

An Isabelle/HOL Formalization of the Modular Assembly Kit for Security Properties

Oliver Bračevac, Richard Gay, Sylvia Grewe,
Heiko Mantel, Henning Sudbrock, Markus Tasch

Abstract

The “Modular Assembly Kit for Security Properties” (MAKS) is a framework for both the definition and verification of possibilistic information-flow security properties at the specification-level. MAKS supports the uniform representation of a wide range of possibilistic information-flow properties and provides support for the verification of such properties via unwinding results and compositionality results. We provide a formalization of this framework in Isabelle/HOL.

Contents

1	Introduction	2
2	Basic Definitions	2
3	System Specification	5
3.1	Event Systems	5
3.2	State-Event Systems	7
4	Security Specification	10
4.1	Views & Flow Policies	10
4.2	Basic Security Predicates	12
4.3	Information-Flow Properties	16
4.4	Property Library	17
5	Verification	20
5.1	Basic Definitions	20
5.2	Taxonomy Results	21
5.3	Unwinding	29
5.3.1	Unwinding Conditions	29
5.3.2	Auxiliary Results	30
5.3.3	Unwinding Theorems	32
5.4	Compositionality	33
5.4.1	Auxiliary Definitions & Results	33
5.4.2	Generalized Zipping Lemma	36
5.4.3	Compositionality Results	37

1 Introduction

This is a formalization of the Modular Assembly Kit for Security Properties (MAKS) [2, 3] in its version from [3]. We provide a more detailed explanation on how key concepts of MAKS are formalized in Isabelle/HOL in [1].

2 Basic Definitions

In the following, we define the notion of prefixes and the notion of projection. These definitions are preliminaries for the remaining parts of the Isabelle/HOL formalization of MAKS.

```
theory Prefix
imports Main
begin
```

```
definition prefix :: 'e list  $\Rightarrow$  'e list  $\Rightarrow$  bool (infixl  $\preceq$  100)
where
(l1  $\preceq$  l2)  $\equiv$  ( $\exists$  l3. l1 @ l3 = l2)
```

```
definition prefixclosed :: ('e list) set  $\Rightarrow$  bool
where
prefixclosed tr  $\equiv$  ( $\forall$  l1  $\in$  tr.  $\forall$  l2. l2  $\preceq$  l1  $\longrightarrow$  l2  $\in$  tr)
```

```
lemma empty-prefix-of-all: []  $\preceq$  l
  <proof>
```

```
lemma empty-trace-contained: [] prefixclosed tr ; tr  $\neq$  {}  $\Longrightarrow$  []  $\in$  tr
  <proof>
```

```
lemma transitive-prefix: [] l1  $\preceq$  l2 ; l2  $\preceq$  l3  $\Longrightarrow$  l1  $\preceq$  l3
  <proof>
```

```
end
theory Projection
imports Main
begin
```

```
definition projection:: 'e list  $\Rightarrow$  'e set  $\Rightarrow$  'e list (infixl  $\upharpoonright$  100)
where
l  $\upharpoonright$  E  $\equiv$  filter ( $\lambda$ x . x  $\in$  E) l
```

```
lemma projection-on-union:
  l  $\upharpoonright$  Y = []  $\Longrightarrow$  l  $\upharpoonright$  (X  $\cup$  Y) = l  $\upharpoonright$  X
  <proof>
```

lemma *projection-on-empty-trace*: $\llbracket \] \ X = \llbracket \] \langle \text{proof} \rangle$

lemma *projection-to-emptyset-is-empty-trace*: $l \upharpoonright \{\} = \llbracket \] \langle \text{proof} \rangle$

lemma *projection-idempotent*: $l \upharpoonright X = (l \upharpoonright X) \upharpoonright X \langle \text{proof} \rangle$

lemma *projection-empty-implies-absence-of-events*: $l \upharpoonright X = \llbracket \] \implies X \cap (\text{set } l) = \{\} \langle \text{proof} \rangle$

lemma *disjoint-projection*: $X \cap Y = \{\} \implies (l \upharpoonright X) \upharpoonright Y = \llbracket \] \langle \text{proof} \rangle$

lemma *projection-concatenation-commute*:
 $(l1 \ @ \ l2) \upharpoonright X = (l1 \upharpoonright X) \ @ \ (l2 \upharpoonright X) \langle \text{proof} \rangle$

lemma *projection-subset-eq-from-superset-eq*:
 $((xs \upharpoonright (X \cup Y)) = (ys \upharpoonright (X \cup Y))) \implies ((xs \upharpoonright X) = (ys \upharpoonright X)) \langle \text{proof} \rangle$
 $(\text{is } ?L1 = ?L2) \implies (?L3 = ?L4)$

lemma *list-subset-iff-projection-neutral*: $(\text{set } l \subseteq X) = ((l \upharpoonright X) = l) \langle \text{proof} \rangle$
 $(\text{is } ?A = ?B)$

lemma *projection-split-last*: $\text{Suc } n = \text{length } (\tau \upharpoonright X) \implies \exists \beta \ x \ \alpha. (x \in X \wedge \tau = \beta \ @ \ [x] \ @ \ \alpha \wedge \alpha \upharpoonright X = \llbracket \] \wedge n = \text{length } ((\beta \ @ \ \alpha) \upharpoonright X)) \langle \text{proof} \rangle$

lemma *projection-rev-commute*:
 $\text{rev } (l \upharpoonright X) = (\text{rev } l) \upharpoonright X \langle \text{proof} \rangle$

lemma *projection-split-first*: $\llbracket \] \ (\tau \upharpoonright X) = x \ # \ xs \implies \exists \alpha \ \beta. (\tau = \alpha \ @ \ [x] \ @ \ \beta \wedge \alpha \upharpoonright X = \llbracket \])$
 $\langle \text{proof} \rangle$

lemma *projection-split-first-with-suffix*:
 $\llbracket \] \ (\tau \upharpoonright X) = x \ # \ xs \implies \exists \alpha \ \beta. (\tau = \alpha \ @ \ [x] \ @ \ \beta \wedge \alpha \upharpoonright X = \llbracket \] \wedge \beta \upharpoonright X = xs) \langle \text{proof} \rangle$

lemma *projection-split-arbitrary-element*:

$\llbracket \tau \upharpoonright X = (\alpha @ [x] @ \beta) \upharpoonright X; x \in X \rrbracket$
 $\implies \exists \alpha' \beta'. (\tau = \alpha' @ [x] @ \beta' \wedge \alpha' \upharpoonright X = \alpha \upharpoonright X \wedge \beta' \upharpoonright X = \beta \upharpoonright X)$
(proof)

lemma *projection-on-intersection*: $l \upharpoonright X = [] \implies l \upharpoonright (X \cap Y) = []$

(is ?L1 = [] \implies ?L2 = [])
(proof)

lemma *projection-on-subset*: $\llbracket Y \subseteq X; l \upharpoonright X = [] \rrbracket \implies l \upharpoonright Y = []$

(proof)

lemma *projection-on-subset2*: $\llbracket \text{set } l \subseteq L; l \upharpoonright X' = []; X \cap L \subseteq X' \rrbracket \implies l \upharpoonright X = []$

(proof)

lemma *non-empty-projection-on-subset*: $X \subseteq Y \wedge l_1 \upharpoonright Y = l_2 \upharpoonright Y \implies l_1 \upharpoonright X = l_2 \upharpoonright X$

(proof)

lemma *projection-intersection-neutral*: $(\text{set } l \subseteq X) \implies (l \upharpoonright (X \cap Y) = l \upharpoonright Y)$

(proof)

lemma *projection-commute*:

$(l \upharpoonright X) \upharpoonright Y = (l \upharpoonright Y) \upharpoonright X$

(proof)

lemma *projection-subset-elim*: $Y \subseteq X \implies (l \upharpoonright X) \upharpoonright Y = l \upharpoonright Y$

(proof)

lemma *projection-sequence*: $(xs \upharpoonright X) \upharpoonright Y = (xs \upharpoonright (X \cap Y))$

(proof)

fun *merge* :: 'e set \Rightarrow 'e set \Rightarrow 'e list \Rightarrow 'e list \Rightarrow 'e list

where

merge A B [] t2 = t2 |

merge A B t1 [] = t1 |

merge A B (e1 # t1') (e2 # t2') = (if e1 = e2 then
e1 # (*merge* A B t1' t2')
else (if e1 \in (A \cap B) then
e2 # (*merge* A B (e1 # t1') t2')
else e1 # (*merge* A B t1' (e2 # t2')))))

lemma *merge-property*: $\llbracket \text{set } t1 \subseteq A; \text{set } t2 \subseteq B; t1 \upharpoonright B = t2 \upharpoonright A \rrbracket$
 $\implies \text{let } t = (\text{merge } A \ B \ t1 \ t2) \text{ in } (t \upharpoonright A = t1 \wedge t \upharpoonright B = t2 \wedge \text{set } t \subseteq ((\text{set } t1) \cup (\text{set } t2)))$
 $\langle \text{proof} \rangle$

end

3 System Specification

3.1 Event Systems

We define the system model of event systems as well as the parallel composition operator for event systems provided as part of MAKS in [3].

theory *EventSystems*
imports *../Basics/Prefix ../Basics/Projection*
begin

record *'e ES-rec* =
E-ES :: *'e set*
I-ES :: *'e set*
O-ES :: *'e set*
Tr-ES :: (*'e list*) *set*

abbreviation *ESrecEES* :: *'e ES-rec* \Rightarrow *'e set*
(*E*- [1000] 1000)
where
E_{ES} \equiv (*E-ES ES*)

abbreviation *ESrecIES* :: *'e ES-rec* \Rightarrow *'e set*
(*I*- [1000] 1000)
where
I_{ES} \equiv (*I-ES ES*)

abbreviation *ESrecOES* :: *'e ES-rec* \Rightarrow *'e set*
(*O*- [1000] 1000)
where
O_{ES} \equiv (*O-ES ES*)

abbreviation *ESrecTrES* :: *'e ES-rec* \Rightarrow (*'e list*) *set*
(*Tr*- [1000] 1000)
where
Tr_{ES} \equiv (*Tr-ES ES*)

definition *es-inputs-are-events* :: *'e ES-rec* \Rightarrow *bool*
where
es-inputs-are-events ES \equiv *I_{ES} \subseteq E_{ES}*

definition *es-outputs-are-events* :: 'e ES-rec \Rightarrow bool

where

es-outputs-are-events ES $\equiv O_{ES} \subseteq E_{ES}$

definition *es-inputs-outputs-disjoint* :: 'e ES-rec \Rightarrow bool

where

es-inputs-outputs-disjoint ES $\equiv I_{ES} \cap O_{ES} = \{\}$

definition *traces-contain-events* :: 'e ES-rec \Rightarrow bool

where

traces-contain-events ES $\equiv \forall l \in Tr_{ES}. \forall e \in (set\ l). e \in E_{ES}$

definition *traces-prefixclosed* :: 'e ES-rec \Rightarrow bool

where

traces-prefixclosed ES $\equiv prefixclosed\ Tr_{ES}$

definition *ES-valid* :: 'e ES-rec \Rightarrow bool

where

ES-valid ES \equiv

es-inputs-are-events ES \wedge *es-outputs-are-events* ES
 \wedge *es-inputs-outputs-disjoint* ES \wedge *traces-contain-events* ES
 \wedge *traces-prefixclosed* ES

definition *total* :: 'e ES-rec \Rightarrow 'e set \Rightarrow bool

where

total ES E $\equiv E \subseteq E_{ES} \wedge (\forall \tau \in Tr_{ES}. \forall e \in E. \tau @ [e] \in Tr_{ES})$

lemma *totality*: $\llbracket total\ ES\ E; t \in Tr_{ES}; set\ t' \subseteq E \rrbracket \Longrightarrow t @ t' \in Tr_{ES}$
(*proof*)

definition *composeES* :: 'e ES-rec \Rightarrow 'e ES-rec \Rightarrow 'e ES-rec

where

composeES ES1 ES2 \equiv

(
 E-ES = $E_{ES1} \cup E_{ES2}$,
 I-ES = $(I_{ES1} - O_{ES2}) \cup (I_{ES2} - O_{ES1})$,
 O-ES = $(O_{ES1} - I_{ES2}) \cup (O_{ES2} - I_{ES1})$,
 Tr-ES = $\{\tau \cdot (\tau \upharpoonright E_{ES1}) \in Tr_{ES1} \wedge (\tau \upharpoonright E_{ES2}) \in Tr_{ES2}$
 $\wedge (set\ \tau \subseteq E_{ES1} \cup E_{ES2})\}$
)

abbreviation *composeESAbbrv* :: 'e ES-rec \Rightarrow 'e ES-rec \Rightarrow 'e ES-rec

(- || -[1000] 1000)

where

ES1 || ES2 $\equiv (composeES\ ES1\ ES2)$

definition *composable* :: 'e ES-rec \Rightarrow 'e ES-rec \Rightarrow bool
where
composable ES1 ES2 $\equiv (E_{ES1} \cap E_{ES2}) \subseteq ((O_{ES1} \cap I_{ES2}) \cup (O_{ES2} \cap I_{ES1}))$

lemma *composeES-yields-ES*:
 $\llbracket ES\text{-valid } ES1; ES\text{-valid } ES2 \rrbracket \Longrightarrow ES\text{-valid } (ES1 \parallel ES2)$
 ⟨proof⟩

end

3.2 State-Event Systems

We define the system model of state-event systems as well as the translation from state-event systems to event systems provided as part of MAKS in [3]. State-event systems are the basis for the unwinding theorems that we prove later in this entry.

