv2'\ statAA'\ statOO'.$
$\varphi a\ \Delta\ \infty\ w1\ w2\ w1'\ w2'\ statA\ s1\ (tr1\ \#\#\ s1')\ s2\ (tr2\ \#\#\ s2')\ statAA'\ statO\ sv1\ trv1'\ sv2\ trv2'\ statOO'$
⟨*proof*⟩

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

and *stat*: (*statA* = *Diff* ⟶ *statO* = *Diff*)
and *v3'*: *Opt.validFromS s1* (*tr1* @ [*s1'*,*s1''*]) **and** *v4'*: *Opt.validFromS s2* (*tr2* @ [*s2'*,*s2''*])
and *i*: *isIntO s1' isIntO s2'*
and *A34*: *getActO s1'* = *getActO s2'*
and *nev*: *never isIntO tr1 never isIntO tr2*
shows ∃ *w1' w2' trv1 trv2 statAA statOO*.
φ*a* Δ ∞ *w1 w2 w1' w2' statA s1* (*tr1* @ [*s1'*,*s1''*]) *s2* (*tr2* @ [*s2'*,*s2''*]) *statAA statO sv1 trv1 sv2 trv2 statOO*
⟨*proof*⟩

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

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

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

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

proposition *unwindCond-ex-φ'*:
assumes *unwind*: *unwindCond* Δ **and** Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *v3'*: *Opt.validFromS s1* ((*tr1* ## *s1'*) ## *s1''*) **and** *v4'*: *Opt.validFromS s2* ((*tr2* ## *s2'*) ## *s2''*)
and *i*: *isIntO s1' isIntO s2'*
and *A34*: *getActO s1'* = *getActO s2'*
and *nev*: *never isIntO tr1 never isIntO tr2*
shows ∃ *w1' w2' trv1 sv1'' trv2 sv2'' statAA statOO*.
φ' Δ *w1 w2 w1' w2' statA s1 tr1 s1' s1'' s2 tr2 s2' s2'' statAA statO sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
⟨*proof*⟩

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

lemma $\chi 3\text{-}final$:
assumes unw: $unwindCond\ \Delta$
and r: $reachO\ s1\ reachO\ s2\ reachV\ sv1\ reachV\ sv2$
and $vtr1$: $Opt.validFromS\ s1\ tr1$
and $\chi 3$: $\chi 3\ \Delta\ w\ w1\ w2\ w1'\ w2'\ s1\ tr1\ s2\ statAA\ sv1\ trv1\ sv2\ trv2\ statOO$
shows $(finalV\ (lastt\ sv1\ trv1) \longleftrightarrow finalO\ (lastt\ s1\ tr1)) \wedge (finalV\ (lastt\ sv2\ trv2) \longleftrightarrow finalO\ s2)$
⟨*proof*⟩

lemma $\chi 3\text{-}completedFrom$: $unwindCond\ \Delta \Longrightarrow$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$Opt.validFromS\ s1\ tr1 \Longrightarrow completedFromO\ s1\ tr1 \Longrightarrow$
$\chi 3\ \Delta\ w\ w1\ w2\ w1'\ w2'\ s1\ tr1\ s2\ statAA\ sv1\ trv1\ sv2\ trv2\ statOO$
$\Longrightarrow completedFromV\ sv1\ trv1 \wedge completedFromV\ sv2\ trv2$
⟨*proof*⟩

lemma $unwindCond\text{-}ex\text{-}\chi 3$:
assumes $unwind$: $unwindCond\ \Delta$
and Δ: $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
and r: $reachO\ s1\ reachO\ s2\ reachV\ sv1\ reachV\ sv2$
and $vtr1$: $Opt.validFromS\ s1\ tr1$
and $nis1$: $\neg\ isIntO\ s1$ **and** $nis2$: $\neg\ isIntO\ s2$
and $inter3$: $never\ isIntO\ tr1$
and sec: $never\ isSecO\ (butlast\ tr1)\ isSecO\ (lastt\ s1\ tr1)$
shows $\exists\, w'\ w1'\ w2'\ trv1\ trv2\ statOO.\ \chi 3\ \Delta\ w'\ w1\ w2\ w1'\ w2'\ s1\ tr1\ s2\ statA\ sv1\ trv1\ sv2\ trv2\ statOO$
⟨*proof*⟩

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

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

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

lemma *unwindCond-ex-χ3b-aux*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* **and**
r: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *tr1NE*: *tr1* ≠ []
and *v3'*: *Opt.validFromS s1* (*tr1* ## *s1'*)
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter3'*: *never isIntO* (*tr1* ## *s1'*)
and *sec*: *never isSecO* (*butlast tr1*) *isSecO* (*lastt s1 tr1*)
shows ∃ *w1' w2' trv1 trv2 statOO*. *χ3b* Δ ∞ *w1 w2 w1' w2' s1* (*tr1* ## *s1'*) *s2 statA sv1 trv1 sv2 trv2 statOO*
⟨*proof*⟩

lemma *unwindCond-ex-χ3b-aux2*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *v3'*: *Opt.validFromS s1* (*tr1* @ [*s1'*,*s1''*])
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter3'*: *never isIntO* (*tr1* @ [*s1'*,*s1''*])
and *sec*: *never isSecO tr1 isSecO s1'*
shows ∃ *w1' w2' trv1 trv2 statOO*. *χ3b* Δ ∞ *w1 w2 w1' w2' s1* (*tr1* @ [*s1'*,*s1''*]) *s2 statA sv1 trv1 sv2 trv2 statOO*
⟨*proof*⟩

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* Δ
and Δ: $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$ **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 $\exists\ w1'\ w2'\ trv1\ sv1''\ trv2\ sv2''\ statOO.\ \chi 3'\ \Delta\ w1\ w2\ w1'\ w2'\ s1\ tr1\ s1'\ s1''\ s2\ statA\ sv1\ trv1\ sv1''\ sv2\ trv2\ sv2''\ statOO$
⟨*proof*⟩

