I-MAKS

A Framework for Information-Flow Security in Isabelle/HOL

Technical Report TUD-CS-2018-0056
April, 2018

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

TECHNISCHE UNIVERSITÄT DARMSTADT

Modeling and Analysis
of Information Systems,
Software Technology Group

I-MAKS

A Framework for Information-Flow Security in Isabelle/HOL

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

Department of Computer Science, TU Darmstadt, Germany
{gay, mantel, sudbrock, tasch}@mais.informatik.tu-darmstadt.de
grewe@st.informatik.tu-darmstadt.de

Abstract The "Modular Assembly Kit for Security Properties" (*MAKS*) is a framework for both the specification and verification of possibilistic information-flow security properties at the specification-level. With *I-MAKS* we provide an Isabelle/HOL formalization of the framework. *I-MAKS* is a machine-checked formalization of *MAKS* benefiting from the rigor of the proof assistant Isabelle/HOL. *I-MAKS* enables the usage of *MAKS* together with the general purpose verification techniques provided by Isabelle/HOL. Moreover, the machine-checked proofs of *I-MAKS* increase the confidence in the existing pen-and-paper proofs for *MAKS* or extensions of *MAKS*. In this report, we give an overview of *I-MAKS* describing how the pen-and-paper formalization of *MAKS* is captured in Isabelle/HOL.

1 Introduction

Information-flow control ensures that a program does not leak secrets when running. Information-flow control is complementary to access control: While access control shall prevent that illegitimate accesses occur, information-flow control shall prevent that secrets are leaked after a legitimate access.

Noninterference [GM82] is probably the best known property that captures information-flow security formally. Intuitively, noninterference is defined as the requirement that the public output in each system run is identical to what the system would output if it were supplied with the same public input but no secret input. This requirement ensures that a system's output to public sinks is completely independent from secret input that is given to the system.

While noninterference is often suitable for capturing confidentiality requirements, it has turned out to be too restrictive when a system specification is nondeterministic. This observation motivated a search for alternative definitions of information-flow security. Starting with nondeducibility [Sut86], generalized noninterference [McC87] and restrictiveness [McC87], numerous, so called, possibilistic information-flow properties were proposed. Each of these properties defines security by a closure condition on the set of possible system runs. Since there seems not to be a unique closure condition that is best for all purposes, various possibilistic information-flow properties co-exist [Man11] and frameworks were developed to enable a uniform treatment of possibilistic information-flow properties (e.g. [McL94, FG95, ZL95, PWK96, Man00a]).

In this report, we present *I-MAKS*, a framework for the formal specification and verification of information-flow properties in Isabelle/HOL. *I-MAKS* is a machine-checked formalization of the Modular Assembly Kit for Security Properties (*MAKS*) [Man00a, Man03] in its version from [Man03] in Isabelle/HOL [NPW02].[1] With *I-MAKS* we enable the usage of *MAKS* and its meta-results for the verification of information-flow properties together with the specification and verification techniques provided in Isabelle/HOL. Moreover, we facilitate the adoption and extension of *MAKS* by providing a machine-checked specification in Isabelle/HOL.

In *I-MAKS*, we re-verified the soundness of the unwinding technique and the techniques for compositional reasoning provided in *MAKS* in Isabelle/HOL. Having machine-checked proofs increases confidence in these techniques. So far only pen-and-paper proofs for these results existed [Man00b,Man02,Man03]. Surprisingly, we detected only a single technical mistake, namely, one incorrect step in the proof of the *Generalized Zipping Lemma*. The original statement of the lemma was correct. We corrected this step and constructed a machine-checked proof of this lemma.

Structure. This report is structured as follows: In Section 2, we recall preliminaries about event-based system models, possibilistic information-flow security, and Isabelle/HOL. In Section 3, we give an overview of *I-MAKS*' high-level structure and introduce some basic definitions. In Sections 4 and 5, we describe the system models and the framework for the specification of security requirements provided by *I-MAKS*. In Section 6, we present the meta-results for the verification of security requirements provided by *I-MAKS*. We discuss related work in Section 7 and conclude in Section 8.