theory *StateEventSystems*
imports *EventSystems*
begin

record ('s, 'e) *SES-rec* =
S-SES :: 's set
s0-SES :: 's
E-SES :: 'e set
I-SES :: 'e set
O-SES :: 'e set
T-SES :: 's \Rightarrow 'e \rightarrow 's

abbreviation *SESrecSSES* :: ('s, 'e) *SES-rec* \Rightarrow 's set
 (*S*- [1000] 1000)
where
SSES $\equiv (S\text{-SES } SES)$

abbreviation *SESrecs0SES* :: ('s, 'e) *SES-rec* \Rightarrow 's
 (*s0*- [1000] 1000)
where
s0SES $\equiv (s0\text{-SES } SES)$

abbreviation *SESrecESES* :: ('s, 'e) *SES-rec* \Rightarrow 'e set
 (*E*- [1000] 1000)
where
ESES $\equiv (E\text{-SES } SES)$

abbreviation *SESrecISES* :: ('s, 'e) *SES-rec* \Rightarrow 'e set
 (*I*- [1000] 1000)
where
ISES $\equiv (I\text{-SES } SES)$

abbreviation $SESrecOSES :: ('s, 'e) SES-rec \Rightarrow 'e \text{ set}$
 $(O- [1000] 1000)$

where
 $O_{SES} \equiv (O-SES SES)$

abbreviation $SESrecTSES :: ('s, 'e) SES-rec \Rightarrow ('s \Rightarrow 'e \rightarrow 's)$
 $(T- [1000] 1000)$

where
 $T_{SES} \equiv (T-SES SES)$

abbreviation $TSESpred :: 's \Rightarrow 'e \Rightarrow ('s, 'e) SES-rec \Rightarrow 's \Rightarrow \text{bool}$
 $(- \longrightarrow - [100,100,100,100] 100)$

where
 $s \ e \longrightarrow_{SES} s' \equiv (T_{SES} s e = \text{Some } s')$

definition $s0\text{-is-state} :: ('s, 'e) SES-rec \Rightarrow \text{bool}$

where
 $s0\text{-is-state } SES \equiv s0_{SES} \in S_{SES}$

definition $\text{ses-inputs-are-events} :: ('s, 'e) SES-rec \Rightarrow \text{bool}$

where
 $\text{ses-inputs-are-events } SES \equiv I_{SES} \subseteq E_{SES}$

definition $\text{ses-outputs-are-events} :: ('s, 'e) SES-rec \Rightarrow \text{bool}$

where
 $\text{ses-outputs-are-events } SES \equiv O_{SES} \subseteq E_{SES}$

definition $\text{ses-inputs-outputs-disjoint} :: ('s, 'e) SES-rec \Rightarrow \text{bool}$

where
 $\text{ses-inputs-outputs-disjoint } SES \equiv I_{SES} \cap O_{SES} = \{\}$

definition $\text{correct-transition-relation} :: ('s, 'e) SES-rec \Rightarrow \text{bool}$

where
 $\text{correct-transition-relation } SES \equiv$
 $\forall x y z. x \ y \longrightarrow_{SES} z \longrightarrow ((x \in S_{SES}) \wedge (y \in E_{SES}) \wedge (z \in S_{SES}))$

definition $SES\text{-valid} :: ('s, 'e) SES-rec \Rightarrow \text{bool}$

where
 $SES\text{-valid } SES \equiv$
 $s0\text{-is-state } SES \wedge \text{ses-inputs-are-events } SES$
 $\wedge \text{ses-outputs-are-events } SES \wedge \text{ses-inputs-outputs-disjoint } SES \wedge$
 $\text{correct-transition-relation } SES$

primrec $\text{path} :: ('s, 'e) SES-rec \Rightarrow 's \Rightarrow 'e \text{ list} \rightarrow 's$

where
 $\text{path-empt: path } SES \ s1 \ [] = (\text{Some } s1) \mid$

path-nonempt: $\text{path SES } s1 \ (e \# t) =$
 (if $(\exists s2. s1 \ e \longrightarrow_{SES} s2)$
 then $(\text{path SES } (\text{the } (T_{SES} s1 \ e))) \ t)$
 else *None*)

abbreviation $\text{pathpred} :: 's \Rightarrow 'e \ \text{list} \Rightarrow ('s, 'e) \ \text{SES-rec} \Rightarrow 's \Rightarrow \text{bool}$
 $(- \Longrightarrow - \ [100, 100, 100, 100] \ 100)$

where
 $s \ t \Longrightarrow_{SES} s' \equiv \text{path SES } s \ t = \text{Some } s'$

definition $\text{reachable} :: ('s, 'e) \ \text{SES-rec} \Rightarrow 's \Rightarrow \text{bool}$

where
 $\text{reachable SES } s \equiv (\exists t. s0_{SES} \ t \Longrightarrow_{SES} s)$

definition $\text{enabled} :: ('s, 'e) \ \text{SES-rec} \Rightarrow 's \Rightarrow 'e \ \text{list} \Rightarrow \text{bool}$

where
 $\text{enabled SES } s \ t \equiv (\exists s'. s \ t \Longrightarrow_{SES} s')$

definition $\text{possible-traces} :: ('s, 'e) \ \text{SES-rec} \Rightarrow ('e \ \text{list}) \ \text{set}$

where
 $\text{possible-traces SES} \equiv \{t. (\text{enabled SES } s0_{SES} \ t)\}$

definition $\text{induceES} :: ('s, 'e) \ \text{SES-rec} \Rightarrow 'e \ \text{ES-rec}$

where
 $\text{induceES SES} \equiv$
 $($
 $\ E\text{-ES} = E_{SES},$
 $\ I\text{-ES} = I_{SES},$
 $\ O\text{-ES} = O_{SES},$
 $\ \text{Tr-ES} = \text{possible-traces SES}$
 $)$

lemma $\text{none-remains-none} : \bigwedge s \ e. (\text{path SES } s \ t) = \text{None}$

$\implies (\text{path SES } s \ (t \ @ \ [e])) = \text{None}$
 $\langle \text{proof} \rangle$

lemma $\text{path-trans-single-neg} : \bigwedge s1. \llbracket s1 \ t \Longrightarrow_{SES} s2; \neg (s2 \ e \longrightarrow_{SES} sn) \rrbracket$

$\implies \neg (s1 \ (t \ @ \ [e]) \Longrightarrow_{SES} sn)$
 $\langle \text{proof} \rangle$

lemma $\text{path-split-single} : s1 \ (t \ @ \ [e]) \Longrightarrow_{SES} sn$

$\implies \exists s'. s1 \ t \Longrightarrow_{SES} s' \ \wedge \ s' \ e \longrightarrow_{SES} sn$
 $\langle \text{proof} \rangle$

lemma *path-trans-single*: $\bigwedge s. \llbracket s \Longrightarrow_{SES} s'; s' e \longrightarrow_{SES} sn \rrbracket$
 $\implies s (t @ [e]) \Longrightarrow_{SES} sn$
 ⟨proof⟩

lemma *path-split*: $\bigwedge sn. \llbracket s1 (t1 @ t2) \Longrightarrow_{SES} sn \rrbracket$
 $\implies (\exists s2. (s1 t1 \Longrightarrow_{SES} s2 \wedge s2 t2 \Longrightarrow_{SES} sn))$
 ⟨proof⟩

lemma *path-trans*:
 $\bigwedge sn. \llbracket s1 l1 \Longrightarrow_{SES} s2; s2 l2 \Longrightarrow_{SES} sn \rrbracket \implies s1 (l1 @ l2) \Longrightarrow_{SES} sn$
 ⟨proof⟩

lemma *enabledPrefixSingle* : $\llbracket \text{enabled } SES \ s \ (t@[e]) \rrbracket \implies \text{enabled } SES \ s \ t$
 ⟨proof⟩

lemma *enabledPrefix* : $\llbracket \text{enabled } SES \ s \ (t1 @ t2) \rrbracket \implies \text{enabled } SES \ s \ t1$
 ⟨proof⟩

lemma *enabledPrefixSingleFinalStep* : $\llbracket \text{enabled } SES \ s \ (t@[e]) \rrbracket \implies \exists t' t''. t' e \longrightarrow_{SES} t''$
 ⟨proof⟩

lemma *induceES-yields-ES*:
 $SES\text{-valid } SES \implies ES\text{-valid } (\text{induceES } SES)$
 ⟨proof⟩

end

4 Security Specification

4.1 Views & Flow Policies

We define views, flow policies and how views can be derived from a given flow policy.

theory *Views*
imports *Main*
begin

record *'e V-rec* =

$V :: 'e \text{ set}$
 $N :: 'e \text{ set}$
 $C :: 'e \text{ set}$

abbreviation $VrecV :: 'e \text{ V-rec} \Rightarrow 'e \text{ set}$
 $(V \text{ [100] 1000})$

where
 $V_v \equiv (V \ v)$

abbreviation $VrecN :: 'e \text{ V-rec} \Rightarrow 'e \text{ set}$
 $(N \text{ [100] 1000})$

where
 $N_v \equiv (N \ v)$

abbreviation $VrecC :: 'e \text{ V-rec} \Rightarrow 'e \text{ set}$
 $(C \text{ [100] 1000})$

where
 $C_v \equiv (C \ v)$

definition $VN\text{-disjoint} :: 'e \text{ V-rec} \Rightarrow \text{bool}$

where
 $VN\text{-disjoint } v \equiv V_v \cap N_v = \{\}$

definition $VC\text{-disjoint} :: 'e \text{ V-rec} \Rightarrow \text{bool}$

where
 $VC\text{-disjoint } v \equiv V_v \cap C_v = \{\}$

definition $NC\text{-disjoint} :: 'e \text{ V-rec} \Rightarrow \text{bool}$

where
 $NC\text{-disjoint } v \equiv N_v \cap C_v = \{\}$

definition $V\text{-valid} :: 'e \text{ V-rec} \Rightarrow \text{bool}$

where
 $V\text{-valid } v \equiv VN\text{-disjoint } v \wedge VC\text{-disjoint } v \wedge NC\text{-disjoint } v$

definition $isViewOn :: 'e \text{ V-rec} \Rightarrow 'e \text{ set} \Rightarrow \text{bool}$

where
 $isViewOn \mathcal{V} E \equiv V\text{-valid } \mathcal{V} \wedge V_{\mathcal{V}} \cup N_{\mathcal{V}} \cup C_{\mathcal{V}} = E$

end

theory $FlowPolicies$

imports $Views$

begin

record $'domain \text{ FlowPolicy-rec} =$

$D :: 'domain \text{ set}$

$v\text{-rel} :: ('domain \times 'domain) \text{ set}$

$n\text{-rel} :: ('domain \times 'domain) \text{ set}$
 $c\text{-rel} :: ('domain \times 'domain) \text{ set}$

definition $FlowPolicy :: 'domain \ FlowPolicy\text{-rec} \Rightarrow bool$

where

$FlowPolicy\ fp \equiv$
 $((v\text{-rel}\ fp) \cup (n\text{-rel}\ fp) \cup (c\text{-rel}\ fp) = ((D\ fp) \times (D\ fp)))$
 $\wedge (v\text{-rel}\ fp) \cap (n\text{-rel}\ fp) = \{\}$
 $\wedge (v\text{-rel}\ fp) \cap (c\text{-rel}\ fp) = \{\}$
 $\wedge (n\text{-rel}\ fp) \cap (c\text{-rel}\ fp) = \{\}$
 $\wedge (\forall d \in (D\ fp). (d, d) \in (v\text{-rel}\ fp))$

type-synonym $('e, 'domain) \text{ dom-type} = 'e \rightarrow 'domain$

definition $dom :: ('e, 'domain) \text{ dom-type} \Rightarrow 'domain \text{ set} \Rightarrow 'e \text{ set} \Rightarrow bool$

where

$dom\ domas\ dset\ es \equiv$
 $(\forall e. \forall d. ((domas\ e = Some\ d) \longrightarrow (e \in es \wedge d \in dset)))$

definition $view\text{-dom} :: 'domain \ FlowPolicy\text{-rec} \Rightarrow 'domain \Rightarrow ('e, 'domain) \text{ dom-type} \Rightarrow 'e \ V\text{-rec}$

where

$view\text{-dom}\ fp\ d\ domas \equiv$
 $(\mid V = \{e. \exists d'. (domas\ e = Some\ d' \wedge (d', d) \in (v\text{-rel}\ fp))\},$
 $N = \{e. \exists d'. (domas\ e = Some\ d' \wedge (d', d) \in (n\text{-rel}\ fp))\},$
 $C = \{e. \exists d'. (domas\ e = Some\ d' \wedge (d', d) \in (c\text{-rel}\ fp))\} \mid)$

end

4.2 Basic Security Predicates

We define all 14 basic security predicates provided as part of MAKS in [3].