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

proposition *unwindCond-ex-*$\omega 3$:
assumes *unwind*: *unwindCond* Δ
and Δ: $\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
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 $\exists\ w1'\ w2'\ trv1\ sv1'\ trv2\ sv2'\ statOO.\ \omega 3\ \Delta\ w1\ w2\ w1'\ w2'\ s1\ s1'\ s2\ statA\ sv1\ trv1\ sv1'\ sv2\ trv2\ sv2'\ statOO$
⟨*proof*⟩

definition $\chi 4\ \Delta\ w\ w1\ (w2::enat)\ w1'\ w2'\ s1\ s2\ tr2\ statAA\ sv1\ trv1\ sv2\ trv2\ statOO \equiv$
$trv1 \neq [] \wedge trv2 \neq [] \wedge (length\ trv1 > Suc\ 0 \vee w1' \leq w1) \wedge$

Van.validFromS sv1 trv1 $\wedge$ *Van.validFromS sv2 trv2* $\wedge$
never isSecV (*butlast trv1*) $\wedge$
never isSecV (*butlast trv2*) $\wedge$
isSecV (*lastt sv2 trv2*) $\wedge$ *getSecV* (*lastt sv2 trv2*) = *getSecO* (*lastt s2 tr2*) $\wedge$
Van.A trv1 = *Van.A trv2* $\wedge$
Δ *w w1' w2' s1* (*lastt s2 tr2*) *statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *χ4-final*:
assumes *unw*: *unwindCond* Δ
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *vtr2*: *Opt.validFromS s2 tr2*
and *χ4*: *χ4* Δ *w w1 w2 w1' w2' s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO*
shows (*finalV* (*lastt sv1 trv1*) $\longleftrightarrow$ *finalO s1*) $\wedge$ (*finalV* (*lastt sv2 trv2*) $\longleftrightarrow$ *finalO* (*lastt s2 tr2*))
⟨*proof*⟩

lemma *χ4-completedFrom*: *unwindCond* Δ $\Longrightarrow$
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Opt.validFromS s2 tr2 $\Longrightarrow$ *completedFromO s2 tr2* $\Longrightarrow$
χ4 Δ *w w1 w2 w1' w2' s1 s2 tr2 statAA sv1 trv1 sv2 trv2 statOO*
$\Longrightarrow$ *completedFromV sv1 trv1* $\wedge$ *completedFromV sv2 trv2*
⟨*proof*⟩

proposition *unwindCond-ex-χ4*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *vtr2*: *Opt.validFromS s2 tr2*
and *nis2*: $\neg$ *isIntO s1* **and** *nis2*: $\neg$ *isIntO s2*
and *inter4*: *never isIntO tr2*
and *sec*: *never isSecO* (*butlast tr2*) *isSecO* (*lastt s2 tr2*)
shows $\exists$ *w' w1' w2' trv1 trv2 statOO*. *χ4* Δ *w' w1 w2 w1' w2' s1 s2 tr2 statA sv1 trv1 sv2 trv2 statOO*
⟨*proof*⟩

definition *χ4a* **where** *χ4a* Δ *w w1* (*w2*::*enat*) *w1' w2' s1 s2 s2' statAA sv1 trv1 sv2 trv2 statOO* ≡
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* $\wedge$
Δ *w w1' w2' s1 s2' statAA* (*lastt sv1 trv1*) (*lastt sv2 trv2*) *statOO*

lemma *unwindCond-ex-χ4a-getSec*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r34*: *reachO s1 reachO s2* **and** *r12*: *reachV sv1 reachV sv2*
and *v*: *validTransO* (*s2*, *s2'*)

and *ii4*: ¬ *isIntO s2*
and *is2*: *isSecO s2* **and** *isv24*: *isSecV sv2 getSecO s2 = getSecV sv2*
shows ∃ *w1′ w2′ trv1 trv2 statOO*.
χ4a Δ ∞ *w1 w2 w1′ w2′ s1 s2 s2′ statA sv1 trv1 sv2 trv2 statOO*
⟨*proof*⟩

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

lemma *unwindCond-ex-χ4b-aux*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *tr2NE*: *tr2* ≠ []
and *v4′*: *Opt.validFromS s2* (*tr2* ## *s2′*)
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter4′*: *never isIntO* (*tr2* ## *s2′*)
and *sec*: *never isSecO* (*butlast tr2*) *isSecO* (*lastt s2 tr2*)
shows ∃ *w1′ w2′ trv1 trv2 statOO*. *χ4b* Δ ∞ *w1 w2 w1′ w2′ s1 s2* (*tr2* ## *s2′*) *statA sv1 trv1 sv2 trv2 statOO*
⟨*proof*⟩

lemma *unwindCond-ex-χ4b-aux2*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* **and**
r: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *v4′*: *Opt.validFromS s2* (*tr2* @ [*s2′*,*s2′′*])
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter4′*: *never isIntO* (*tr2* @ [*s2′*,*s2′′*])
and *sec*: *never isSecO tr2 isSecO s2′*
shows ∃ *w1′ w2′ trv1 trv2 statOO*. *χ4b* Δ ∞ *w1 w2 w1′ w2′ s1 s2* (*tr2* @ [*s2′*,*s2′′*]) *statA sv1 trv1 sv2 trv2 statOO*
⟨*proof*⟩

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

$\Delta \infty\ w1'\ w2'\ s1\ s2''\ statAA\ sv1''\ sv2''\ statOO$

proposition *unwindCond-ex-χ4'*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* **and**
r: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *v4'*: *Opt.validFromS s2* ((*tr2* ## *s2'*) ## *s2''*)
and *nis1*: ¬ *isIntO s1* **and** *nis2*: ¬ *isIntO s2*
and *ninter4'*: *never isIntO* ((*tr2* ## *s2'*) ## *s2''*)
and *sec*: *never isSecO tr2 isSecO s2'*
shows ∃ *w1' w2' trv1 sv1'' trv2 sv2'' statOO*. *χ4'* Δ *w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
⟨*proof*⟩