2 Preliminaries

2.1 Event-Based System Models

Events and Traces. In this report, we use events to model actions of a system and we use traces to model (partial) system runs.

Definition 1. *An* event *is a term that models an atomic action.* $\diamond$

We use events to model the actions in which a given system can engage and that can be seen atomic on the considered level of abstraction. For instance, the term $send(msg)$ could be used to model that the message msg is sent. Similarly, the term $recv(msg)$ could be used to model that the message msg is received.

Definition 2. *Let E be a set of events. A* trace over E *is a finite list of events from the set E.* $\diamond$

We use traces to model possible (partial) system runs. The events in a trace model which actions occur in the system run modeled by the trace. The order of events in the trace reflects the order in which these actions occur.

[1] The Isabelle/HOL theories of *I-MAKS* can be found under `http://www.mais.informatik.tu-darmstadt.de/assets/tools/I-MAKS2018.tar.gz`.

As convention, we refer to a trace over E just as trace if the set of events E is clear from the context. As notational convention, we denote the empty trace by $\langle\rangle$. We denote the trace that starts with an occurrence of an event e and then continues as the trace τ by $e.\tau$. We denote the trace consisting of the events e_1, e_2, $\ldots$, and e_k in this order by $\langle e_1.e_2.\cdots.e_k\rangle$. Finally, we denote the set of all traces over a set of events E by E^*.

Definition 3. *Let E be a set of events. The* concatenation of two traces *traces $\alpha \in E^*$ and $\beta \in E^*$ (denoted by $\alpha.\beta$) is recursively defined by:*

$$\alpha.\beta = \begin{cases} \beta & \text{if } \alpha = \langle\rangle \\ e.(\alpha'.\beta) & \text{if } \alpha = e.\alpha' \end{cases}$$

$\diamond$

For instance, $\langle e_1.e_2\rangle.\langle e_2.e_3\rangle$ equals the trace $\langle e_1.e_2.e_2.e_3\rangle$.

Definition 4. *Let E be a set of events. A trace $\alpha \in E^*$ is a* prefix *of a trace $\tau \in E^*$ if there exists a trace $\beta \in E^*$ such that $\tau = \alpha.\beta$. A set of traces $Tr \subseteq E^*$ is* closed under prefixes *if each prefix α of each trace $\tau \in Tr$ is also contained in Tr, i.e. $\alpha \in Tr$.* $\diamond$

For instance, $\langle\rangle$, $\langle e_1\rangle$, $\langle e_1.e_2\rangle$, and $\langle e_1.e_2.e_3\rangle$ are prefixes of the trace $\langle e_1.e_2.e_3\rangle$ and the set $\{\langle\rangle, \langle e_1\rangle, \langle e_1.e_2\rangle, \langle e_1.e_2.e_3\rangle\}$ is prefix closed.

Definition 5. *Let E and E' be sets of events. Let $\tau \in E^*$ be a trace. The* projection of τ to E' *(denoted by $\tau{\restriction}E'$) is recursively defined by:*

$$\tau{\restriction}E' = \begin{cases} \langle\rangle & \text{if } \tau = \langle\rangle \\ e.(\tau'{\restriction}E') & \text{if } \tau = e.\tau' \wedge e \in E' \\ \tau'{\restriction}E' & \text{if } \tau = e.\tau' \wedge e \notin E' \end{cases}$$

$\diamond$

The event sequence $\tau{\restriction}E'$ contains those occurrences of events in τ that are contained in the set E'. Note that the relative order of events in $\tau{\restriction}E'$ is the same as in τ. For instance, $\langle e_1.e_2.e_2.e_3\rangle{\restriction}\{e_1, e_2\}$ equals the trace $\langle e_1.e_2.e_2\rangle$.

Processes. We use processes to model the actions and the behavior of systems.

Definition 6. *A* process *is a pair (E, Tr) where E is a set of events and Tr is a set of traces over E(i.e. $Tr \subseteq E^*$) that is closed under prefixes.* $\diamond$

With the set of events E we model in which actions the system can engage, and with the set of traces Tr we model which system runs are possible. Due to prefix closure, we do not only model complete system runs by Tr but also partial ones.