theory $BasicSecurityPredicates$

imports $Views \ ./\ Basics/Projection$

begin

definition $areTracesOver :: ('e \text{ list}) \text{ set} \Rightarrow 'e \text{ set} \Rightarrow bool$

where

$areTracesOver\ Tr\ E \equiv$
 $\forall \tau \in Tr. (set\ \tau) \subseteq E$

type-synonym $'e \ BSP = 'e \ V\text{-rec} \Rightarrow (('e \ \text{list}) \ \text{set}) \Rightarrow bool$

definition $BSP\text{-valid} :: 'e \ BSP \Rightarrow bool$

where

BSP-valid bsp \equiv

$$\forall \mathcal{V} \text{ Tr } E. (\text{isViewOn } \mathcal{V} E \wedge \text{areTracesOver } \text{Tr } E) \\ \longrightarrow (\exists \text{ Tr}' . \text{Tr}' \supseteq \text{Tr} \wedge \text{bsp } \mathcal{V} \text{ Tr}')$$

definition *R* :: 'e BSP

where

R \mathcal{V} *Tr* \equiv

$$\forall \tau \in \text{Tr} . \exists \tau' \in \text{Tr} . \tau' \upharpoonright C_{\mathcal{V}} = [] \wedge \tau' \upharpoonright V_{\mathcal{V}} = \tau \upharpoonright V_{\mathcal{V}}$$

lemma *BSP-valid-R*: *BSP-valid R*

<proof>

definition *D* :: 'e BSP

where

D \mathcal{V} *Tr* \equiv

$$\forall \alpha \beta . \forall c \in C_{\mathcal{V}} . ((\beta @ [c] @ \alpha) \in \text{Tr} \wedge \alpha \upharpoonright C_{\mathcal{V}} = []) \\ \longrightarrow (\exists \alpha' \beta' . ((\beta' @ [c] @ \alpha') \in \text{Tr} \wedge \alpha' \upharpoonright V_{\mathcal{V}} = \alpha \upharpoonright V_{\mathcal{V}} \wedge \alpha' \upharpoonright C_{\mathcal{V}} = [] \\ \wedge \beta' \upharpoonright (V_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta \upharpoonright (V_{\mathcal{V}} \cup C_{\mathcal{V}})))$$

lemma *BSP-valid-D*: *BSP-valid D*

<proof>

definition *I* :: 'e BSP

where

I \mathcal{V} *Tr* \equiv

$$\forall \alpha \beta . \forall c \in C_{\mathcal{V}} . ((\beta @ \alpha) \in \text{Tr} \wedge \alpha \upharpoonright C_{\mathcal{V}} = []) \\ \longrightarrow (\exists \alpha' \beta' . ((\beta' @ [c] @ \alpha') \in \text{Tr} \wedge \alpha' \upharpoonright V_{\mathcal{V}} = \alpha \upharpoonright V_{\mathcal{V}} \wedge \alpha' \upharpoonright C_{\mathcal{V}} = [] \\ \wedge \beta' \upharpoonright (V_{\mathcal{V}} \cup C_{\mathcal{V}}) = \beta \upharpoonright (V_{\mathcal{V}} \cup C_{\mathcal{V}})))$$

lemma *BSP-valid-I*: *BSP-valid I*

<proof>

type-synonym 'e Rho = 'e V-rec \Rightarrow 'e set

definition

Adm :: 'e V-rec \Rightarrow 'e Rho \Rightarrow ('e list) set \Rightarrow 'e list \Rightarrow 'e \Rightarrow bool

where

Adm \mathcal{V} ϱ *Tr* β *e* \equiv

$$\exists \gamma . ((\gamma @ [e]) \in \text{Tr} \wedge \gamma \upharpoonright (\varrho \mathcal{V}) = \beta \upharpoonright (\varrho \mathcal{V}))$$

definition *IA* :: 'e Rho \Rightarrow 'e BSP

where

IA ϱ \mathcal{V} *Tr* \equiv

$$\forall \alpha \beta . \forall c \in C_{\mathcal{V}} . ((\beta @ \alpha) \in \text{Tr} \wedge \alpha \upharpoonright C_{\mathcal{V}} = []) \wedge (\text{Adm } \mathcal{V} \varrho \text{ Tr } \beta c) \\ \longrightarrow (\exists \alpha' \beta' . ((\beta' @ [c] @ \alpha') \in \text{Tr} \wedge \alpha' \upharpoonright V_{\mathcal{V}} = \alpha \upharpoonright V_{\mathcal{V}}))$$

$$\wedge \alpha' \uparrow C_{\mathcal{Y}} = [] \wedge \beta' \uparrow (V_{\mathcal{Y}} \cup C_{\mathcal{Y}}) = \beta \uparrow (V_{\mathcal{Y}} \cup C_{\mathcal{Y}}))$$

lemma *BSP-valid-IA: BSP-valid (IA ϱ)*
 ⟨proof⟩

definition *BSD* :: 'e *BSP*

where

BSD \mathcal{V} *Tr* \equiv

$$\forall \alpha \beta. \forall c \in C_{\mathcal{Y}}. ((\beta @ [c] @ \alpha) \in Tr \wedge \alpha \uparrow C_{\mathcal{Y}} = []) \\ \longrightarrow (\exists \alpha'. ((\beta @ \alpha') \in Tr \wedge \alpha' \uparrow V_{\mathcal{Y}} = \alpha \uparrow V_{\mathcal{Y}} \wedge \alpha' \uparrow C_{\mathcal{Y}} = []))$$

lemma *BSP-valid-BSD: BSP-valid BSD*
 ⟨proof⟩

definition *BSI* :: 'e *BSP*

where

BSI \mathcal{V} *Tr* \equiv

$$\forall \alpha \beta. \forall c \in C_{\mathcal{Y}}. ((\beta @ \alpha) \in Tr \wedge \alpha \uparrow C_{\mathcal{Y}} = []) \\ \longrightarrow (\exists \alpha'. ((\beta @ [c] @ \alpha') \in Tr \wedge \alpha' \uparrow V_{\mathcal{Y}} = \alpha \uparrow V_{\mathcal{Y}} \wedge \alpha' \uparrow C_{\mathcal{Y}} = []))$$

lemma *BSP-valid-BSI: BSP-valid BSI*
 ⟨proof⟩

definition *BSIA* :: 'e *Rho* \Rightarrow 'e *BSP*

where

BSIA ϱ \mathcal{V} *Tr* \equiv

$$\forall \alpha \beta. \forall c \in C_{\mathcal{Y}}. ((\beta @ \alpha) \in Tr \wedge \alpha \uparrow C_{\mathcal{Y}} = [] \wedge (Adm \mathcal{V} \varrho Tr \beta c)) \\ \longrightarrow (\exists \alpha'. ((\beta @ [c] @ \alpha') \in Tr \wedge \alpha' \uparrow V_{\mathcal{Y}} = \alpha \uparrow V_{\mathcal{Y}} \wedge \alpha' \uparrow C_{\mathcal{Y}} = []))$$

lemma *BSP-valid-BSIA: BSP-valid (BSIA ϱ)*
 ⟨proof⟩

record 'e *Gamma* =

Nabla :: 'e *set*

Delta :: 'e *set*

Upsilon :: 'e *set*

abbreviation *GammaNabla* :: 'e *Gamma* \Rightarrow 'e *set*

(∇ - [100] 1000)

where

$\nabla_{\Gamma} \equiv (Nabla \Gamma)$

abbreviation *GammaDelta* :: 'e *Gamma* \Rightarrow 'e *set*

(Δ - [100] 1000)

where

$\Delta_{\Gamma} \equiv (Delta \Gamma)$

abbreviation $\text{GammaUpsilon} :: 'e \text{ Gamma} \Rightarrow 'e \text{ set}$
 $(\Upsilon - [100] 1000)$

where

$\Upsilon_{\Gamma} \equiv (\text{Upsilonion } \Gamma)$

definition $\text{FCD} :: 'e \text{ Gamma} \Rightarrow 'e \text{ BSP}$

where

$\text{FCD } \Gamma \mathcal{V} \text{ Tr} \equiv$

$\forall \alpha \beta. \forall c \in (\mathcal{C}_{\mathcal{V}} \cap \Upsilon_{\Gamma}). \forall v \in (V_{\mathcal{V}} \cap \nabla_{\Gamma}).$
 $((\beta @ [c, v] @ \alpha) \in \text{Tr} \wedge \alpha \upharpoonright \mathcal{C}_{\mathcal{V}} = [])$
 $\longrightarrow (\exists \alpha'. \exists \delta'. (\text{set } \delta') \subseteq (N_{\mathcal{V}} \cap \Delta_{\Gamma})$
 $\wedge ((\beta @ \delta' @ [v] @ \alpha') \in \text{Tr}$
 $\wedge \alpha' \upharpoonright V_{\mathcal{V}} = \alpha \upharpoonright V_{\mathcal{V}} \wedge \alpha' \upharpoonright \mathcal{C}_{\mathcal{V}} = []))$

lemma BSP-valid-FCD : $\text{BSP-valid } (\text{FCD } \Gamma)$

$\langle \text{proof} \rangle$

definition $\text{FCI} :: 'e \text{ Gamma} \Rightarrow 'e \text{ BSP}$

where

$\text{FCI } \Gamma \mathcal{V} \text{ Tr} \equiv$

$\forall \alpha \beta. \forall c \in (\mathcal{C}_{\mathcal{V}} \cap \Upsilon_{\Gamma}). \forall v \in (V_{\mathcal{V}} \cap \nabla_{\Gamma}).$
 $((\beta @ [v] @ \alpha) \in \text{Tr} \wedge \alpha \upharpoonright \mathcal{C}_{\mathcal{V}} = [])$
 $\longrightarrow (\exists \alpha'. \exists \delta'. (\text{set } \delta') \subseteq (N_{\mathcal{V}} \cap \Delta_{\Gamma})$
 $\wedge ((\beta @ [c] @ \delta' @ [v] @ \alpha') \in \text{Tr}$
 $\wedge \alpha' \upharpoonright V_{\mathcal{V}} = \alpha \upharpoonright V_{\mathcal{V}} \wedge \alpha' \upharpoonright \mathcal{C}_{\mathcal{V}} = []))$

lemma BSP-valid-FCI : $\text{BSP-valid } (\text{FCI } \Gamma)$

$\langle \text{proof} \rangle$

definition $\text{FCIA} :: 'e \text{ Rho} \Rightarrow 'e \text{ Gamma} \Rightarrow 'e \text{ BSP}$

where

$\text{FCIA } \varrho \Gamma \mathcal{V} \text{ Tr} \equiv$

$\forall \alpha \beta. \forall c \in (\mathcal{C}_{\mathcal{V}} \cap \Upsilon_{\Gamma}). \forall v \in (V_{\mathcal{V}} \cap \nabla_{\Gamma}).$
 $((\beta @ [v] @ \alpha) \in \text{Tr} \wedge \alpha \upharpoonright \mathcal{C}_{\mathcal{V}} = [] \wedge (\text{Adm } \mathcal{V} \varrho \text{ Tr } \beta c))$
 $\longrightarrow (\exists \alpha'. \exists \delta'. (\text{set } \delta') \subseteq (N_{\mathcal{V}} \cap \Delta_{\Gamma})$
 $\wedge ((\beta @ [c] @ \delta' @ [v] @ \alpha') \in \text{Tr}$
 $\wedge \alpha' \upharpoonright V_{\mathcal{V}} = \alpha \upharpoonright V_{\mathcal{V}} \wedge \alpha' \upharpoonright \mathcal{C}_{\mathcal{V}} = []))$

lemma BSP-valid-FCIA : $\text{BSP-valid } (\text{FCIA } \varrho \Gamma)$

$\langle \text{proof} \rangle$

definition $\text{SR} :: 'e \text{ BSP}$

where

$\text{SR } \mathcal{V} \text{ Tr} \equiv \forall \tau \in \text{Tr}. \tau \upharpoonright (V_{\mathcal{V}} \cup N_{\mathcal{V}}) \in \text{Tr}$

lemma BSP-valid SR

<proof>

definition $SD :: 'e \text{ BSP}$

where

$SD \mathcal{V} Tr \equiv$

$\forall \alpha \beta. \forall c \in C_{\mathcal{V}}. ((\beta @ [c] @ \alpha) \in Tr \wedge \alpha \upharpoonright C_{\mathcal{V}} = []) \longrightarrow \beta @ \alpha \in Tr$

lemma *BSP-valid SD*

<proof>

definition $SI :: 'e \text{ BSP}$

where

$SI \mathcal{V} Tr \equiv$

$\forall \alpha \beta. \forall c \in C_{\mathcal{V}}. ((\beta @ \alpha) \in Tr \wedge \alpha \upharpoonright C_{\mathcal{V}} = []) \longrightarrow \beta @ [c] @ \alpha \in Tr$

lemma *BSP-valid SI*

<proof>

definition $SIA :: 'e \text{ Rho} \Rightarrow 'e \text{ BSP}$

where

$SIA \varrho \mathcal{V} Tr \equiv$

$\forall \alpha \beta. \forall c \in C_{\mathcal{V}}. ((\beta @ \alpha) \in Tr \wedge \alpha \upharpoonright C_{\mathcal{V}} = [] \wedge (Adm \mathcal{V} \varrho Tr \beta c))$
 $\longrightarrow (\beta @ [c] @ \alpha) \in Tr$

lemma *BSP-valid (SIA ϱ)*

<proof>

end

4.3 Information-Flow Properties

We define the notion of information-flow properties from [3].

theory *InformationFlowProperties*

imports *BasicSecurityPredicates*

begin

type-synonym $'e \text{ SP} = ('e \text{ BSP}) \text{ set}$

type-synonym $'e \text{ IFP-type} = ('e \text{ V-rec set}) \times 'e \text{ SP}$

definition *IFP-valid* $:: 'e \text{ set} \Rightarrow 'e \text{ IFP-type} \Rightarrow \text{bool}$

where

$IFP\text{-valid } E \text{ ifp} \equiv$

$\forall \mathcal{V} \in (\text{fst ifp}). \text{isViewOn } \mathcal{V} E$
 $\wedge (\forall \text{BSP} \in (\text{snd ifp}). \text{BSP-valid BSP})$

definition *IFPIsSatisfied* :: 'e IFP-type ⇒ ('e list) set ⇒ bool
where
IFPIsSatisfied ifp Tr ≡
 ∀ V∈(fst ifp). ∀ BSP∈(snd ifp). BSP ∨ Tr
end