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

proposition *unwindCond-ex-ω4*:
assumes *unwind*: *unwindCond* Δ
and Δ: Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
and *r34*: *reachO s1 reachO s2* **and** *r12*: *reachV sv1 reachV sv2*
and *nis1*: ¬ *isIntO s1*
and *v4*: *validTransO* (*s2*,*s2'*)
and *nis2*: ¬ *isIntO s2* ¬ *isIntO s2'* ¬ *isSecO s2*
shows ∃ *w1' w2' trv1 sv1' trv2 sv2' statOO*. *ω4* Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO*
⟨*proof*⟩

definition *φφ s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'* ≡
ltr1 = *lappend* (*llist-of* (*tr1* ## *s1'*)) (*s1''* $ *ltr1'*) ∧
ltr2 = *lappend* (*llist-of* (*tr2* ## *s2'*)) (*s2''* $ *ltr2'*) ∧
Opt.validFromS s1 ((*tr1* ## *s1'*) ## *s1''*) ∧ *Opt.validFromS s2* ((*tr2* ## *s2'*) ## *s2''*) ∧

never isIntO tr1 $\wedge$ *never isIntO tr2* $\wedge$
isIntO s1' $\wedge$ *isIntO s2'* $\wedge$ *getActO s1'* = *getActO s2'* $\wedge$
Opt.lvalidFromS s1'' (*s1''* $ *ltr1'*) $\wedge$ *Opt.lcompletedFrom s1''* (*s1''* $ *ltr1'*) $\wedge$
Opt.lvalidFromS s2'' (*s2''* $ *ltr2'*) $\wedge$ *Opt.lcompletedFrom s2''* (*s2''* $ *ltr2'*) $\wedge$
Opt.lA (*s1''* $ *ltr1'*) = *Opt.lA* (*s2''* $ *ltr2'*)

lemma *isIntO-φφ*:
assumes *vltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *vltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A*: *Opt.lA ltr1* = *Opt.lA ltr2* **and** *inter3*: ¬ *lnever isIntO ltr1*
shows ∃ *tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'*. *φφ s1 ltr1 s2 ltr2 tr1 s1' s1'' ltr1' tr2 s2' s2'' ltr2'*
⟨*proof*⟩

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

lemma *isSecO-χχ*:
assumes *vltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *inter*: *lnever isIntO ltr1* **and** *isec*: ¬ *lnever isSecO ltr1*
shows ∃ *tr1 s1' s1'' ltr1'*. *χχ s1 ltr1 tr1 s1' s1'' ltr1'*
⟨*proof*⟩

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

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

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

fun *isn* :: *turn* × ('*stateO*,'*stateV*) *tuple34* ⇒ *bool*

where
isn (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) ⟷ *ltr1* = [[]] ∧ *ltr2* = [[]]

fun *h-t* ::
turn × (′*stateO*,′*stateV*)*tuple34* ⇒
(′*stateO*,′*stateV*)*tuple12* ×
turn × (′*stateO*,′*stateV*)*tuple34*
where
h-t (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
(*if trn* = *L*
then if lnever isSecO ltr1
then let (*s1*′,*ltr1*′) = (*lhd* (*ltl ltr1*), *ltl ltr1*)
in let (*w1*′,*w2*′,*trv1*,*sv1*′,*trv2*,*sv2*′,*statOO*) =
(*SOME k. case k of* (*w1*′,*w2*′,*trv1*,*sv1*′,*trv2*,*sv2*′,*statOO*) ⇒
ω3 Δ *w1 w2 w1*′ *w2*′ *s1 s1*′ *s2 statA sv1 trv1 sv1*′ *sv2 trv2 sv2*′ *statOO*)
in ((*trv1*,*sv1*′,*trv2*,*sv2*′,*statA*,*statOO*),
(*if trv1* = [] *then L else R*,
w1′,*w2*′,*s1*′,*ltr1*′,*s2*,*ltr2*,*statA*,*sv1*′,*sv2*′,*statOO*))
else
let (*tr1*,*s1*′,*s1*″,*ltr1*′) =
(*SOME k. case k of* (*tr1*,*s1*′,*s1*″,*ltr1*′) ⇒
χχ s1 ltr1 tr1 s1′ *s1*″ *ltr1*′)
in let (*w1*′,*w2*′,*trv1*,*sv1*″,*trv2*,*sv2*″,*statOO*) =
(*SOME k*′. *case k*′ *of* (*w1*′,*w2*′,*trv1*,*sv1*″,*trv2*,*sv2*″,*statOO*) ⇒
χ3′ Δ *w1 w2 w1*′ *w2*′ *s1 tr1 s1*′ *s1*″ *s2 statA sv1 trv1 sv1*″ *sv2 trv2*
sv2″ *statOO*)
in ((*trv1*,*sv1*″,*trv2*,*sv2*″,*statA*,*statOO*),
(*R*,*w1*′,*w2*′,*s1*″,*s1*″ $ *ltr1*′,*s2*,*ltr2*,*statA*,*sv1*″,*sv2*″,*statOO*))
—
else if lnever isSecO ltr2
then let (*s2*′,*ltr2*′) = (*lhd* (*ltl ltr2*), *ltl ltr2*)
in let (*w1*′,*w2*′,*trv1*,*sv1*′,*trv2*,*sv2*′,*statOO*) =
(*SOME k. case k of* (*w1*′,*w2*′,*trv1*,*sv1*′,*trv2*,*sv2*′,*statOO*) ⇒
ω4 Δ *w1 w2 w1*′ *w2*′ *s1 s2 s2*′ *statA sv1 trv1 sv1*′ *sv2 trv2 sv2*′ *statOO*)
in ((*trv1*,*sv1*′,*trv2*,*sv2*′,*statA*,*statOO*),
(*if trv2* = [] *then R else L*,
w1′,*w2*′,*s1*,*ltr1*,*s2*′,*ltr2*′,*statA*,*sv1*′,*sv2*′,*statOO*))
else
let (*tr2*,*s2*′,*s2*″,*ltr2*′) =
(*SOME k. case k of* (*tr2*,*s2*′,*s2*″,*ltr2*′) ⇒
χχ s2 ltr2 tr2 s2′ *s2*″ *ltr2*′)
in let (*w1*′,*w2*′,*trv1*,*sv1*″,*trv2*,*sv2*″,*statOO*) =
(*SOME k*′. *case k*′ *of* (*w1*′,*w2*′,*trv1*,*sv1*″,*trv2*,*sv2*″,*statOO*) ⇒
χ4′ Δ *w1 w2 w1*′ *w2*′ *s1 s2 tr2 s2*′ *s2*″ *statA sv1 trv1 sv1*″ *sv2 trv2*
sv2″ *statOO*)
in ((*trv1*,*sv1*″,*trv2*,*sv2*″,*statA*,*statOO*),
(*L*,*w1*′,*w2*′,*s1*, *ltr1*, *s2*″,*s2*″ $ *ltr2*′,*statA*,*sv1*″,*sv2*″,*statOO*))
)