As convention, we refer to a trace $\tau \in Tr$ as *possible trace of Tr* for a given set of traces Tr modeling the behavior of a system. If the set of traces Tr is clear from the context, we leave it out.

Labeled Transition Systems. As a stateful alternative to processes, we use labeled transition systems to model the behavior of systems [Har87].

Definition 7. *A state is a term that models a snapshot of a system.* ◇

We use states to model a snapshot of a system during the system's execution. For instance, the term $rdyToRecv$ could be used to model that a system is currently able to receive a message. Similarly, the term $MsgProcessing$ could be used to model that a system is currently processing a message.

We use labeled transitions to model the change of a system's snapshot after the system engaged in a specific action.

Definition 8. *Let S be a set of states. Let E be a set of events. A* labeled transition *from a state s_1 to a state s_2 using the event e is a triple (s_1, e, s_2). A* labeled transition relation *between S using E is a set of labeled transitions from a state $s_1 \in S$ to a state $s_2 \in S$ using a event $e \in E$.* ◇

For instance, the labeled transition $(rdyToRecv, recv(msg), MsgProcessing)$ could be used to model that when a system is ready to receive a message, receiving a message msg changes the system's snapshot to processing the message.

As convention, we refer to a labeled transition relation between S using E as labeled transition relation if the set of states S and the set of events E are clear from the context.

Definition 9. *A labeled transition system is a tuple (S, s_0, E, T) where S is a set of states, s_0 is a state, E is a set of events, and T is a labeled transition relation.* ◇

The set of states S models the possible intermediate snapshots of the system during execution. The initial state s_0 models the snapshot of the system in which the execution starts. Likewise to processes, the set of events E models in which action the system can engage. Finally, the labeled transition relation T models how the snapshot of the system changes after engaging in an action.

2.2 Possibilistic Information-Flow Security

We consider a scenario where the interface to the system of concern is protected by appropriate access control. Hence, an attacker in this scenario cannot observe secret actions directly. In addition, we make the worst case assumption that the attacker knows how the system operates in principle. Based on his observation during a system run and his knowledge about the system, the attacker tries to infer additional information about secret actions. Intuitively, we say that a system has secure information flow if such an attacker cannot obtain information about secrets either directly by his observations or by his inference.

Formally, we model what actions an attacker can observe for a process (E, Tr) by the set $L \subseteq E$. The set L must contain all events modeling actions that are observable to the attacker. Consequently, the trace $\tau \upharpoonright L$ models the observation an attacker makes for a trace τ. We refer to a trace ν as a possible observation if there exists a trace $\tau \in \mathit{Tr}$ such that $\nu = \tau \upharpoonright L$ holds.

Complementarily to L, we model what actions an attacker cannot observe by the set $H \subseteq E$. The set H must contain all events modeling actions that are not observable to the attacker. Based on our assumption of appropriate access control, we assume that all secret actions are modeled by events in H. We also assume that L and H form a disjoint partition of E, i.e. $L \cup H = E$ and $L \cap H = \emptyset$.

Our model captures what an attacker can infer from a possible observation ν by the set $\{\tau \in Tr \mid \tau \upharpoonright L = \nu\}$. That is, the set of all possible traces that potentially generated the observation ν.

In our model, a process is considered secure, iff the attacker is unable to exclude the possibility of certain secret behavior. Complementarily, a process is considered insecure, iff the attacker is able to to do so. What secret behavior must be possible from an attacker's perspective in order for a process to be considered secure is defined by information-flow properties.

In the following we illustrate the spectrum of concrete information-flow properties by two well-known possibilistic information-flow properties before this class of security properties is explained more generally.

Definition 10. *Let (E, Tr) be a process and $L \subseteq E$ be the set of all events that model actions whose occurrences are observable to the attacker. The property* noninference *[O'H90, McL94, ZL97] is defined by:*

$$NF \equiv \forall \tau \in Tr.\ \tau \upharpoonright L \in Tr \qquad \diamond$$

Note that noninference classifies a process as secure if each possible observation $\tau \upharpoonright L$ that is generated by this process, is also a possible trace of this process. This means, no matter what observation the attacker makes during a trace, the attacker cannot infer that events in H must have occurred in this trace.