4.4 Property Library

We define the representations of several possibilistic information-flow properties from the literature that are provided as part of MAKS in [3].

theory *PropertyLibrary*
imports *InformationFlowProperties ../SystemSpecification/EventSystems ../Verification/Basics/BSPTaxonomy*
begin

definition
HighInputsConfidential :: 'e set ⇒ 'e set ⇒ 'e set ⇒ 'e V-rec
where
HighInputsConfidential L H IE ≡ (| V=L, N=H-IE, C=H ∩ IE |)

definition *HighConfidential* :: 'e set ⇒ 'e set ⇒ 'e V-rec
where
HighConfidential L H ≡ (| V=L, N={}, C=H |)

fun *interleaving* :: 'e list ⇒ 'e list ⇒ ('e list) set
where
interleaving t1 [] = {t1} |
interleaving [] t2 = {t2} |
interleaving (e1 # t1) (e2 # t2) =
 {t. (∃ t'. t=(e1 # t') ∧ t' ∈ *interleaving* t1 (e2 # t2))}
 ∪ {t. (∃ t'. t=(e2 # t') ∧ t' ∈ *interleaving* (e1 # t1) t2)}

definition *GNI* :: 'e set ⇒ 'e set ⇒ 'e set ⇒ 'e IFP-type
where
GNI L H IE ≡ ({*HighInputsConfidential* L H IE}, {BSD, BSI})

lemma *GNI-valid*: L ∩ H = {} ⇒ IFP-valid (L ∪ H) (*GNI* L H IE)
 ⟨proof⟩

definition *litGNI* :: 'e set ⇒ 'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool
where
litGNI L H IE Tr ≡
 ∀ t1 t2 t3.

$$\begin{aligned}
t1 @ t2 \in Tr \wedge t3 \uparrow (L \cup (H - IE)) &= t2 \uparrow (L \cup (H - IE)) \\
\rightarrow (\exists t4. t1 @ t4 \in Tr \wedge t4 \uparrow (L \cup (H \cap IE)) &= t3 \uparrow (L \cup (H \cap IE)))
\end{aligned}$$

definition *IBGNI* :: 'e set \Rightarrow 'e set \Rightarrow 'e set \Rightarrow 'e IFP-type
where *IBGNI L H IE* \equiv ({HighInputsConfidential L H IE}, {D, I})

lemma *IBGNI-valid*: $L \cap H = \{\} \implies$ IFP-valid $(L \cup H)$ (*IBGNI L H IE*)
⟨proof⟩

definition

litIBGNI :: 'e set \Rightarrow 'e set \Rightarrow 'e set \Rightarrow ('e list) set \Rightarrow bool

where

litIBGNI L H IE Tr \equiv
 $\forall \tau\text{-}l \in Tr. \forall t\text{-}hi\ t.$
 $(set\ t\text{-}hi) \subseteq (H \cap IE) \wedge t \in interleaving\ t\text{-}hi\ (\tau\text{-}l \uparrow L)$
 $\rightarrow (\exists \tau' \in Tr. \tau' \uparrow (L \cup (H \cap IE)) = t)$

definition *FC* :: 'e set \Rightarrow 'e set \Rightarrow 'e set \Rightarrow 'e IFP-type

where

FC L H IE \equiv
({HighInputsConfidential L H IE},
{BSD, BSI, (FCD \uparrow Nabla=IE, Delta={}, Upsilon=IE \uparrow)},
(FCI \uparrow Nabla=IE, Delta={}, Upsilon=IE \uparrow) })

lemma *FC-valid*: $L \cap H = \{\} \implies$ IFP-valid $(L \cup H)$ (*FC L H IE*)
⟨proof⟩

definition *litFC* :: 'e set \Rightarrow 'e set \Rightarrow 'e set \Rightarrow ('e list) set \Rightarrow bool

where

litFC L H IE Tr \equiv
 $\forall t\text{-}1\ t\text{-}2. \forall hi \in (H \cap IE).$
($\forall li \in (L \cap IE).$
 $t\text{-}1 @ [li] @ t\text{-}2 \in Tr \wedge t\text{-}2 \uparrow (H \cap IE) = []$
 $\rightarrow (\exists t\text{-}3. t\text{-}1 @ [hi] @ [li] @ t\text{-}3 \in Tr$
 $\wedge t\text{-}3 \uparrow L = t\text{-}2 \uparrow L \wedge t\text{-}3 \uparrow (H \cap IE) = [])$
 $\wedge (t\text{-}1 @ t\text{-}2 \in Tr \wedge t\text{-}2 \uparrow (H \cap IE) = []$
 $\rightarrow (\exists t\text{-}3. t\text{-}1 @ [hi] @ t\text{-}3 \in Tr$
 $\wedge t\text{-}3 \uparrow L = t\text{-}2 \uparrow L \wedge t\text{-}3 \uparrow (H \cap IE) = [])$
 $\wedge (\forall li \in (L \cap IE).$
 $t\text{-}1 @ [hi] @ [li] @ t\text{-}2 \in Tr \wedge t\text{-}2 \uparrow (H \cap IE) = []$
 $\rightarrow (\exists t\text{-}3. t\text{-}1 @ [li] @ t\text{-}3 \in Tr$
 $\wedge t\text{-}3 \uparrow L = t\text{-}2 \uparrow L \wedge t\text{-}3 \uparrow (H \cap IE) = [])$
 $\wedge (t\text{-}1 @ [hi] @ t\text{-}2 \in Tr \wedge t\text{-}2 \uparrow (H \cap IE) = []$
 $\rightarrow (\exists t\text{-}3. t\text{-}1 @ t\text{-}3 \in Tr$
 $\wedge t\text{-}3 \uparrow L = t\text{-}2 \uparrow L \wedge t\text{-}3 \uparrow (H \cap IE) = [])$

)

definition $NDO :: 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ IFP\text{-}type$

where

$NDO\ UI\ L\ H \equiv$

$(\{HighConfidential\ L\ H\}, \{BSD, (BSIA\ (\lambda\ \mathcal{V}. C_{\mathcal{V}} \cup (V_{\mathcal{V}} \cap UI))\})\})$

lemma $NDO\text{-}valid: L \cap H = \{\} \Longrightarrow IFP\text{-}valid\ (L \cup H)\ (NDO\ UI\ L\ H)$

$\langle proof \rangle$

definition $litNDO :: 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ set \Rightarrow ('e\ list)\ set \Rightarrow bool$

where

$litNDO\ UI\ L\ H\ Tr \equiv$

$\forall \tau\text{-}l \in Tr. \forall \tau\text{-}hlui \in Tr. \forall t.$

$t \upharpoonright L = \tau\text{-}l \upharpoonright L \wedge t \upharpoonright (H \cup (L \cap UI)) = \tau\text{-}hlui \upharpoonright (H \cup (L \cap UI)) \longrightarrow t \in Tr$

definition $NF :: 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ IFP\text{-}type$

where

$NF\ L\ H \equiv (\{HighConfidential\ L\ H\}, \{R\})$

lemma $NF\text{-}valid: L \cap H = \{\} \Longrightarrow IFP\text{-}valid\ (L \cup H)\ (NF\ L\ H)$

$\langle proof \rangle$

definition $litNF :: 'e\ set \Rightarrow 'e\ set \Rightarrow ('e\ list)\ set \Rightarrow bool$

where

$litNF\ L\ H\ Tr \equiv \forall \tau \in Tr. \tau \upharpoonright L \in Tr$

definition $GNF :: 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ IFP\text{-}type$

where

$GNF\ L\ H\ IE \equiv (\{HighInputsConfidential\ L\ H\ IE\}, \{R\})$

lemma $GNF\text{-}valid: L \cap H = \{\} \Longrightarrow IFP\text{-}valid\ (L \cup H)\ (GNF\ L\ H\ IE)$

$\langle proof \rangle$

definition $litGNF :: 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ set \Rightarrow ('e\ list)\ set \Rightarrow bool$

where

$litGNF\ L\ H\ IE\ Tr \equiv$

$\forall \tau \in Tr. \exists \tau' \in Tr. \tau' \upharpoonright (H \cap IE) = [] \wedge \tau' \upharpoonright L = \tau \upharpoonright L$

definition $SEP :: 'e\ set \Rightarrow 'e\ set \Rightarrow 'e\ IFP\text{-}type$

where

$SEP\ L\ H \equiv (\{HighConfidential\ L\ H\}, \{BSD, (BSIA\ (\lambda\ \mathcal{V}. C_{\mathcal{V}}))\})$

lemma *SEP-valid*: $L \cap H = \{\} \implies IFP\text{-valid}\ (L \cup H)\ (SEP\ L\ H)$
<proof>

definition *litSEP* :: 'e set \Rightarrow 'e set \Rightarrow ('e list) set \Rightarrow bool

where

$litSEP\ L\ H\ Tr \equiv$

$\forall \tau\text{-}l \in Tr. \forall \tau\text{-}h \in Tr.$

$interleaving\ (\tau\text{-}l \upharpoonright L)\ (\tau\text{-}h \upharpoonright H) \subseteq \{\tau \in Tr . \tau \upharpoonright L = \tau\text{-}l \upharpoonright L\}$

definition *PSP* :: 'e set \Rightarrow 'e set \Rightarrow 'e IFP-type

where

$PSP\ L\ H \equiv$

$(\{HighConfidential\ L\ H\}, \{BSD, (BSIA\ (\lambda\ \mathcal{V}. C_{\mathcal{V}} \cup N_{\mathcal{V}} \cup V_{\mathcal{V}}))\})$

lemma *PSP-valid*: $L \cap H = \{\} \implies IFP\text{-valid}\ (L \cup H)\ (PSP\ L\ H)$
<proof>

definition *litPSP* :: 'e set \Rightarrow 'e set \Rightarrow ('e list) set \Rightarrow bool

where

$litPSP\ L\ H\ Tr \equiv$

$(\forall \tau \in Tr. \tau \upharpoonright L \in Tr)$

$\wedge (\forall \alpha\ \beta. (\beta @ \alpha) \in Tr \wedge (\alpha \upharpoonright H) = []$

$\longrightarrow (\forall h \in H. \beta @ [h] \in Tr \longrightarrow \beta @ [h] @ \alpha \in Tr))$

end

5 Verification

5.1 Basic Definitions

We define when an event system and a state-event system are secure given an information-flow property.

theory *SecureSystems*

imports *../SystemSpecification/StateEventSystems*

../SecuritySpecification/InformationFlowProperties

begin

locale *SecureESIFP* =

fixes *ES* :: 'e ES-rec

and *IFP* :: 'e IFP-type

assumes *validES*: *ES-valid ES*

and *validIFPES*: *IFP-valid E_{ES} IFP*

```

context SecureESIFP
begin

definition ES-sat-IFP :: bool
where
ES-sat-IFP  $\equiv$  IFPIsSatisfied IFP TrES

end

locale SecureSESIFP =
fixes SES :: ('s, 'e) SES-rec
and IFP :: 'e IFP-type

assumes validSES: SES-valid SES
and validIFPSES: IFP-valid ESES IFP

sublocale SecureSESIFP  $\subseteq$  SecureESIFP induceES SES IFP
 $\langle$ proof $\rangle$ 

context SecureSESIFP
begin

abbreviation SES-sat-IFP
where
SES-sat-IFP  $\equiv$  ES-sat-IFP

end

end

```

5.2 Taxonomy Results

We prove the taxonomy results from [3].

```

theory BSPTaxonomy
imports ../SystemSpecification/EventSystems
        ../SecuritySpecification/BasicSecurityPredicates
begin

locale BSPTaxonomyDifferentCorrections =
fixes ES :: 'e ES-rec

```

and $\mathcal{V} :: 'e$ *V-rec*

assumes *validES*: *ES-valid ES*

and *VIIsViewOnE*: *isViewOn* \mathcal{V} E_{ES}

locale *BSPTaxonomyDifferentViews* =

fixes *ES* :: 'e *ES-rec*

and $\mathcal{V}_1 :: 'e$ *V-rec*

and $\mathcal{V}_2 :: 'e$ *V-rec*

assumes *validES*: *ES-valid ES*

and \mathcal{V}_1 *IsViewOnE*: *isViewOn* \mathcal{V}_1 E_{ES}

and \mathcal{V}_2 *IsViewOnE*: *isViewOn* \mathcal{V}_2 E_{ES}

locale *BSPTaxonomyDifferentViewsFirstDim* = *BSPTaxonomyDifferentViews* +

assumes *V2-subset-V1*: $V_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1}$

and *N2-supset-N1*: $N_{\mathcal{V}_2} \supseteq N_{\mathcal{V}_1}$

and *C2-subset-C1*: $C_{\mathcal{V}_2} \subseteq C_{\mathcal{V}_1}$

sublocale *BSPTaxonomyDifferentViewsFirstDim* \subseteq *BSPTaxonomyDifferentViews*

<proof>

locale *BSPTaxonomyDifferentViewsSecondDim* = *BSPTaxonomyDifferentViews* +

assumes *V2-subset-V1*: $V_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1}$

and *N2-supset-N1*: $N_{\mathcal{V}_2} \supseteq N_{\mathcal{V}_1}$

and *C2-equals-C1*: $C_{\mathcal{V}_2} = C_{\mathcal{V}_1}$

sublocale *BSPTaxonomyDifferentViewsSecondDim* \subseteq *BSPTaxonomyDifferentViews*

<proof>

context *BSPTaxonomyDifferentCorrections*

begin

lemma *SR-implies-R*:

$SR \vee Tr_{ES} \implies R \vee Tr_{ES}$

<proof>

lemma *SD-implies-BSD* :