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

definition $h \equiv fst\ o\ h\text{-}t$
definition $t \equiv snd\ o\ h\text{-}t$

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

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

definition *f* :: *turn* × (*'stateO*,*'stateV*)*tuple34* ⇒ (*'stateO*,*'stateV*)*tuple12 llist*
where *f* ≡ *ccorec-llist isn h econd e t*

lemma *f-LNil*:
ltr1 = [[]] ⟹ *ltr2* = [[]] ⟹ *f* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
[[]]
⟨*proof*⟩

lemma *f-length-1*:
assumes *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] *llength ltr1* ≤ *Suc 0* ∨ *llength ltr2* ≤ *Suc 0*
shows *f* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[([*sv1*],*sv1*,[*sv2*],*sv2*,*statA*,*statO*)]]

⟨*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*,*sta
))
⟨*proof*⟩

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

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

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

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

lemma *lltrv1-ne-imp*:
assumes *lltrv1 trn-tp* $\neq$ [[]]
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 = [[]] $\Longrightarrow$ *ltr2* = [[]] $\Longrightarrow$ *lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
= [[]]
⟨*proof*⟩
lemma *lltrv2-LNil*[*simp*]:
ltr1 = [[]] $\Longrightarrow$ *ltr2* = [[]] $\Longrightarrow$ *lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
= [[]]
⟨*proof*⟩

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*]]
⟨*proof*⟩

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*]]
⟨*proof*⟩

lemma *h-t-lnever-L*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *l'*: *lnever isIntO ltr1* $\neg$ *isIntO s2*

and *len*: *llength ltr1 > Suc 0 llength ltr2 > Suc 0*
and *l*: *trn = L lnever isSecO ltr1*
shows ∃ *w1′ w2′ s1′ ltr1′ trv1 sv1′ trv2 sv2′ statOO*.
ltr1 = s1 $ ltr1′ ∧ validTransO (s1,s1′) ∧
Opt.lvalidFromS s1′ ltr1′ ∧ Opt.lcompletedFrom s1′ ltr1′ ∧ lnever isIntO ltr1′ ∧

ω3 Δ w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO ∧
h-t (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
((trv1,sv1′,trv2,sv2′,statA,statOO),
(if trv1 = [] then L else R,
w1′,w2′, s1′, ltr1′,s2,ltr2,statA,sv1′,sv2′,statOO))
⟨*proof*⟩

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

ω3 Δ w1 w2 w1′ w2′ s1 s1′ s2 statA sv1 trv1 sv1′ sv2 trv2 sv2′ statOO ∧
lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
lappend (llist-of trv1) (lltrv1 (if trv1 = [] then L else R,
w1′,w2′,s1′,ltr1′,s2,ltr2,statA,sv1′,sv2′,statOO)) ∧
lltrv2 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
lappend (llist-of trv2) (lltrv2 (if trv1 = [] then L else R,
w1′,w2′,s1′,ltr1′,s2,ltr2,statA,sv1′,sv2′,statOO))
⟨*proof*⟩

lemma *h-t-not-lnever-L*:
assumes *unw*: *unwindCond Δ*
and *Δ*: *Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *l′*: *lnever isIntO ltr1 ¬ isIntO s2*
and *len*: *llength ltr1 > Suc 0 llength ltr2 > Suc 0*
and *l*: *trn = L ¬ lnever isSecO ltr1*
shows ∃ *w1′ w2′ tr1 s1′ s1″ ltr1′ trv1 sv1″ trv2 sv2″ statOO*.
χχ s1 ltr1 tr1 s1′ s1″ ltr1′ ∧
χ3′ Δ w1 w2 w1′ w2′ s1 tr1 s1′ s1″ s2 statA sv1 trv1 sv1″ sv2 trv2 sv2″ statOO
∧
h-t (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =

((*trv1*,*sv1''*,*trv2*,*sv2''*,*statA*,*statOO*),
(*R*,*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2*,*ltr2*,*statA*,*sv1''*,*sv2''*,*statOO*))
⟨*proof*⟩

lemma *lltrv1-lltrv2-not-lnever-L*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *l'*: *lnever isIntO ltr1* ¬ *isIntO s2*
and *len*: *llength ltr1* > *Suc 0 llength ltr2* > *Suc 0*
and *l*: *trn* = *L* ¬ *lnever isSecO ltr1*
shows ∃ *w1' w2' tr1 s1' s1'' ltr1' trv1 sv1'' trv2 sv2'' statOO*.
χχ *s1 ltr1 tr1 s1' s1'' ltr1'* ∧
χ3' Δ *w1 w2 w1' w2' s1 tr1 s1' s1'' s2 statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO*
∧
lltrv1 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv1*) (*lltrv1* (*R*,*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2*,*ltr2*,*statA*,*sv1''*,*sv2''*,*statOO*))
∧
lltrv2 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv2*) (*lltrv2* (*R*,*w1'*,*w2'*,*s1''*,*s1''* $ *ltr1'*,*s2*,*ltr2*,*statA*,*sv1''*,*sv2''*,*statOO*))