Definition 11. *Let (E, Tr) be a process, $L \subseteq E$ be the set of events that model actions whose occurrences are observable to the attacker, and $H = E \setminus L$ be the set of events that model actions whose occurrence are not observable to the attacker. The property* separability *[McL94] is defined by:*

$$SEP \equiv \forall \tau_L, \tau_H \in Tr.\ \forall t \in E^*.\ (t \upharpoonright L = \tau_L \upharpoonright L \ \wedge\ t \upharpoonright H = \tau_H \upharpoonright H) \Rightarrow t \in Tr \qquad \diamond$$

Note that separability classifies a process as secure if each possible observation $\tau_L \upharpoonright L$ that the process can generate is co-possible with the projection $\tau_H \upharpoonright H$ of each possible trace τ_H. Moreover, it is not enough if one interleaving of $\tau_L \upharpoonright L$ with $\tau_H \upharpoonright H$ is a possible trace of the process, but rather each such interleaving must be a possible trace of the process. This means that, no matter what observation the attacker makes during a trace, the attacker cannot infer that any possible projection to H of a possible trace must have or cannot have occurred.

Note that separability is more restrictive than noninference.

Theorem 1. *If a process (E, Tr) satisfies SEP then it satisfies NF.*

The additional restrictiveness is caused by requiring that not only the possible observation is a possible trace, but also any interleaving of any possible secret behavior with the possible observation is a possible trace.

Both noninference and separability require that, for each possible observation of an attacker, traces must be possible during which the observation is possible and that bare certain properties. The requirement that for each possible observation there must be certain possible traces during which the observation is possible constitutes a closure property on the set of possible traces.

Definition 12. *A property of sets of traces $P : \mathcal{P}(E^*) \to \mathbb{B}$ is a* closure property on sets of traces *if and only if for all $Tr \subseteq E^*$ there exists a set of traces such that $\overline{Tr} \supseteq Tr$ and $P(\overline{Tr})$.* $\Diamond$

Such closure properties on sets of traces are used to formally specify information-flow properties for trace-based system models. Prominent examples besides noninference and separability are, for instance, *nondeducibility* [Sut86], *generalized noninterference* [McC87], *restrictiveness* [McC87], *forward correctability* [JT88], and *perfect security property* [ZL97].

The relation between such properties is usually not as obvious as for noninference and separability (cf. Theorem 1) because the differences and similarities between the definitions of different possibilistic information-flow properties are often rather subtle. In order to simplify the comparison of properties and to achieve uniformity, several frameworks for possibilistic information-flow security were developed (e.g., [McL94, FG95, ZL95, PWK96, Man00a, Man03, BFPR03, MC12, KLP14]).

For I-MAKS, we chose *MAKS* [Man00a] in its version from [Man03] as the conceptual basis. *MAKS* supports the uniform representation of a wide range of possibilistic information-flow properties [Man00a]. It also supports the verification of such properties, using unwinding [Man00b], compositional reasoning [Man02], and model checking [DHRS11].

As part of our presentation of *I-MAKS*, we provide more detailed explanations of *MAKS* in later sections.

2.3 The Proof Assistant Isabelle/HOL

Isabelle/HOL is a specialization of the generic proof assistant Isabelle to typed higher-order logic (HOL). It supports one specification language and two verification languages. The specification language combines aspects of the functional programming language ML with typed HOL: It offers constructs for defining types, constants, functions, and formulas. The verification languages provide proof commands for creating machine-checked proofs of propositions.

Isabelle/HOL supports the structuring of type definitions, functions, theorems and proofs scripts into multiple theory files.[2] Theories may import other theories, i.e. structuring of theories is done hierarchically.

Types. Isabelle/HOL supports several base types and type constructors by default.

Throughout this report, we use the base type `bool` for boolean values and the base type `nat` for natural numbers. The base type `bool` consists of the constants

[2] For the remainder of this report *theory* is used as a shorthand for theory file.