$(SD \vee Tr_{ES}) \implies BSD \vee Tr_{ES}$

<proof>

lemma *BSD-implies-D*:

$BSD \vee Tr_{ES} \implies D \vee Tr_{ES}$

<proof>

lemma *SD-implies-SR*:

$SD \vee Tr_{ES} \implies SR \vee Tr_{ES}$

<proof>

lemma *D-implies-R:*

$$D \mathcal{V} \text{Tr}_{ES} \Longrightarrow R \mathcal{V} \text{Tr}_{ES}$$

<proof>

lemma *SR-implies-R-for-modified-view :*

$$\llbracket SR \mathcal{V} \text{Tr}_{ES}; \mathcal{V}' = \langle V = V_{\mathcal{V}} \cup N_{\mathcal{V}}, N = \{\}, C = C_{\mathcal{V}} \rangle \rrbracket \Longrightarrow R \mathcal{V}' \text{Tr}_{ES}$$

<proof>

lemma *R-implies-SR-for-modified-view :*

$$\llbracket R \mathcal{V}' \text{Tr}_{ES}; \mathcal{V}' = \langle V = V_{\mathcal{V}} \cup N_{\mathcal{V}}, N = \{\}, C = C_{\mathcal{V}} \rangle \rrbracket \Longrightarrow SR \mathcal{V} \text{Tr}_{ES}$$

<proof>

lemma *SD-implies-BSD-for-modified-view :*

$$\llbracket SD \mathcal{V} \text{Tr}_{ES}; \mathcal{V}' = \langle V = V_{\mathcal{V}} \cup N_{\mathcal{V}}, N = \{\}, C = C_{\mathcal{V}} \rangle \rrbracket \Longrightarrow BSD \mathcal{V}' \text{Tr}_{ES}$$

<proof>

lemma *BSD-implies-SD-for-modified-view :*

$$\llbracket BSD \mathcal{V}' \text{Tr}_{ES}; \mathcal{V}' = \langle V = V_{\mathcal{V}} \cup N_{\mathcal{V}}, N = \{\}, C = C_{\mathcal{V}} \rangle \rrbracket \Longrightarrow SD \mathcal{V} \text{Tr}_{ES}$$

<proof>

lemma *SD-implies-FCD:*

$$(SD \mathcal{V} \text{Tr}_{ES}) \Longrightarrow FCD \Gamma \mathcal{V} \text{Tr}_{ES}$$

<proof>

lemma *SI-implies-BSI :*

$$(SI \mathcal{V} \text{Tr}_{ES}) \Longrightarrow BSI \mathcal{V} \text{Tr}_{ES}$$

<proof>

lemma *BSI-implies-I:*

$$(BSI \mathcal{V} \text{Tr}_{ES}) \Longrightarrow (I \mathcal{V} \text{Tr}_{ES})$$

<proof>

lemma *SIA-implies-BSIA:*

$$(SIA \varrho \mathcal{V} \text{Tr}_{ES}) \Longrightarrow (BSIA \varrho \mathcal{V} \text{Tr}_{ES})$$

<proof>

lemma *BSIA-implies-IA:*

$$(BSIA \varrho \mathcal{V} \text{Tr}_{ES}) \Longrightarrow (IA \varrho \mathcal{V} \text{Tr}_{ES})$$

<proof>

lemma *SI-implies-SIA:*
 $SI \mathcal{V} Tr_{ES} \Longrightarrow SIA \varrho \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *BSI-implies-BSIA:*
 $BSI \mathcal{V} Tr_{ES} \Longrightarrow BSIA \varrho \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *I-implies-IA:*
 $I \mathcal{V} Tr_{ES} \Longrightarrow IA \varrho \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *SI-implies-BSI-for-modified-view :*
 $\llbracket SI \mathcal{V} Tr_{ES}; \mathcal{V}' = (V = V_{\mathcal{V}} \cup N_{\mathcal{V}}, N = \{\}, C = C_{\mathcal{V}}) \rrbracket \Longrightarrow BSI \mathcal{V}' Tr_{ES}$
 ⟨proof⟩

lemma *BSI-implies-SI-for-modified-view :*
 $\llbracket BSI \mathcal{V}' Tr_{ES}; \mathcal{V}' = (V = V_{\mathcal{V}} \cup N_{\mathcal{V}}, N = \{\}, C = C_{\mathcal{V}}) \rrbracket \Longrightarrow SI \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *SIA-implies-BSIA-for-modified-view :*
 $\llbracket SIA \varrho \mathcal{V} Tr_{ES}; \mathcal{V}' = (V = V_{\mathcal{V}} \cup N_{\mathcal{V}}, N = \{\}, C = C_{\mathcal{V}}) ; \varrho \mathcal{V} = \varrho' \mathcal{V}' \rrbracket \Longrightarrow BSIA \varrho' \mathcal{V}' Tr_{ES}$
 ⟨proof⟩

lemma *BSIA-implies-SIA-for-modified-view :*
 $\llbracket BSIA \varrho' \mathcal{V}' Tr_{ES}; \mathcal{V}' = (V = V_{\mathcal{V}} \cup N_{\mathcal{V}}, N = \{\}, C = C_{\mathcal{V}}) ; \varrho \mathcal{V} = \varrho' \mathcal{V}' \rrbracket \Longrightarrow SIA \varrho \mathcal{V} Tr_{ES}$
 ⟨proof⟩
end

lemma *Adm-implies-Adm-for-modified-rho:*
 $\llbracket Adm \mathcal{V}_2 \varrho_2 Tr \alpha e; \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1) \rrbracket \Longrightarrow Adm \mathcal{V}_1 \varrho_1 Tr \alpha e$
 ⟨proof⟩

context *BSPTaxonomyDifferentCorrections*
begin

lemma *SI-implies-FCI:*
 $(SI \mathcal{V} Tr_{ES}) \Longrightarrow FCI \Gamma \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *SIA-implies-FCIA:*
 $(SIA \varrho \mathcal{V} Tr_{ES}) \Longrightarrow FCIA \varrho \Gamma \mathcal{V} Tr_{ES}$

<proof>

lemma *FCI-implies-FCIA:*
 $(FCI \Gamma \vee Tr_{ES}) \implies FCIA \varrho \Gamma \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-SR-C-empty:*
 $C_{\mathcal{V}} = \{\} \implies SR \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-R-C-empty:*
 $C_{\mathcal{V}} = \{\} \implies R \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-SD-C-empty:*
 $C_{\mathcal{V}} = \{\} \implies SD \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-BSD-C-empty:*
 $C_{\mathcal{V}} = \{\} \implies BSD \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-D-C-empty:*
 $C_{\mathcal{V}} = \{\} \implies D \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-FCD-C-empty:*
 $C_{\mathcal{V}} = \{\} \implies FCD \Gamma \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-R-V-empty:*
 $V_{\mathcal{V}} = \{\} \implies R \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-BSD-V-empty:*
 $V_{\mathcal{V}} = \{\} \implies BSD \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-D-V-empty:*
 $V_{\mathcal{V}} = \{\} \implies D \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-FCD-V-empty:*
 $V_{\mathcal{V}} = \{\} \implies FCD \Gamma \vee Tr_{ES}$
<proof>

lemma *Trivially-fulfilled-FCD-Nabla-Y-empty:*
 $[\nabla_{\Gamma} = \{\} \vee \Upsilon_{\Gamma} = \{\}] \implies FCD \Gamma \vee Tr_{ES}$

<proof>

lemma *Trivially-fulfilled-FCD-N-subseteq-Δ-and-BSD:*

$$\llbracket N_{\mathcal{V}} \subseteq \Delta_{\Gamma}; BSD \vee Tr_{ES} \rrbracket \implies FCD \Gamma \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-SI-C-empty:*

$$C_{\mathcal{V}} = \{\} \implies SI \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-BSI-C-empty:*

$$C_{\mathcal{V}} = \{\} \implies BSI \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-I-C-empty:*

$$C_{\mathcal{V}} = \{\} \implies I \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-FCI-C-empty:*

$$C_{\mathcal{V}} = \{\} \implies FCI \Gamma \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-SIA-C-empty:*

$$C_{\mathcal{V}} = \{\} \implies SIA \varrho \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-BSIA-C-empty:*

$$C_{\mathcal{V}} = \{\} \implies BSIA \varrho \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-IA-C-empty:*

$$C_{\mathcal{V}} = \{\} \implies IA \varrho \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-FCIA-C-empty:*

$$C_{\mathcal{V}} = \{\} \implies FCIA \Gamma \varrho \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-FCI-V-empty:*

$$V_{\mathcal{V}} = \{\} \implies FCI \Gamma \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-FCIA-V-empty:*

$$V_{\mathcal{V}} = \{\} \implies FCIA \varrho \Gamma \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-BSIA-V-empty-rho-subseteq-C-N:*

$$\llbracket V_{\mathcal{V}} = \{\}; \varrho \vee \supseteq (C_{\mathcal{V}} \cup N_{\mathcal{V}}) \rrbracket \implies BSIA \varrho \vee Tr_{ES}$$

<proof>

lemma *Trivially-fulfilled-IA-V-empty-rho-subseteq-C-N:*

$\llbracket V_{\mathcal{V}} = \{\}; \varrho \mathcal{V} \supseteq (C_{\mathcal{V}} \cup N_{\mathcal{V}}) \rrbracket \Longrightarrow IA \varrho \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *Trivially-fulfilled-BSI-V-empty-total-ES-C:*
 $\llbracket V_{\mathcal{V}} = \{\}; total\ ES\ C_{\mathcal{V}} \rrbracket \Longrightarrow BSI \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *Trivially-fulfilled-I-V-empty-total-ES-C:*
 $\llbracket V_{\mathcal{V}} = \{\}; total\ ES\ C_{\mathcal{V}} \rrbracket \Longrightarrow I \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *Trivially-fulfilled-FCI-Nabla-Υ-empty:*
 $\llbracket \nabla_{\Gamma} = \{\} \vee \Upsilon_{\Gamma} = \{\} \rrbracket \Longrightarrow FCI \Gamma \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *Trivially-fulfilled-FCIA-Nabla-Υ-empty:*
 $\llbracket \nabla_{\Gamma} = \{\} \vee \Upsilon_{\Gamma} = \{\} \rrbracket \Longrightarrow FCIA \varrho \Gamma \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *Trivially-fulfilled-FCI-N-subseteq-Δ-and-BSI:*
 $\llbracket N_{\mathcal{V}} \subseteq \Delta_{\Gamma}; BSI \mathcal{V} Tr_{ES} \rrbracket \Longrightarrow FCI \Gamma \mathcal{V} Tr_{ES}$
 ⟨proof⟩

lemma *Trivially-fulfilled-FCIA-N-subseteq-Δ-and-BSIA:*
 $\llbracket N_{\mathcal{V}} \subseteq \Delta_{\Gamma}; BSIA \varrho \mathcal{V} Tr_{ES} \rrbracket \Longrightarrow FCIA \varrho \Gamma \mathcal{V} Tr_{ES}$
 ⟨proof⟩

end

context *BSPTaxonomyDifferentViewsFirstDim*
begin

lemma *R-implies-R-for-modified-view:*
 $R \mathcal{V}_1 Tr_{ES} \Longrightarrow R \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *BSD-implies-BSD-for-modified-view:*
 $BSD \mathcal{V}_1 Tr_{ES} \Longrightarrow BSD \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *D-implies-D-for-modified-view:*
 $D \mathcal{V}_1 Tr_{ES} \Longrightarrow D \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

end

context *BSPTaxonomyDifferentViewsSecondDim*
begin

lemma *FCD-implies-FCD-for-modified-view-gamma:*
 $\llbracket FCD \Gamma_1 \mathcal{V}_1 Tr_{ES};$
 $V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}; N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}; C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1} \rrbracket$

$\implies FCD \Gamma_2 \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *SI-implies-SI-for-modified-view :*
 $SI \mathcal{V}_1 Tr_{ES} \implies SI \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *BSI-implies-BSI-for-modified-view :*
 $BSI \mathcal{V}_1 Tr_{ES} \implies BSI \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *I-implies-I-for-modified-view :*
 $I \mathcal{V}_1 Tr_{ES} \implies I \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *SIA-implies-SIA-for-modified-view :*
 $\llbracket SIA \varrho_1 \mathcal{V}_1 Tr_{ES}; \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1) \rrbracket \implies SIA \varrho_2 \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *BSIA-implies-BSIA-for-modified-view :*
 $\llbracket BSIA \varrho_1 \mathcal{V}_1 Tr_{ES}; \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1) \rrbracket \implies BSIA \varrho_2 \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *IA-implies-IA-for-modified-view :*
 $\llbracket IA \varrho_1 \mathcal{V}_1 Tr_{ES}; \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1) \rrbracket \implies IA \varrho_2 \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *FCI-implies-FCI-for-modified-view-gamma:*
 $\llbracket FCI \Gamma_1 \mathcal{V}_1 Tr_{ES};$
 $V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}; N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}; C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1} \rrbracket$
 $\implies FCI \Gamma_2 \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

lemma *FCIA-implies-FCIA-for-modified-view-rho-gamma:*
 $\llbracket FCIA \varrho_1 \Gamma_1 \mathcal{V}_1 Tr_{ES}; \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1);$
 $V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}; N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}; C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1} \rrbracket$
 $\implies FCIA \varrho_2 \Gamma_2 \mathcal{V}_2 Tr_{ES}$
 ⟨proof⟩

end

end

5.3 Unwinding

We define the unwinding conditions provided in [3] and prove the unwinding theorems from [3] that use these unwinding conditions.