⟨*proof*⟩

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

ω4 Δ *w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO* ∧
h-t (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
((*trv1*,*sv1'*,*trv2*,*sv2'*,*statA*,*statOO*),
(*if trv2* = [] *then R else L*,
w1',*w2'*,*s1*,*ltr1*,*s2'*,*ltr2'*,*statA*,*sv1'*,*sv2'*,*statOO*))
⟨*proof*⟩

lemma *lltrv1-lltrv2-lnever-R*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*

and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l'*: ¬ *isIntO s1 lnever isIntO ltr2*
and *len*: *llength ltr1 > Suc 0 llength ltr2 > Suc 0*
and *l*: *trn = R lnever isSecO ltr2*
shows ∃ *w1' w2' s2' ltr2' trv1 sv1' trv2 sv2' statOO.*
ltr2 = s2 $ ltr2' ∧ validTransO (s2,s2') ∧
Opt.lvalidFromS s2' ltr2' ∧ Opt.lcompletedFrom s2' ltr2' ∧ lnever isIntO ltr2' ∧

ω4 Δ w1 w2 w1' w2' s1 s2 s2' statA sv1 trv1 sv1' sv2 trv2 sv2' statOO ∧
lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
lappend (llist-of trv1) (lltrv1 (if trv2 = [] then R else L,
w1',w2',s1,ltr1,s2',ltr2',statA,sv1',sv2',statOO)) ∧
lltrv2 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
lappend (llist-of trv2) (lltrv2 (if trv2 = [] then R else L,
w1',w2',s1,ltr1,s2',ltr2',statA,sv1',sv2',statOO))
⟨*proof*⟩

lemma *h-t-not-lnever-R*:
assumes *unw*: *unwindCond Δ*
and *Δ*: *Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l'*: ¬ *isIntO s1 lnever isIntO ltr2*
and *len*: *llength ltr1 > Suc 0 llength ltr2 > Suc 0*
and *l*: *trn = R ¬ lnever isSecO ltr2*
shows ∃ *w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO.*
χχ s2 ltr2 tr2 s2' s2'' ltr2' ∧
χ4' Δ w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO
∧
h-t (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
((trv1,sv1'',trv2,sv2'',statA,statOO),
(L,w1',w2',s1,ltr1,s2'',s2'' $ ltr2',statA,sv1'',sv2'',statOO))
⟨*proof*⟩

lemma *lltrv1-lltrv2-not-lnever-R*:
assumes *unw*: *unwindCond Δ*
and *Δ*: *Δ ∞ w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l'*: ¬ *isIntO s1 lnever isIntO ltr2*
and *len*: *llength ltr1 > Suc 0 llength ltr2 > Suc 0*
and *l*: *trn = R ¬ lnever isSecO ltr2*
shows ∃ *w1' w2' tr2 s2' s2'' ltr2' trv1 sv1'' trv2 sv2'' statOO.*
χχ s2 ltr2 tr2 s2' s2'' ltr2' ∧
χ4' Δ w1 w2 w1' w2' s1 s2 tr2 s2' s2'' statA sv1 trv1 sv1'' sv2 trv2 sv2'' statOO
∧
lltrv1 (trn,w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =

lappend (*llist-of trv1*) (*lltrv1* (*L*,*w1'*,*w2'*,*s1*,*ltr1*,*s2''*,*s2''* $ *ltr2'*,*statA*,*sv1''*,*sv2''*,*statOO*))
∧
lltrv2 (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
lappend (*llist-of trv2*) (*lltrv2* (*L*,*w1'*,*w2'*,*s1*,*ltr1*,*s2''*,*s2''* $ *ltr2'*,*statA*,*sv1''*,*sv2''*,*statOO*))
⟨*proof*⟩

lemma *f-not-LNil*: *ltr1* ≠ [[]] ∨ *ltr2* ≠ [[]] ⟹
f (*w1*,*w2*,*trn*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*) ≠ [[]]
⟨*proof*⟩

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

lemma *lvalidFromS-lltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lvalidFromS sv2* (*lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
⟨*proof*⟩

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

lemma *lcompletedFrom-lltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*

shows *Van.lcompletedFrom sv2* (*lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
⟨*proof*⟩

lemma *lS-lltrv1-ltr1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lS* (*lltrv1* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) = *Opt.lS ltr1*
⟨*proof*⟩

lemma *lS-lltrv2-ltr2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1 lnever isIntO ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2 lnever isIntO ltr2*
shows *Van.lS* (*lltrv2* (*trn*,*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) = *Opt.lS ltr2*
⟨*proof*⟩

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

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

fun *H-T* ::
(*'stateO*,*'stateV*)*tuple34* ⇒
(*'stateO*,*'stateV*)*tuple12* ×
(*'stateO*,*'stateV*)*tuple34*
where

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

declare *H-T.simps*[*simp del*]

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

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

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

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

lemma *F-LNil*:
ltr1 = [[]] ∨ *ltr2* = [[]] ⟹ *F* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[]]
⟨*proof*⟩

lemma *F-lnever*:
assumes *ltr1* ≠ [[]] *ltr2* ≠ [[]] *lnever isIntO ltr1*
shows *F* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = *f* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*,
ltr2, *statA*, *sv1*, *sv2*, *statO*)
⟨*proof*⟩

lemma *F-not-lnever*:
assumes *ltr1* ≠ [[]] *ltr2* ≠ [[]] ¬ *lnever isIntO ltr1*
shows *F* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) =
LCons (*H* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) (*F* (*T* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)
))
⟨*proof*⟩