True and False. The base type nat of the constant 0 and the constructor Suc, i.e. Suc 0 represents 1, Suc Suc 0 represents 2 and so on. Note that Isabelle/HOL permits to abbreviate applications of Suc with the corresponding number, e.g. we can write 2 instead of Suc Suc 0.

Moreover, we use the type constructors list (for lists), set (for sets), and option (for an option type). For instance, nat list is the type for lists of natural numbers, nat set is the type for sets of natural numbers, and nat option is the type for natural numbers and a special undefined value.

In addition to the type constructors introduced above, Isabelle/HOL supports the definition of function types and product types. A function type t1 ⇒ ... ⇒ tn ⇒ t declares the type of a total function from t1, ..., tn to t. Note that the declaration of function types is right-associative, i.e. t1 ⇒ t2 ⇒ t stands for t1 ⇒(t2 ⇒ t). A product type t1 × t2 declares the type of pairs. The first element of a pair can be retrieved using the selector fst, and the second element of a pair can be retrieved using the selector snd. For instance, fst (1,2) is 1.

Finally, Isabelle/HOL supports the declaration of polymorphic types using type variables. For example, 'a list declares the type of lists of arbitrary type 'a.

Terms. Terms in Isabelle/HOL are either basic constants or function applications.

Isabelle/HOL supports some basic functional programming constructs to construct terms. For example, if **if** b **then** t **else** t2, case **case** e **of** c1 ⇒ t1 | ... | cn ⇒ tn, and let **let** x=e **in** u with their usual semantics.

Furthermore, Isabelle/HOL supports λ-abstractions. For instance, λ x. x+x is the function that takes the argument x and returns x+x.

Formulas. Formulas are terms of type bool composed out of the basic constants True and False, the usual logical connectives (in decreasing priority) ¬, ∧, ∨ as well as ⟶, and the quantifiers ∃ as well as ∀.

Sets. Sets in Isabelle/HOL can be defined by explicitly listing all elements, e.g. {1, 2 ,3} is the set containing 1, 2, and 3, or by set comprehension. For example the set of all successors of natural numbers satisfying a predicate P can be defined by set comprehension as follows:

```
definition nat_P :: "nat set"
where nat_P ≡ { n. P n}
```

Sets support the usual set membership relation ∈ and its negation ∉, the subset or equal relation ⊆ as well as the set operations union ∪, intersection ∩, set difference -, and the union of all sets in a set of sets $\bigcup$.

Type Definitions. New types in Isabelle/HOL can be defined using the type constructors introduced beforehand using the keyword **type_synonym**. For instance, **type_synonym** foo = nat × bool defines the type foo as pairs of nat and bool.

Furthermore, Isabelle/HOL supports the definition of new types by the specification of a list of constructors using the keyword **datatype**. Each constructor has a finite, possible empty list of arguments. The type of an argument can either be a concrete type or a type variable. For instance, the type of lists [UT18b] that can be used as type constructor is defined by:

```
datatype 'a list = Nil | Cons 'a "'a list"
```

The term `Nil`, also denoted by `[]`, models the empty list, while a term `Cons x xs`, also denoted by `x # xs`, models a non-empty list with head element `x` and rest `xs`.

Finally, Isabelle/HOL supports the definition of an n-ary product type, called record type, whose fields are named using the keyword **record**. For instance, the type `rec` with three fields `A`, `B`, and `C` of type `nat` can be defined by:

```
record rec = A::nat B::nat C::nat
```

A term of a record type can be defined by a list of equations of the form `field name = value` where comma is used as separator between list elements and the symbols ⦇ and ⦈ are used to mark the start and the end of the list, respectively. For instance, `⦇ A = 3, B = 5, C = 7 ⦈` is a term of type `rec` from which the value of the second field can be retrieved using the field name of the second field, i.e. `B ⦇ A = 3, B = 5, C = 7 ⦈` is `5`.

Function Definitions. Functions in Isabelle/HOL can be defined using the keywords **definition**, **primrec**, **fun**, and **function**.