5.3.1 Unwinding Conditions

```

theory UnwindingConditions
imports ../Basics/BSPTaxonomy
        ../SystemSpecification/StateEventSystems
begin

locale Unwinding =
fixes SES :: ('s, 'e) SES-rec
and V :: 'e V-rec

assumes validSES: SES-valid SES
and validVU: isViewOn V ESES

sublocale Unwinding  $\subseteq$  BSPTaxonomyDifferentCorrections induceES SES V
  <proof>

context Unwinding
begin

definition osc :: 's rel  $\Rightarrow$  bool
where
osc ur  $\equiv$ 
   $\forall s1 \in S_{SES}. \forall s1' \in S_{SES}. \forall s2' \in S_{SES}. \forall e \in (E_{SES} - C_V).$ 
  (reachable SES s1  $\wedge$  reachable SES s1'
    $\wedge$  s1' e  $\longrightarrow_{SES}$  s2'  $\wedge$  (s1', s1)  $\in$  ur)
   $\longrightarrow$  ( $\exists s2 \in S_{SES}. \exists \delta. \delta \upharpoonright C_V = [] \wedge \delta \upharpoonright V_V = [e] \upharpoonright V_V$ 
    $\wedge$  s1  $\delta \Longrightarrow_{SES}$  s2  $\wedge$  (s2', s2)  $\in$  ur)

definition lrf :: 's rel  $\Rightarrow$  bool
where
lrf ur  $\equiv$ 
   $\forall s \in S_{SES}. \forall s' \in S_{SES}. \forall c \in C_V.$ 
  ((reachable SES s  $\wedge$  s c  $\longrightarrow_{SES}$  s')  $\longrightarrow$  (s', s)  $\in$  ur)

definition lrb :: 's rel  $\Rightarrow$  bool
where
lrb ur  $\equiv$   $\forall s \in S_{SES}. \forall c \in C_V.$ 
  (reachable SES s  $\longrightarrow$  ( $\exists s' \in S_{SES}. (s c \longrightarrow_{SES} s' \wedge ((s, s') \in ur)$ )))

```

definition $fcrf :: 'e \text{ Gamma} \Rightarrow 's \text{ rel} \Rightarrow \text{bool}$

where

$fcrf \ \Gamma \ ur \equiv$
 $\forall c \in (C_{\mathcal{V}} \cap \Upsilon_{\Gamma}). \forall v \in (V_{\mathcal{V}} \cap \nabla_{\Gamma}). \forall s \in S_{SES}. \forall s' \in S_{SES}.$
 $((\text{reachable } SES \ s \wedge s \ ([c] \ @ \ [v]) \Longrightarrow_{SES} s')$
 $\longrightarrow (\exists s'' \in S_{SES}. \exists \delta. (\forall d \in (\text{set } \delta). d \in (N_{\mathcal{V}} \cap \Delta_{\Gamma})) \wedge$
 $s \ (\delta \ @ \ [v]) \Longrightarrow_{SES} s'' \wedge (s', s'') \in ur))$

definition $fcrb :: 'e \text{ Gamma} \Rightarrow 's \text{ rel} \Rightarrow \text{bool}$

where

$fcrb \ \Gamma \ ur \equiv$
 $\forall c \in (C_{\mathcal{V}} \cap \Upsilon_{\Gamma}). \forall v \in (V_{\mathcal{V}} \cap \nabla_{\Gamma}). \forall s \in S_{SES}. \forall s'' \in S_{SES}.$
 $((\text{reachable } SES \ s \wedge s \ v \longrightarrow_{SES} s'')$
 $\longrightarrow (\exists s' \in S_{SES}. \exists \delta. (\forall d \in (\text{set } \delta). d \in (N_{\mathcal{V}} \cap \Delta_{\Gamma})) \wedge$
 $s \ ([c] \ @ \ \delta \ @ \ [v]) \Longrightarrow_{SES} s' \wedge (s'', s') \in ur))$

definition $En :: 'e \text{ Rho} \Rightarrow 's \Rightarrow 'e \Rightarrow \text{bool}$

where

$En \ \varrho \ s \ e \equiv$
 $\exists \beta \ \gamma. \exists s' \in S_{SES}. \exists s'' \in S_{SES}.$
 $s0_{SES} \ \beta \Longrightarrow_{SES} s \wedge (\gamma \uparrow (\varrho \ \mathcal{V}) = \beta \uparrow (\varrho \ \mathcal{V}))$
 $\wedge s0_{SES} \ \gamma \Longrightarrow_{SES} s' \wedge s' \ e \longrightarrow_{SES} s''$

definition $lrbe :: 'e \text{ Rho} \Rightarrow 's \text{ rel} \Rightarrow \text{bool}$

where

$lrbe \ \varrho \ ur \equiv$
 $\forall s \in S_{SES}. \forall c \in C_{\mathcal{V}}.$
 $((\text{reachable } SES \ s \wedge (En \ \varrho \ s \ c))$
 $\longrightarrow (\exists s' \in S_{SES}. (s \ c \longrightarrow_{SES} s' \wedge (s, s') \in ur)))$

definition $fcrbe :: 'e \text{ Gamma} \Rightarrow 'e \text{ Rho} \Rightarrow 's \text{ rel} \Rightarrow \text{bool}$

where

$fcrbe \ \Gamma \ \varrho \ ur \equiv$
 $\forall c \in (C_{\mathcal{V}} \cap \Upsilon_{\Gamma}). \forall v \in (V_{\mathcal{V}} \cap \nabla_{\Gamma}). \forall s \in S_{SES}. \forall s'' \in S_{SES}.$
 $((\text{reachable } SES \ s \wedge s \ v \longrightarrow_{SES} s'' \wedge (En \ \varrho \ s \ c))$
 $\longrightarrow (\exists s' \in S_{SES}. \exists \delta. (\forall d \in (\text{set } \delta). d \in (N_{\mathcal{V}} \cap \Delta_{\Gamma})) \wedge$
 $s \ ([c] \ @ \ \delta \ @ \ [v]) \Longrightarrow_{SES} s' \wedge (s'', s') \in ur))$

end

end

5.3.2 Auxiliary Results

theory *AuxiliaryLemmas*

imports *UnwindingConditions*

begin

context *Unwinding*
begin

lemma *osc-property:*

$\bigwedge s1\ s1'. \llbracket \text{osc } ur; s1 \in S_{SES}; s1' \in S_{SES}; \alpha \upharpoonright C_{\mathcal{V}} = \llbracket ;$
 $\text{reachable } SES\ s1; \text{reachable } SES\ s1'; \text{enabled } SES\ s1' \alpha; (s1', s1) \in ur \rrbracket$
 $\implies (\exists \alpha'. \alpha' \upharpoonright C_{\mathcal{V}} = \llbracket \wedge \alpha' \upharpoonright V_{\mathcal{V}} = \alpha \upharpoonright V_{\mathcal{V}} \wedge \text{enabled } SES\ s1\ \alpha')$
 $\langle \text{proof} \rangle$

lemma *path-state-closure:* $\llbracket s \tau \implies_{SES} s'; s \in S_{SES} \rrbracket \implies s' \in S_{SES}$
 $(\text{is } \llbracket ?P\ s\ \tau\ s'; ?S\ s\ SES \rrbracket \implies ?S\ s'\ SES)$
 $\langle \text{proof} \rangle$

theorem *En-to-Adm:*

$\llbracket \text{reachable } SES\ s; \text{En } \varrho\ s\ e \rrbracket$
 $\implies \exists \beta. (s0_{SES} \beta \implies_{SES} s \wedge \text{Adm } \mathcal{V} \varrho\ \text{Tr}(\text{induceES } SES) \beta\ e)$
 $\langle \text{proof} \rangle$

theorem *Adm-to-En:*

$\llbracket \beta \in \text{Tr}(\text{induceES } SES); \text{Adm } \mathcal{V} \varrho\ \text{Tr}(\text{induceES } SES) \beta\ e \rrbracket$
 $\implies \exists s \in S_{SES}. (s0_{SES} \beta \implies_{SES} s \wedge \text{En } \varrho\ s\ e)$
 $\langle \text{proof} \rangle$

lemma *state-from-induceES-trace:*

$\llbracket (\beta @ \alpha) \in \text{Tr}(\text{induceES } SES) \rrbracket$
 $\implies \exists s \in S_{SES}. s0_{SES} \beta \implies_{SES} s \wedge \text{enabled } SES\ s\ \alpha \wedge \text{reachable } SES\ s$
 $\langle \text{proof} \rangle$

lemma *path-split2:* $s0_{SES} (\beta @ \alpha) \implies_{SES} s$

$\implies \exists s' \in S_{SES}. (s0_{SES} \beta \implies_{SES} s' \wedge s' \alpha \implies_{SES} s \wedge \text{reachable } SES\ s')$
 $\langle \text{proof} \rangle$

lemma *path-split-single2:*

$s0_{SES} (\beta @ [x]) \implies_{SES} s$
 $\implies \exists s' \in S_{SES}. (s0_{SES} \beta \implies_{SES} s' \wedge s' x \longrightarrow_{SES} s \wedge \text{reachable } SES\ s')$
 $\langle \text{proof} \rangle$

lemma *modified-view-valid:* $\text{isViewOn } (\llbracket V = (V_{\mathcal{V}} \cup N_{\mathcal{V}}), N = \{\}, C = C_{\mathcal{V}} \rrbracket) E_{SES}$

<proof>

end

end

5.3.3 Unwinding Theorems

theory *UnwindingResults*

imports *AuxiliaryLemmas*

begin

context *Unwinding*

begin

theorem *unwinding-theorem-BSD:*

$\llbracket \text{lr}f \text{ ur}; \text{osc ur} \rrbracket \implies \text{BSD } \mathcal{V} \text{ Tr}_{(\text{induceES SES})}$

<proof>

theorem *unwinding-theorem-BSI:*

$\llbracket \text{lr}b \text{ ur}; \text{osc ur} \rrbracket \implies \text{BSI } \mathcal{V} \text{ Tr}_{(\text{induceES SES})}$

<proof>

theorem *unwinding-theorem-BSIA:*

$\llbracket \text{lr}be \varrho \text{ ur}; \text{osc ur} \rrbracket \implies \text{BSIA } \varrho \mathcal{V} \text{ Tr}_{(\text{induceES SES})}$

<proof>

theorem *unwinding-theorem-FCD:*

$\llbracket \text{fcr}f \Gamma \text{ ur}; \text{osc ur} \rrbracket \implies \text{FCD } \Gamma \mathcal{V} \text{ Tr}_{(\text{induceES SES})}$

<proof>

theorem *unwinding-theorem-FCI:*

$\llbracket \text{fcr}b \Gamma \text{ ur}; \text{osc ur} \rrbracket \implies \text{FCI } \Gamma \mathcal{V} \text{ Tr}_{(\text{induceES SES})}$

<proof>

theorem *unwinding-theorem-FCIA:*

$\llbracket \text{fcr}be \Gamma \varrho \text{ ur}; \text{osc ur} \rrbracket \implies \text{FCIA } \varrho \Gamma \mathcal{V} \text{ Tr}_{(\text{induceES SES})}$

<proof>

theorem *unwinding-theorem-SD:*

$\llbracket \mathcal{V}' = \langle V = (V_{\mathcal{V}} \cup N_{\mathcal{V}}), N = \{\}, C = C_{\mathcal{V}} \rangle;$
Unwinding.lrf SES $\mathcal{V}' \text{ ur};$ *Unwinding.osc SES* $\mathcal{V}' \text{ ur} \rrbracket$

$\implies \text{SD } \mathcal{V} \text{ Tr}_{(\text{induceES SES})}$

<proof>

theorem *unwinding-theorem-SI:*

$\llbracket \mathcal{V}' = \langle V = (V_{\mathcal{V}} \cup N_{\mathcal{V}}), N = \{\}, C = C_{\mathcal{V}} \rangle;$
Unwinding.lrb SES $\mathcal{V}' \text{ ur};$ *Unwinding.osc SES* $\mathcal{V}' \text{ ur} \rrbracket$

$\implies \text{SI } \mathcal{V} \text{ Tr}_{(\text{induceES SES})}$

<proof>

theorem *unwinding-theorem-SIA*:
 $\llbracket \mathcal{V}' = \langle V = (V_{\mathcal{V}} \cup N_{\mathcal{V}}), N = \{\}, C = C_{\mathcal{V}} \rangle; \varrho \mathcal{V} = \varrho \mathcal{V}';$
Unwinding.lrbe *SES* $\mathcal{V}' \varrho ur$; *Unwinding.osc* *SES* $\mathcal{V}' ur \rrbracket$
 $\implies SIA \varrho \mathcal{V} Tr(\text{induceES } SES)$
 $\langle \text{proof} \rangle$

theorem *unwinding-theorem-SR*:
 $\llbracket \mathcal{V}' = \langle V = (V_{\mathcal{V}} \cup N_{\mathcal{V}}), N = \{\}, C = C_{\mathcal{V}} \rangle;$
Unwinding.lrf *SES* $\mathcal{V}' ur$; *Unwinding.osc* *SES* $\mathcal{V}' ur \rrbracket$
 $\implies SR \mathcal{V} Tr(\text{induceES } SES)$
 $\langle \text{proof} \rangle$

theorem *unwinding-theorem-D*:
 $\llbracket \text{lrf } ur; \text{osc } ur \rrbracket \implies D \mathcal{V} Tr(\text{induceES } SES)$
 $\langle \text{proof} \rangle$

theorem *unwinding-theorem-I*:
 $\llbracket \text{lrb } ur; \text{osc } ur \rrbracket \implies I \mathcal{V} Tr(\text{induceES } SES)$
 $\langle \text{proof} \rangle$

theorem *unwinding-theorem-IA*:
 $\llbracket \text{lrbe } \varrho ur; \text{osc } ur \rrbracket \implies IA \varrho \mathcal{V} Tr(\text{induceES } SES)$
 $\langle \text{proof} \rangle$

theorem *unwinding-theorem-R*:
 $\llbracket \text{lrf } ur; \text{osc } ur \rrbracket \implies R \mathcal{V} (Tr(\text{induceES } SES))$
 $\langle \text{proof} \rangle$

end

end

5.4 Compositionality

We prove the compositionality results from [3].