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

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

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

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

lemma *ltrv1-ne-imp*:
assumes *ltrv1 tp* ≠ [[]]
shows ∃ *trv1 sv1'' trv2 sv2'' statAA statOO*. (*trv1*,*sv1''*,*trv2*,*sv2''*,*statAA*,*statOO*) ∈ *lset* (*F tp*) ∧
trv1 ≠ []
⟨*proof*⟩

lemma *ltrv2-ne-imp*:
assumes *ltrv2 tp* ≠ [[]]
shows ∃ *trv1 sv1'' trv2 sv2'' statAA statOO*. (*trv1*,*sv1''*,*trv2*,*sv2''*,*statAA*,*statOO*) ∈ *lset* (*F tp*) ∧
trv2 ≠ []
⟨*proof*⟩

lemma *ltrv1-LNil*[*simp*]:
ltr1 = [[]] ∨ *ltr2* = [[]] ⟹ *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[]]
⟨*proof*⟩
lemma *ltrv2-LNil*[*simp*]:
ltr1 = [[]] ∨ *ltr2* = [[]] ⟹ *ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = [[]]
⟨*proof*⟩

lemma *ltrv1-lnever*:
assumes *ltr1* ≠ [[]] *ltr2* ≠ [[]] *lnever isIntO ltr1*
shows *ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*) = *lltrv1* (*L*, *w1*, *w2*, *s1*, *ltr1*, *s2*, *ltr2*, *statA*, *sv1*, *sv2*, *statO*)
⟨*proof*⟩

lemma *ltrv2-lnever*:
assumes *ltr1* ≠ [[]] *ltr2* ≠ [[]] *lnever isIntO ltr1*

shows *ltrv2 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) = lltrv2 (L, w1, w2, s1, ltr1, s2, ltr2, statA, sv1, sv2, statO)*
⟨*proof*⟩

lemma *H-T-not-lnever*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA = Diff* ⟶ *statO = Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l*: ¬ *lnever isIntO ltr1 Opt.lA ltr1 = Opt.lA ltr2*
shows ∃ *w1′ w2′ tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′ trv1 sv1″ trv2 sv2″ statAA statOO*.
φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′ ∧
φ′ Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1″ s2 tr2 s2′ s2″ statAA statO sv1 trv1 sv1″ sv2 trv2 sv2″ statOO* ∧
H-T (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
((trv1,sv1″,trv2,sv2″,statAA,statOO),
(w1′,w2′, s1″,s1″ $ ltr1′,s2″,s2″ $ ltr2′,statAA,sv1″,sv2″,statOO))
⟨*proof*⟩

lemma *ltrv1-ltrv2-not-lnever*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA = Diff* ⟶ *statO = Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *l*: ¬ *lnever isIntO ltr1 Opt.lA ltr1 = Opt.lA ltr2*
shows ∃ *w1′ w2′ tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′ trv1 sv1″ trv2 sv2″ statAA statOO*.
φφ s1 ltr1 s2 ltr2 tr1 s1′ s1″ ltr1′ tr2 s2′ s2″ ltr2′ ∧
φ′ Δ *w1 w2 w1′ w2′ statA s1 tr1 s1′ s1″ s2 tr2 s2′ s2″ statAA statO sv1 trv1 sv1″ sv2 trv2 sv2″ statOO* ∧
ltrv1 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
lappend (llist-of trv1) (ltrv1 (w1′,w2′,s1″,s1″ $ ltr1′,s2″,s2″ $ ltr2′,statAA,sv1″,sv2″,statOO))
∧
ltrv2 (w1,w2,s1,ltr1,s2,ltr2,statA,sv1,sv2,statO) =
lappend (llist-of trv2) (ltrv2 (w1′,w2′,s1″,s1″ $ ltr1′,s2″,s2″ $ ltr2′,statAA,sv1″,sv2″,statOO))
⟨*proof*⟩

lemma *lvalidFromS-ltrv1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*

and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA = Diff* $\longrightarrow$ *statO = Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *ltr14*: *Opt.lA ltr1 = Opt.lA ltr2*
shows *Van.lvalidFromS sv1* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
⟨*proof*⟩

lemma *lvalidFromS-ltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA = Diff* $\longrightarrow$ *statO = Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *ltr14*: *Opt.lA ltr1 = Opt.lA ltr2*
shows *Van.lvalidFromS sv2* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
⟨*proof*⟩

lemma *lcompletedFrom-ltrv1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA = Diff* $\longrightarrow$ *statO = Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1 = Opt.lA ltr2*
shows *Van.lcompletedFrom sv1* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
⟨*proof*⟩

lemma *lcompletedFrom-ltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA = Diff* $\longrightarrow$ *statO = Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1 = Opt.lA ltr2*
shows *Van.lcompletedFrom sv2* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
⟨*proof*⟩

lemma *lS-ltrv1-ltr1*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*

and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lS* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) = *Opt.lS ltr1*
⟨*proof*⟩

lemma *lS-ltrv2-ltr2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lS* (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) = *Opt.lS ltr2*
⟨*proof*⟩

lemma *lA-ltrv1-ltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
shows *Van.lA* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) =
Van.lA (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
⟨*proof*⟩

lemma *lO-ltrv1-ltrv2*:
assumes *unw*: *unwindCond* Δ
and Δ: Δ ∞ *w1 w2 s1 s2 statA sv1 sv2 statO*
and *r*: *reachO s1 reachO s2 reachV sv1 reachV sv2*
and *stat*: *statA* = *Diff* ⟶ *statO* = *Diff*
and *ltr1*: *Opt.lvalidFromS s1 ltr1 Opt.lcompletedFrom s1 ltr1*
and *ltr2*: *Opt.lvalidFromS s2 ltr2 Opt.lcompletedFrom s2 ltr2*
and *A34*: *Opt.lA ltr1* = *Opt.lA ltr2*
and *O12*: *Van.lO* (*ltrv1* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*)) =
Van.lO (*ltrv2* (*w1*,*w2*,*s1*,*ltr1*,*s2*,*ltr2*,*statA*,*sv1*,*sv2*,*statO*))
and *stO*: *statO* = *Eq*
shows *Opt.lO ltr1* = *Opt.lO ltr2*
⟨*proof*⟩

end

theorem *unwind-lrsecure*:
assumes *init*: *initCond* Δ **and** *unwind*: *unwindCond* Δ
shows *lrsecure*
⟨*proof*⟩