The keyword **definition** is used to define *non-recursive* functions. Definitions of non-recursive functions consist of a name (optionally followed by a type signature) and of exactly one *equation*. The left-hand side of the equation is the name of the definition plus the list of formal parameters. The right-hand side of the equation is a closed term of the function's return type, i.e. a term that contains no free variables. For example, the function `square` for a natural number can be defined by:

```
definition square :: "nat ⇒ nat"
where "square x ≡ x⋆x"
```

The keywords **primrec**, **fun**, and **function** are used to define *recursive functions*. All of these keywords define recursive functions using multiple equations. For instance, the functions `set` and `append` that, respectively, convert a list to a set and concatenate two lists are defined by:

```
primrec set :: "'a list ⇒ 'a set"
where
"set [] = {}" |
"set (Cons x xs) = {x} ∪ (set xs)"

primrec append :: "'a list ⇒ 'a list ⇒ 'a list"
where
"[] @ ys = ys" |
"(x#xs) @ ys = x # xs @ ys"
```

Each of the function definitions consists of two equations (separated by `|`). The first equation defines the function's return value for an empty list, the second equation for a non-empty list (using recursion). Here pattern matching is used to decide which of the two equations is relevant for a given parameter. In pattern matching, the special symbol "_" is used as a wild-card for arbitrary terms.

The three keywords **`primrec`**, **`fun`**, or **`function`** define different classes of recursive functions. The keyword **`primrec`** defines recursive functions by giving exactly one reduction rule for each constructor. In contrast, **`fun`** and **`function`** define recursive functions without these restrictions. However, it is then required to prove the exhaustiveness of the function (each possible input is covered by one equation), the compatibility of patters (each possible input is covered by exactly one equation), and termination. For functions defined using the keyword **`fun`**, these proofs are performed automatically. For functions defined using the keyword **`function`**, these proofs must be performed manually. Hence, **`function`** must be used instead of **`fun`** if the automatic proofs fail.

Theorems, Lemmas, and Proofs. Theorems and lemmas in Isabelle/HOL are defined using the keywords **`theorem`** and **`lemma`** followed by a formula or a proposition in Isabelle/HOL's meta-logic. For instance, that from `A` and `B` it follows that `A ∧ B` holds is captured by the following theorem:

```
theorem "⟦ A ; B ⟧ ⟹ A ∧ B"
```

Note that there is no difference between theorems and lemmas except the intuition that a theorem is more important than a lemma.

Proofs of lemmas or theorems can be generated interactively by applying a series of proof commands in one of the two verification languages supplied by Isabelle/HOL. With the language *apply*, proof tactics can be applied to transform the proof obligations until they are fully discharged. With the language *Isar*, proofs can be generated in a mathematical fashion close to proofs on paper. It is also possible to mix both languages. For an introduction to proving in Isabelle/HOL see [UT18a].

3 Structure of I-MAKS and Basic Definitions

I-MAKS is an Isabelle/HOL formalization of MAKS in its version from [Man03]. The structure of theories in *I-MAKS* is depicted in Fig. 1. *I-MAKS* consists of two top-level components, a specification and a verification component. The *specification component* contains all parts of *I-MAKS* related to supported system models and security properties. Structurally, the specification component again consists of two subcomponents: *system specification* and *security specification*. The former contains everything related to the specification of system behavior in *I-MAKS*. The latter contains everything related to the specification of information-flow properties in *I-MAKS*. The *verification component* contains all parts of *I-MAKS* related to the verification of security properties.

On a technical level, each component consists of a collection of theories. In Fig. 1, the theories are visualized by boxes where a box with a thick border corresponds to

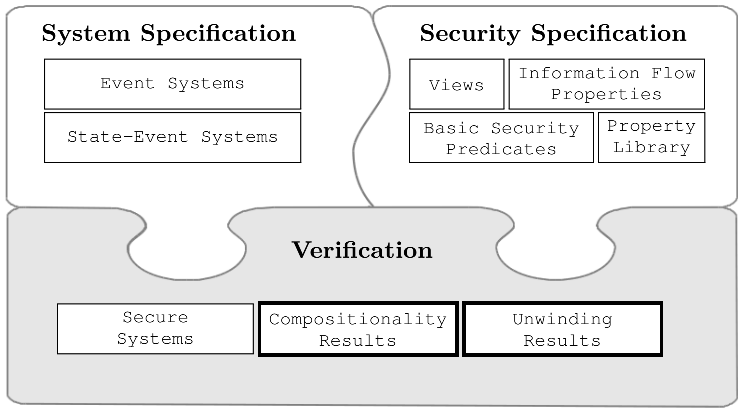

Figure 1: Structure of Theories in *I-MAKS*

multiple theories. For instance, the system specification component consists of two theories: `Event Systems` and `State-Event Systems`.