5.4.1 Auxiliary Definitions & Results

theory *CompositionBase*
imports *../Basics/BSPTaxonomy*
begin

definition
properSeparationOfViews ::
 $'e \text{ ES-rec} \Rightarrow 'e \text{ ES-rec} \Rightarrow 'e \text{ V-rec} \Rightarrow 'e \text{ V-rec} \Rightarrow 'e \text{ V-rec} \Rightarrow \text{bool}$
where
properSeparationOfViews *ES1 ES2* $\mathcal{V} \mathcal{V}1 \mathcal{V}2 \equiv$
 $V_{\mathcal{V}} \cap E_{ES1} = V_{\mathcal{V}1}$
 $\wedge V_{\mathcal{V}} \cap E_{ES2} = V_{\mathcal{V}2}$
 $\wedge C_{\mathcal{V}} \cap E_{ES1} \subseteq C_{\mathcal{V}1}$
 $\wedge C_{\mathcal{V}} \cap E_{ES2} \subseteq C_{\mathcal{V}2}$

$$\wedge N_{\mathcal{V}1} \cap N_{\mathcal{V}2} = \{\}$$

definition

wellBehavedComposition ::

'e ES-rec \Rightarrow 'e ES-rec \Rightarrow 'e V-rec \Rightarrow 'e V-rec \Rightarrow 'e V-rec \Rightarrow bool

where

wellBehavedComposition ES1 ES2 \mathcal{V} $\mathcal{V}1$ $\mathcal{V}2$ \equiv

($N_{\mathcal{V}1} \cap E_{ES2} = \{\}$ \wedge $N_{\mathcal{V}2} \cap E_{ES1} = \{\}$)
 \vee ($\exists \varrho1. (N_{\mathcal{V}1} \cap E_{ES2} = \{\} \wedge total\ ES1 (C_{\mathcal{V}1} \cap N_{\mathcal{V}2})$
 $\wedge BSIA\ \varrho1\ \mathcal{V}1\ Tr_{ES1})$)
 \vee ($\exists \varrho2. (N_{\mathcal{V}2} \cap E_{ES1} = \{\} \wedge total\ ES2 (C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$
 $\wedge BSIA\ \varrho2\ \mathcal{V}2\ Tr_{ES2})$)
 \vee ($\exists \varrho1\ \varrho2\ \Gamma1\ \Gamma2. ($
 $\nabla_{\Gamma1} \subseteq E_{ES1} \wedge \Delta_{\Gamma1} \subseteq E_{ES1} \wedge \Upsilon_{\Gamma1} \subseteq E_{ES1}$
 $\wedge \nabla_{\Gamma2} \subseteq E_{ES2} \wedge \Delta_{\Gamma2} \subseteq E_{ES2} \wedge \Upsilon_{\Gamma2} \subseteq E_{ES2}$
 $\wedge BSIA\ \varrho1\ \mathcal{V}1\ Tr_{ES1} \wedge BSIA\ \varrho2\ \mathcal{V}2\ Tr_{ES2}$
 $\wedge total\ ES1 (C_{\mathcal{V}1} \cap N_{\mathcal{V}2}) \wedge total\ ES2 (C_{\mathcal{V}2} \cap N_{\mathcal{V}1})$
 $\wedge FCIA\ \varrho1\ \Gamma1\ \mathcal{V}1\ Tr_{ES1} \wedge FCIA\ \varrho2\ \Gamma2\ \mathcal{V}2\ Tr_{ES2}$
 $\wedge V_{\mathcal{V}1} \cap V_{\mathcal{V}2} \subseteq \nabla_{\Gamma1} \cup \nabla_{\Gamma2}$
 $\wedge C_{\mathcal{V}1} \cap N_{\mathcal{V}2} \subseteq \Upsilon_{\Gamma1} \wedge C_{\mathcal{V}2} \cap N_{\mathcal{V}1} \subseteq \Upsilon_{\Gamma2}$
 $\wedge N_{\mathcal{V}1} \cap \Delta_{\Gamma1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma2} \cap E_{ES1} = \{\})$)

locale *Compositionality* =

fixes ES1 :: 'e ES-rec

and ES2 :: 'e ES-rec

and \mathcal{V} :: 'e V-rec

and $\mathcal{V}1$:: 'e V-rec

and $\mathcal{V}2$:: 'e V-rec

assumes *validES1*: ES-valid ES1

and *validES2*: ES-valid ES2

and *composableES1ES2*: composable ES1 ES2

and *validVC*: *isViewOn* \mathcal{V} ($E_{(ES1 \parallel ES2)}$)

and *validV1*: *isViewOn* $\mathcal{V}1$ E_{ES1}

and *validV2*: *isViewOn* $\mathcal{V}2$ E_{ES2}

and *propSepViews*: *properSeparationOfViews* ES1 ES2 \mathcal{V} $\mathcal{V}1$ $\mathcal{V}2$

and *well-behaved-composition*: *wellBehavedComposition* ES1 ES2 \mathcal{V} $\mathcal{V}1$ $\mathcal{V}2$

sublocale *Compositionality* \subseteq *BSPTaxonomyDifferentCorrections* ES1 \parallel ES2 \mathcal{V}

<proof>

context *Compositionality*
begin

lemma *Vv-is-Vv1-union-Vv2*: $V_{\mathcal{V}} = V_{\mathcal{V}1} \cup V_{\mathcal{V}2}$
 ⟨proof⟩

lemma *disjoint-Nv1-Vv2*: $N_{\mathcal{V}1} \cap V_{\mathcal{V}2} = \{\}$
 ⟨proof⟩

lemma *disjoint-Nv2-Vv1*: $N_{\mathcal{V}2} \cap V_{\mathcal{V}1} = \{\}$
 ⟨proof⟩

lemma *merge-property'*: $\llbracket \text{set } t1 \subseteq E_{ES1}; \text{set } t2 \subseteq E_{ES2};$
 $t1 \upharpoonright E_{ES2} = t2 \upharpoonright E_{ES1}; t1 \upharpoonright V_{\mathcal{V}} = \llbracket; t2 \upharpoonright V_{\mathcal{V}} = \llbracket;$
 $t1 \upharpoonright C_{\mathcal{V}} = \llbracket; t2 \upharpoonright C_{\mathcal{V}} = \llbracket \rrbracket$
 $\implies \exists t. (t \upharpoonright E_{ES1} = t1 \wedge t \upharpoonright E_{ES2} = t2 \wedge t \upharpoonright V_{\mathcal{V}} = \llbracket \wedge t \upharpoonright C_{\mathcal{V}} = \llbracket \wedge \text{set } t \subseteq (E_{ES1} \cup E_{ES2}))$
 ⟨proof⟩

lemma *Nv1-union-Nv2-subsetof-Nv*: $N_{\mathcal{V}1} \cup N_{\mathcal{V}2} \subseteq N_{\mathcal{V}}$
 ⟨proof⟩

end

end

theory *CompositionSupport*
imports *CompositionBase*
begin

locale *CompositionSupport* =
fixes $ESi :: 'e \text{ ES-rec}$
and $\mathcal{V} :: 'e \text{ V-rec}$
and $\mathcal{V}i :: 'e \text{ V-rec}$

assumes *validESi*: *ES-valid* ESi

and *validVi*: *isViewOn* $\mathcal{V}i \ E_{ESi}$
and *Vv-inter-Ei-is-Vvi*: $V_{\mathcal{V}} \cap E_{ESi} = V_{\mathcal{V}i}$
and *Cv-inter-Ei-subsetof-Cvi*: $C_{\mathcal{V}} \cap E_{ESi} \subseteq C_{\mathcal{V}i}$

context *CompositionSupport*
begin

lemma *BSD-in-subsystem:*

$\llbracket c \in C_{\mathcal{V}}; ((\beta @ [c] @ \alpha) \upharpoonright E_{ESi}) \in Tr_{ESi}; BSD \ \forall i \ Tr_{ESi} \rrbracket$
 $\implies \exists \alpha-i'. ((\beta \upharpoonright E_{ESi}) @ \alpha-i') \in Tr_{ESi}$
 $\wedge (\alpha-i' \upharpoonright V_{\mathcal{V}i}) = (\alpha \upharpoonright V_{\mathcal{V}i}) \wedge \alpha-i' \upharpoonright C_{\mathcal{V}i} = []$
<proof>

lemma *BSD-in-subsystem2:*

$\llbracket ((\beta @ \alpha) \upharpoonright E_{ESi}) \in Tr_{ESi}; BSD \ \forall i \ Tr_{ESi} \rrbracket$
 $\implies \exists \alpha-i'. ((\beta \upharpoonright E_{ESi}) @ \alpha-i') \in Tr_{ESi} \wedge (\alpha-i' \upharpoonright V_{\mathcal{V}i}) = (\alpha \upharpoonright V_{\mathcal{V}i}) \wedge \alpha-i' \upharpoonright C_{\mathcal{V}i} = []$
<proof>

end

end

5.4.2 Generalized Zipping Lemma

theory *GeneralizedZippingLemma*

imports *CompositionBase*

begin

context *Compositionality*

begin

lemma *generalized-zipping-lemma1:* $\llbracket N_{\mathcal{V}1} \cap E_{ES2} = \{\}; N_{\mathcal{V}2} \cap E_{ES1} = \{\} \rrbracket \implies$
 $\forall \tau \ \text{lambda } t1 \ t2. ((\text{set } \tau \subseteq E_{(ES1 \parallel ES2)} \wedge \text{set } \text{lambda} \subseteq V_{\mathcal{V}} \wedge \text{set } t1 \subseteq E_{ES1} \wedge \text{set } t2 \subseteq E_{ES2}$
 $\wedge ((\tau \upharpoonright E_{ES1}) @ t1) \in Tr_{ES1} \wedge ((\tau \upharpoonright E_{ES2}) @ t2) \in Tr_{ES2} \wedge (\text{lambda} \upharpoonright E_{ES1}) = (t1 \upharpoonright V_{\mathcal{V}})$
 $\wedge (\text{lambda} \upharpoonright E_{ES2}) = (t2 \upharpoonright V_{\mathcal{V}}) \wedge (t1 \upharpoonright C_{\mathcal{V}1}) = [] \wedge (t2 \upharpoonright C_{\mathcal{V}2}) = [])$
 $\longrightarrow (\exists t. ((\tau @ t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \upharpoonright V_{\mathcal{V}}) = \text{lambda} \wedge (t \upharpoonright C_{\mathcal{V}}) = [])))$
<proof>

lemma *generalized-zipping-lemma2:* $\llbracket N_{\mathcal{V}1} \cap E_{ES2} = \{\}; \text{total } ES1 \ (C_{\mathcal{V}1} \cap N_{\mathcal{V}2}); \text{BSIA } \rho1 \ \forall1 \ Tr_{ES1} \rrbracket$
 \implies

$\forall \tau \ \text{lambda } t1 \ t2. ((\text{set } \tau \subseteq (E_{(ES1 \parallel ES2)}) \wedge \text{set } \text{lambda} \subseteq V_{\mathcal{V}} \wedge \text{set } t1 \subseteq E_{ES1} \wedge \text{set } t2 \subseteq E_{ES2}$
 $\wedge ((\tau \upharpoonright E_{ES1}) @ t1) \in Tr_{ES1} \wedge ((\tau \upharpoonright E_{ES2}) @ t2) \in Tr_{ES2}$
 $\wedge (\text{lambda} \upharpoonright E_{ES1}) = (t1 \upharpoonright V_{\mathcal{V}}) \wedge (\text{lambda} \upharpoonright E_{ES2}) = (t2 \upharpoonright V_{\mathcal{V}})$
 $\wedge (t1 \upharpoonright C_{\mathcal{V}1}) = [] \wedge (t2 \upharpoonright C_{\mathcal{V}2}) = [])$
 $\longrightarrow (\exists t. ((\tau @ t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \upharpoonright V_{\mathcal{V}}) = \text{lambda} \wedge (t \upharpoonright C_{\mathcal{V}}) = [])))$
<proof>

lemma *generalized-zipping-lemma3:* $\llbracket N_{\mathcal{V}2} \cap E_{ES1} = \{\}; \text{total } ES2 \ (C_{\mathcal{V}2} \cap N_{\mathcal{V}1}); \text{BSIA } \rho2 \ \forall2 \ Tr_{ES2} \rrbracket$
 \implies

$\forall \tau \ \text{lambda } t1 \ t2. ((\text{set } \tau \subseteq E_{(ES1 \parallel ES2)} \wedge \text{set } \text{lambda} \subseteq V_{\mathcal{V}} \wedge \text{set } t1 \subseteq E_{ES1} \wedge \text{set } t2 \subseteq E_{ES2}$
 $\wedge ((\tau \upharpoonright E_{ES1}) @ t1) \in Tr_{ES1} \wedge ((\tau \upharpoonright E_{ES2}) @ t2) \in Tr_{ES2}$