4.4 Compositional unwinding

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

definition *unwindIntoCond* ::
(*enat* ⇒ *enat* ⇒ *enat* ⇒ *'stateO* ⇒ *'stateO* ⇒ *status* ⇒ *'stateV* ⇒ *'stateV* ⇒ *status* ⇒ *bool*) ⇒
(*enat* ⇒ *enat* ⇒ *enat* ⇒ *'stateO* ⇒ *'stateO* ⇒ *status* ⇒ *'stateV* ⇒ *'stateV* ⇒ *status* ⇒ *bool*)
⇒ *bool*
where
unwindIntoCond Δ Δ' ≡ ∀ *w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 ∧ *reachO s2* ∧ *reachV sv1* ∧ *reachV sv2* ∧
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* ⟶
(*finalO s1* ⟷ *finalO s2*) ∧ (*finalV sv1* ⟷ *finalO s1*) ∧ (*finalV sv2* ⟷ *finalO s2*)
∧
(*statA* = *Eq* ⟶ (*isIntO s1* ⟷ *isIntO s2*))
∧
((∃ *v*<*w*. *proact* Δ' *v w1 w2 s1 s2 statA sv1 sv2 statO*)
∨
match Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*)

theorem *distrib-unwind-lrsecure*:
assumes *m*: *0* < *m* **and** *nxt*: ⋀*i*. *i* < (*m*::*nat*) ⟹ *nxt i* ⊆ {*0*..<*m*}
and *init*: *initCond* (*Δs 0*)
and *step*: ⋀*i*. *i* < *m* ⟹
unwindIntoCond (*Δs i*) (λ*w w1 w2 s1 s2 statA sv1 sv2 statO*.
∃ *j* ∈ *nxt i*. *Δs j w w1 w2 s1 s2 statA sv1 sv2 statO*)
shows *lrsecure*
⟨*proof*⟩

lemma *unwindIntoCond-simpleI*:
assumes
⋀*w w1 w2 s1 s2 statA sv1 sv2 statO*.
reachO s1 ⟹ *reachO s2* ⟹ *reachV sv1* ⟹ *reachV sv2* ⟹
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟹
(*finalO s1* ⟷ *finalO s2*) ∧ (*finalV sv1* ⟷ *finalO s1*) ∧ (*finalV sv2* ⟷ *finalO*

s2)
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
statA = *Eq*
$\Longrightarrow$
isIntO s1 $\longleftrightarrow$ *isIntO s2*
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
match Δ' *w1 w2 s1 s2 statA sv1 sv2 statO*
shows *unwindIntoCond* Δ Δ'
⟨*proof*⟩

lemma *unwindIntoCond-simpleI2*:
assumes
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
(*finalO s1* $\longleftrightarrow$ *finalO s2*) $\wedge$ (*finalV sv1* $\longleftrightarrow$ *finalO s1*) $\wedge$ (*finalV sv2* $\longleftrightarrow$ *finalO s2*)
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
statA = *Eq*
$\Longrightarrow$
isIntO s1 $\longleftrightarrow$ *isIntO s2*
and
$\bigwedge$*w w1 w2 s1 s2 statA sv1 sv2 statO.*
reachO s1 $\Longrightarrow$ *reachO s2* $\Longrightarrow$ *reachV sv1* $\Longrightarrow$ *reachV sv2* $\Longrightarrow$
Δ *w w1 w2 s1 s2 statA sv1 sv2 statO*
$\Longrightarrow$
($\exists$ *v*<*w. proact* Δ' *v w1 w2 s1 s2 statA sv1 sv2 statO*)
shows *unwindIntoCond* Δ Δ'
⟨*proof*⟩

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

$\bigwedge w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$
$statA = Eq$
$\Longrightarrow$
$isIntO\ s1 \longleftrightarrow isIntO\ s2$
and
$\bigwedge w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO.$
$reachO\ s1 \Longrightarrow reachO\ s2 \Longrightarrow reachV\ sv1 \Longrightarrow reachV\ sv2 \Longrightarrow$
$\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\Longrightarrow$
$(\exists\, v{<}w.\ proact\ \Delta'\ v\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO) \vee match\ \Delta'\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
shows $unwindIntoCond\ \Delta\ \Delta'$
$\langle proof \rangle$

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*:
$\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow oor\ \Delta\ \Delta_2\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\langle proof \rangle$

lemma *oorI2*:
$\Delta_2\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow oor\ \Delta\ \Delta_2\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\langle proof \rangle$

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*:
$\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow oor3\ \Delta\ \Delta_2\ \Delta_3\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\langle proof \rangle$

lemma *oor3I2*:
$\Delta_2\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow oor3\ \Delta\ \Delta_2\ \Delta_3\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO$
$\langle proof \rangle$

lemma *oor3I3*:
$\Delta_3\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow oor3\ \Delta\ \Delta_2\ \Delta_3\ w\ w1\ w2\ s1\ s2\ statA$

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*:
$\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$ *oor4* $\Delta\ \Delta_2\ \Delta_3\ \Delta_4\ w\ w1\ w2\ s1\ s2\ statA$ *sv1 sv2 statO*
⟨*proof*⟩

lemma *oor4I2*:
$\Delta_2\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$ *oor4* $\Delta\ \Delta_2\ \Delta_3\ \Delta_4\ w\ w1\ w2\ s1\ s2\ statA$ *sv1 sv2 statO*
⟨*proof*⟩

lemma *oor4I3*:
$\Delta_3\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$ *oor4* $\Delta\ \Delta_2\ \Delta_3\ \Delta_4\ w\ w1\ w2\ s1\ s2\ statA$ *sv1 sv2 statO*
⟨*proof*⟩

lemma *oor4I4*:
$\Delta_4\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$ *oor4* $\Delta\ \Delta_2\ \Delta_3\ \Delta_4\ w\ w1\ w2\ s1\ s2\ statA$ *sv1 sv2 statO*
⟨*proof*⟩