Basic Definitions in I-MAKS. Before we present the individual components of *I-MAKS* in the following sections, we present a few basic definitions used in the components of *I-MAKS*. These basic definitions formalize the notions prefix and closed under prefixes (cf. Definition 4) as well as the definition of projection of a trace τ to a set of events E (cf. Definition 5) in Isabelle/HOL. The definitions are located in the theories `Prefix` and `Projection`.

The notion of a prefix is formalized by the binary predicate `prefix`.

definition *prefix :: "'e list ⇒ 'e list ⇒ bool"* (**infixl** *"≼" 100)*
where
"(l1 ≼ l2) ≡ (∃l3. l1 @ l3 = l2)"

As syntactic abbreviation for the predicate `prefix l1 l2`, *I-MAKS* supports the infix notation `l1 ≼ l2`.

The notion of closed under prefixes is formalized by the predicate `prefixclosed`.

definition *prefixclosed :: "('e list) set ⇒ bool"*
where
"prefixclosed tr ≡ (∀l1 ∈ tr. ∀l2. l2 ≼ l1 ⟶ l2 ∈ tr)"

The notion of projection of `l` to a set `E` is formalized by the function `projection`.

definition *projection:: "'e list ⇒ 'e set ⇒ 'e list"* (**infixl** *"↿" 100)*
where
"l ↿ E ≡ filter (λx . x ∈ E) l"

As syntactic abbreviation for the function `projection l E`, *I-MAKS* supports the infix notation `l ↿ E`.

4 System Specification Component

The underlying system models of *MAKS* are event systems and state-event systems. These two system models are the conceptual basis for the specification and verification of information-flow security in *MAKS*.

We introduce the two system models, emphasize the close relationship between the two system models by providing a translation from state-event systems to event systems, and provide means for the specification of complex systems by composition.

Remark. The following subsections first introduce notions of *MAKS* using mathematical notation and then provide the corresponding formalization of these notions in *I-MAKS* using the syntax of Isabelle/HOL. For the sake of clarity, definitions of notions in *I-MAKS* are denoted in small capital letters, e.g. EVENT SYSTEM denotes the notion of event systems in *I-MAKS*.

4.1 Event Systems

Event systems extend the notion of processes (cf. Definition 6) by explicit input and output interfaces to the outside modeled by subsets of events.

Definition 13. *An* event system *ES is a tuple* (E, I, O, Tr) *such that* $I \subseteq E$, $O \subseteq E$, $I \cap O = \emptyset$, $Tr \subseteq E^*$, *and Tr is prefix closed.* ◇

The sets I and O model the in- and output interfaces of a system. This means each event in I models an input action, each event in O models an output action and each event neither in I nor in O models an internal action. Note that the two sets I and O are disjoint and, thus, feedback loops must be modeled internally.

***I-MAKS* Formalization.** In *I-MAKS*, an event system is formalized by a combination of a record type and a predicate (see Theory `Event Systems`). While the record type formalizes the elements of an event system, the predicate, referred to as *validity predicate*, formalizes the semantic side conditions on these elements, i.e. when a term of the record type is a *valid* event system.

The record type formalizing the elements of an event system is the parametric record type `'e ES_rec` where the type variable `'e` corresponds to the type of events of the record type.

```
record 'e ES_rec =
  E_ES ::  "'e set"
  I_ES ::  "'e set"
  O_ES ::  "'e set"
  Tr_ES :: "('e list) set"
```

A term of record type `'e ES_rec` consists of the fields `E_ES` (the events), `I_ES` (the input events), `O_ES` (the output events) and `Tr_ES` (the possible traces) that correspond to the sets E, I, O and Tr of an event system. *I-MAKS* provides the following syntactic abbreviation for retrieving the value of a field: Given a record `R`, $\texttt{F}_{\texttt{R}}$ is equivalent to `(F R)`.