$\wedge (\text{lambda} \upharpoonright E_{ES1}) = (t1 \upharpoonright V_{\mathcal{V}}) \wedge (\text{lambda} \upharpoonright E_{ES2}) = (t2 \upharpoonright V_{\mathcal{V}})$
 $\wedge (t1 \upharpoonright C_{\mathcal{V}1}) = [] \wedge (t2 \upharpoonright C_{\mathcal{V}2}) = []$
 $\longrightarrow (\exists t. ((\tau @ t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \upharpoonright V_{\mathcal{V}}) = \text{lambda} \wedge (t \upharpoonright C_{\mathcal{V}}) = []))$
 <proof>

lemma *generalized-zipping-lemma4*:

$\llbracket \nabla_{\Gamma1} \subseteq E_{ES1}; \Delta_{\Gamma1} \subseteq E_{ES1}; \Upsilon_{\Gamma1} \subseteq E_{ES1}; \nabla_{\Gamma2} \subseteq E_{ES2}; \Delta_{\Gamma2} \subseteq E_{ES2}; \Upsilon_{\Gamma2} \subseteq E_{ES2};$
 $BSIA \ \varrho1 \ \mathcal{V}1 \ Tr_{ES1}; BSIA \ \varrho2 \ \mathcal{V}2 \ Tr_{ES2}; \text{total } ES1 \ (C_{\mathcal{V}1} \cap N_{\mathcal{V}2}); \text{total } ES2 \ (C_{\mathcal{V}2} \cap N_{\mathcal{V}1});$
 $FCIA \ \varrho1 \ \Gamma1 \ \mathcal{V}1 \ Tr_{ES1}; FCIA \ \varrho2 \ \Gamma2 \ \mathcal{V}2 \ Tr_{ES2}; V_{\mathcal{V}1} \cap V_{\mathcal{V}2} \subseteq \nabla_{\Gamma1} \cup \nabla_{\Gamma2};$
 $C_{\mathcal{V}1} \cap N_{\mathcal{V}2} \subseteq \Upsilon_{\Gamma1}; C_{\mathcal{V}2} \cap N_{\mathcal{V}1} \subseteq \Upsilon_{\Gamma2};$
 $N_{\mathcal{V}1} \cap \Delta_{\Gamma1} \cap E_{ES2} = \{\}; N_{\mathcal{V}2} \cap \Delta_{\Gamma2} \cap E_{ES1} = \{\} \rrbracket \implies$
 $\forall \tau \text{ lambda } t1 \ t2. ((\text{set } \tau \subseteq (E_{(ES1 \parallel ES2)}) \wedge \text{set } \text{lambda} \subseteq V_{\mathcal{V}} \wedge \text{set } t1 \subseteq E_{ES1}$
 $\wedge \text{set } t2 \subseteq E_{ES2} \wedge ((\tau \upharpoonright E_{ES1}) @ t1) \in Tr_{ES1} \wedge ((\tau \upharpoonright E_{ES2}) @ t2) \in Tr_{ES2}$
 $\wedge (\text{lambda} \upharpoonright E_{ES1}) = (t1 \upharpoonright V_{\mathcal{V}}) \wedge (\text{lambda} \upharpoonright E_{ES2}) = (t2 \upharpoonright V_{\mathcal{V}})$
 $\wedge (t1 \upharpoonright C_{\mathcal{V}1}) = [] \wedge (t2 \upharpoonright C_{\mathcal{V}2}) = []$
 $\longrightarrow (\exists t. ((\tau @ t) \in (Tr_{(ES1 \parallel ES2)}) \wedge (t \upharpoonright V_{\mathcal{V}}) = \text{lambda} \wedge (t \upharpoonright C_{\mathcal{V}}) = [])))$
 <proof>

lemma *generalized-zipping-lemma*:

$\forall \tau \text{ lambda } t1 \ t2. ((\text{set } \tau \subseteq E_{(ES1 \parallel ES2)}$
 $\wedge \text{set } \text{lambda} \subseteq V_{\mathcal{V}} \wedge \text{set } t1 \subseteq E_{ES1} \wedge \text{set } t2 \subseteq E_{ES2}$
 $\wedge ((\tau \upharpoonright E_{ES1}) @ t1) \in Tr_{ES1} \wedge ((\tau \upharpoonright E_{ES2}) @ t2) \in Tr_{ES2}$
 $\wedge (\text{lambda} \upharpoonright E_{ES1}) = (t1 \upharpoonright V_{\mathcal{V}}) \wedge (\text{lambda} \upharpoonright E_{ES2}) = (t2 \upharpoonright V_{\mathcal{V}})$
 $\wedge (t1 \upharpoonright C_{\mathcal{V}1}) = [] \wedge (t2 \upharpoonright C_{\mathcal{V}2}) = []$
 $\longrightarrow (\exists t. ((\tau @ t) \in Tr_{(ES1 \parallel ES2)} \wedge (t \upharpoonright V_{\mathcal{V}}) = \text{lambda} \wedge (t \upharpoonright C_{\mathcal{V}}) = [])))$
 <proof>

end

end

5.4.3 Compositionality Results

theory *CompositionalityResults*

imports *GeneralizedZippingLemma CompositionSupport*

begin

context *Compositionality*

begin

theorem *compositionality-BSD*:

$\llbracket BSD \ \mathcal{V}1 \ Tr_{ES1}; BSD \ \mathcal{V}2 \ Tr_{ES2} \rrbracket \implies BSD \ \mathcal{V} \ Tr_{(ES1 \parallel ES2)}$

<proof>

theorem *compositionality-BSI*:

$\llbracket BSD \ \mathcal{V}1 \ Tr_{ES1}; BSD \ \mathcal{V}2 \ Tr_{ES2}; BSI \ \mathcal{V}1 \ Tr_{ES1}; BSI \ \mathcal{V}2 \ Tr_{ES2} \rrbracket$

$\implies BSI \vee Tr_{(ES1 \parallel ES2)}$
 ⟨proof⟩

theorem compositionality-BSIA:

$\llbracket BSD \vee 1 Tr_{ES1}; BSD \vee 2 Tr_{ES2}; BSIA \varrho 1 \vee 1 Tr_{ES1}; BSIA \varrho 2 \vee 2 Tr_{ES2};$
 $(\varrho 1 \vee 1) \subseteq (\varrho \vee) \cap E_{ES1}; (\varrho 2 \vee 2) \subseteq (\varrho \vee) \cap E_{ES2} \rrbracket$
 $\implies BSIA \varrho \vee (Tr_{(ES1 \parallel ES2)})$
 ⟨proof⟩

theorem compositionality-FCD:

$\llbracket BSD \vee 1 Tr_{ES1}; BSD \vee 2 Tr_{ES2};$
 $\nabla_{\Gamma} \cap E_{ES1} \subseteq \nabla_{\Gamma 1}; \nabla_{\Gamma} \cap E_{ES2} \subseteq \nabla_{\Gamma 2};$
 $\Upsilon_{\Gamma} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}; \Upsilon_{\Gamma} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2};$
 $(\Delta_{\Gamma 1} \cap N_{\vee 1} \cup \Delta_{\Gamma 2} \cap N_{\vee 2}) \subseteq \Delta_{\Gamma};$
 $N_{\vee 1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\}; N_{\vee 2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\};$
 $FCD \Gamma 1 \vee 1 Tr_{ES1}; FCD \Gamma 2 \vee 2 Tr_{ES2} \rrbracket$
 $\implies FCD \Gamma \vee (Tr_{(ES1 \parallel ES2)})$
 ⟨proof⟩

theorem compositionality-FCI:

$\llbracket BSD \vee 1 Tr_{ES1}; BSD \vee 2 Tr_{ES2}; BSIA \varrho 1 \vee 1 Tr_{ES1}; BSIA \varrho 2 \vee 2 Tr_{ES2};$
 $total ES1 (C_{\vee 1} \cap \Upsilon_{\Gamma 1}); total ES2 (C_{\vee 2} \cap \Upsilon_{\Gamma 2});$
 $\nabla_{\Gamma} \cap E_{ES1} \subseteq \nabla_{\Gamma 1}; \nabla_{\Gamma} \cap E_{ES2} \subseteq \nabla_{\Gamma 2};$
 $\Upsilon_{\Gamma} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}; \Upsilon_{\Gamma} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2};$
 $(\Delta_{\Gamma 1} \cap N_{\vee 1} \cup \Delta_{\Gamma 2} \cap N_{\vee 2}) \subseteq \Delta_{\Gamma};$
 $(N_{\vee 1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\vee 2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1})$
 $\vee (N_{\vee 2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\} \wedge N_{\vee 1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}) ;$
 $FCI \Gamma 1 \vee 1 Tr_{ES1}; FCI \Gamma 2 \vee 2 Tr_{ES2} \rrbracket$
 $\implies FCI \Gamma \vee (Tr_{(ES1 \parallel ES2)})$
 ⟨proof⟩

theorem compositionality-FCIA:

$\llbracket BSD \vee 1 Tr_{ES1}; BSD \vee 2 Tr_{ES2}; BSIA \varrho 1 \vee 1 Tr_{ES1}; BSIA \varrho 2 \vee 2 Tr_{ES2};$
 $(\varrho 1 \vee 1) \subseteq (\varrho \vee) \cap E_{ES1}; (\varrho 2 \vee 2) \subseteq (\varrho \vee) \cap E_{ES2};$
 $total ES1 (C_{\vee 1} \cap \Upsilon_{\Gamma 1} \cap N_{\vee 2} \cap \Delta_{\Gamma 2}); total ES2 (C_{\vee 2} \cap \Upsilon_{\Gamma 2} \cap N_{\vee 1} \cap \Delta_{\Gamma 1});$
 $\nabla_{\Gamma} \cap E_{ES1} \subseteq \nabla_{\Gamma 1}; \nabla_{\Gamma} \cap E_{ES2} \subseteq \nabla_{\Gamma 2};$
 $\Upsilon_{\Gamma} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1}; \Upsilon_{\Gamma} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2};$
 $(\Delta_{\Gamma 1} \cap N_{\vee 1} \cup \Delta_{\Gamma 2} \cap N_{\vee 2}) \subseteq \Delta_{\Gamma};$
 $(N_{\vee 1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\vee 2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1})$
 $\vee (N_{\vee 2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\} \wedge N_{\vee 1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2}) ;$
 $FCIA \varrho 1 \Gamma 1 \vee 1 Tr_{ES1}; FCIA \varrho 2 \Gamma 2 \vee 2 Tr_{ES2} \rrbracket$
 $\implies FCIA \varrho \Gamma \vee (Tr_{(ES1 \parallel ES2)})$
 ⟨proof⟩

theorem compositionality-R:

$\llbracket R \vee 1 \text{ Tr}_{ES1}; R \vee 2 \text{ Tr}_{ES2} \rrbracket \Longrightarrow R \vee (\text{Tr}_{(ES1 \parallel ES2)})$
 ⟨proof⟩

end

locale *CompositionalityStrictBSPs* = *Compositionality* +

assumes *NV-inter-E1-is-NV1*: $N_{\mathcal{V}} \cap E_{ES1} = N_{\mathcal{V}1}$
and *NV-inter-E2-is-NV2*: $N_{\mathcal{V}} \cap E_{ES2} = N_{\mathcal{V}2}$

sublocale *CompositionalityStrictBSPs* \subseteq *Compositionality*
 ⟨proof⟩

context *CompositionalityStrictBSPs*
begin

theorem *compositionality-SR*:
 $\llbracket SR \vee 1 \text{ Tr}_{ES1}; SR \vee 2 \text{ Tr}_{ES2} \rrbracket \Longrightarrow SR \vee (\text{Tr}_{(ES1 \parallel ES2)})$
 ⟨proof⟩

theorem *compositionality-SD*:
 $\llbracket SD \vee 1 \text{ Tr}_{ES1}; SD \vee 2 \text{ Tr}_{ES2} \rrbracket \Longrightarrow SD \vee (\text{Tr}_{(ES1 \parallel ES2)})$
 ⟨proof⟩

theorem *compositionality-SI*:
 $\llbracket SD \vee 1 \text{ Tr}_{ES1}; SD \vee 2 \text{ Tr}_{ES2}; SI \vee 1 \text{ Tr}_{ES1}; SI \vee 2 \text{ Tr}_{ES2} \rrbracket$
 $\Longrightarrow SI \vee (\text{Tr}_{(ES1 \parallel ES2)})$
 ⟨proof⟩

theorem *compositionality-SIA*:
 $\llbracket SD \vee 1 \text{ Tr}_{ES1}; SD \vee 2 \text{ Tr}_{ES2}; SIA \varrho 1 \vee 1 \text{ Tr}_{ES1}; SIA \varrho 2 \vee 2 \text{ Tr}_{ES2};$
 $(\varrho 1 \vee 1) \subseteq (\varrho \vee) \cap E_{ES1}; (\varrho 2 \vee 2) \subseteq (\varrho \vee) \cap E_{ES2} \rrbracket$
 $\Longrightarrow SIA \varrho \vee (\text{Tr}_{(ES1 \parallel ES2)})$
 ⟨proof⟩

end

end

Acknowledgments

This work was partially funded by the DFG (German Research Foundation) under the projects FM-SecEng (MA 3326/1-2, MA 3326/1-3) and RSCP (MA 3326/4-3).

References

- [1] S. Grewe, H. Mantel, M. Tasch, R. Gay, and H. Sudbrock. I-MAKS – A Framework for Information-Flow Security in Isabelle/HOL. Technical Report TUD-CS-2018-0056, TU Darmstadt, 2018.
- [2] H. Mantel. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 185–199, 2000.
- [3] H. Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Saarland University, Saarbrücken, Germany, 2003.