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*:
$\Delta\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$ *oor5* $\Delta\ \Delta_2\ \Delta_3\ \Delta_4\ \Delta_5\ w\ w1\ w2\ s1\ s2$ *statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor5I2*:
$\Delta_2\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$ *oor5* $\Delta\ \Delta_2\ \Delta_3\ \Delta_4\ \Delta_5\ w\ w1\ w2\ s1\ s2$ *statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor5I3*:
$\Delta_3\ w\ w1\ w2\ s1\ s2\ statA\ sv1\ sv2\ statO \Longrightarrow$ *oor5* $\Delta\ \Delta_2\ \Delta_3\ \Delta_4\ \Delta_5\ w\ w1\ w2\ s1\ s2$ *statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor5I4*:
Δ_4 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *oor5I5*:
Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$ *oor5* Δ Δ_2 Δ_3 Δ_4 Δ_5 *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *isIntO-match1*: *isIntO s1* $\Longrightarrow$ *match1* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *isIntO-match2*: *isIntO s2* $\Longrightarrow$ *match2* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *isIntO-match*:
assumes ‹*isIntO s1*› **and** ‹*isIntO s2*›
and ‹*match12* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*›
shows ‹*match* Δ *w1 w2 s1 s2 statA sv1 sv2 statO*›
⟨*proof*⟩

lemma *match1-1-oorI1*:
match1-1 Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$
match1-1 (*oor* Δ Δ_2) *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match1-1-oorI2*:
match1-1 Δ_2 *w1 w2 s1 s1' s2 statA sv1 sv2 statO* $\Longrightarrow$
match1-1 (*oor* Δ Δ_2) *w1 w2 s1 s1' s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match1-oorI1*:
match1 Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match1 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match1-oorI2*:
match1 Δ_2 *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match1 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-1-oorI1*:
match2-1 Δ *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\Longrightarrow$
match2-1 (*oor* Δ Δ_2) *w1 w2 s1 s2 s2' statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-1-oorI2*:
match2-1 Δ_2 *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\Longrightarrow$
match2-1 (*oor* Δ Δ_2) *w1 w2 s1 s2 s2' statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-oorI1*:
match2 Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match2 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match2-oorI2*:
match2 Δ_2 *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match2 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-oorI1*:
match12 Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match12 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-oorI2*:
match12 Δ_2 *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match12 (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-1-oorI1*:
match12-1 Δ *w1 w2 s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-1 (*oor* Δ Δ_2) *w1 w2 s1' s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-1-oorI2*:
match12-1 Δ_2 *w1 w2 s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-1 (*oor* Δ Δ_2) *w1 w2 s1' s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-2-oorI1*:
match12-2 Δ *w1 w2 s2 s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-2 (*oor* Δ Δ_2) *w1 w2 s2 s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-2-oorI2*:
match12-2 Δ_2 *w1 w2 s2 s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-2 (*oor* Δ Δ_2) *w1 w2 s2 s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-12-oorI1*:
match12-12 Δ *w1 w2 s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-12 (*oor* Δ Δ_2) *w1 w2 s1' s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match12-12-oorI2*:
match12-12 Δ_2 *w1 w2 s1' s2' statA' sv1 sv2 statO* $\Longrightarrow$
match12-12 (*oor* Δ Δ_2) *w1 w2 s1' s2' statA' sv1 sv2 statO*
⟨*proof*⟩

lemma *match-oorI1*:
match Δ *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *match-oorI2*:
match Δ_2 *w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
match (*oor* Δ Δ_2) *w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *proact-oorI1*:
proact Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
proact (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *proact-oorI2*:
proact Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
proact (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-1-oorI1*:
move-1 Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-1 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-1-oorI2*:
move-1 Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-1 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-2-oorI1*:
move-2 Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-2 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-2-oorI2*:
move-2 Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-2 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-12-oorI1*:
move-12 Δ *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-12 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

lemma *move-12-oorI2*:
move-12 Δ_2 *w w1 w2 s1 s2 statA sv1 sv2 statO* $\Longrightarrow$
move-12 (*oor* Δ Δ_2) *w w1 w2 s1 s2 statA sv1 sv2 statO*
⟨*proof*⟩

end

context *Relative-Security-Determ*
begin

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

$\neg$ *finalV sv1* $\wedge$ Δ ∞ *w1 w2 s1' s2 statA* (*nextO sv1*) *sv2 statO*
⟨*proof*⟩

lemma *match1-12-defD*: *match1-12* Δ *w1 w2 s1 s1' s2 statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$
Δ ∞ *w1 w2 s1' s2 statA* (*nextO sv1*) (*nextO sv2*) (*sstatO' statO sv1 sv2*)
⟨*proof*⟩

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

lemma *match2-1-defD*: *match2-1* Δ *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv2* $\wedge$ Δ ∞ *w1 w2 s1 s2' statA sv1* (*nextO sv2*) *statO*
⟨*proof*⟩

lemma *match2-12-defD*: *match2-12* Δ *w1 w2 s1 s2 s2' statA sv1 sv2 statO* $\longleftrightarrow$
$\neg$ *finalV sv1* $\wedge$ $\neg$ *finalV sv2* $\wedge$ Δ ∞ *w1 w2 s1 s2' statA* (*nextO sv1*) (*nextO sv2*)
(*sstatO' statO sv1 sv2*)
⟨*proof*⟩

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

lemma *match12-1-defD*: *match12-1 Δ w1 w2 s1′ s2′ statA′ sv1 sv2 statO ⟷*
¬ finalV sv1 ∧ Δ ∞ w1 w2 s1′ s2′ statA′ (nextO sv1) sv2 statO
⟨*proof*⟩

lemma *match12-2-defD*: *match12-2 Δ w1 w2 s1′ s2′ statA′ sv1 sv2 statO ⟷*
¬ finalV sv2 ∧ Δ ∞ w1 w2 s1′ s2′ statA′ sv1 (nextO sv2) statO
⟨*proof*⟩

lemma *match12-12-defD*: *match12-12 Δ w1 w2 s1′ s2′ statA′ sv1 sv2 statO ⟷*

(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

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.