The semantic side conditions for terms of the record type `'e ES_rec` are formalized by the predicate `ES_valid`.

```
definition ES_valid :: "'e ES_rec ⇒ bool"
where
"ES_valid ES ≡
  es_inputs_are_events ES ∧ es_outputs_are_events ES
  ∧ es_inputs_outputs_disjoint ES ∧ traces_contain_events ES
  ∧ traces_prefixclosed ES"
```

```
definition es_inputs_are_events :: "'e ES_rec ⇒ bool"
where
"es_inputs_are_events ES ≡ I_ES ⊆ E_ES"
```

```
definition es_outputs_are_events :: "'e ES_rec ⇒ bool"
where
"es_outputs_are_events ES ≡ O_ES ⊆ E_ES"
```

```
definition es_inputs_outputs_disjoint :: "'e ES_rec ⇒ bool"
where
"es_inputs_outputs_disjoint ES ≡ I_ES ∩ O_ES = {}"
```

```
definition traces_contain_events :: "'e ES_rec ⇒ bool"
where
"traces_contain_events ES ≡ ∀ l ∈ Tr_ES. ∀ e ∈ (set l). e ∈ E_ES"
```

```
definition traces_prefixclosed :: "'e ES_rec ⇒ bool"
where
"traces_prefixclosed ES ≡ prefixclosed Tr_ES"
```

Based on the record type `'e ES_rec` and the predicate `ES_valid`, event systems in *I-MAKS* are formalized as follows.

Definition 14. *An* EVENT SYSTEM *for a type of events* `'e` *is a term of the record type* `'e ES_rec` *that satisfies the predicate* `ES_valid`. ◇

4.2 State-Event Systems

The second system model of *MAKS*, state-event systems, extend the notion of labeled transition systems (cf. Definition 9) by explicit input and output interfaces to the outside modeled by subsets of events.

Definition 15. *A* state-event system *SES is a tuple* (S, s_0, E, I, O, T) *such that* $s_0 \in S$, $I \subseteq E$, $O \subseteq E$, $I \cap O = \emptyset$, $T \subseteq S \times E \times S$ *and* T *contains at most one triple* (s, e, s') *for each* $s \in S$ *and* $e \in E$. ◇

The set of input events I and the set of output events O respectively, model the in- and output actions of the system. Note that the transition relation is further restricted and at most contains one transition for each event in each state. This restriction ensures determinism in the effect of an event but still permits non-determinism in the choice of the event.

***I-MAKS* Formalization.** State-event systems in *I-MAKS* are formalized by a combination of a record type and a corresponding validity predicate (see Theory `State-Event Systems`).

The record type `('s 'e) SES_rec` formalizing the elements of state-event systems is parametric in both the type of states `'s` and the type of events `'e`.

```
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 ⇒ 'e ⇀ 's"
```

Each term of the record type `('s 'e) SES_rec` consists of the fields `S_SES` (the states), `s0_SES` (the initial state), `E_SES` (the events), `I_SES` (the input events), `O_SES` (the output events) and `T_SES` the transition relation. These fields correspond to the respective sets of state-event systems. Note that the transition relation is defined as partial function from a state and an event to a unique successor state. Hence, the last semantic side condition of state-event systems on the transition relation, namely, that for each state and each event there exists at most one transition is already covered. As syntactic abbreviation for $(T_{SES}\ s\ e) = Some\ s'$, *I-MAKS* supports the usage of $s\ e \longrightarrow_{SES}\ s'$.

Similar to `ES_valid`, the predicate `SES_valid` defines the semantic side conditions on terms of the record type `('s 'e) SES_rec`.

```
definition SES_valid :: "('s, 'e) SES_rec ⇒ bool"
where
"SES_valid SES ≡
  s0_is_state SES ∧ ses_inputs_are_events SES
  ∧ ses_outputs_are_events SES ∧ ses_inputs_outputs_disjoint SES ∧
  correct_transition_relation SES"

definition s0_is_state :: "('s, 'e) SES_rec ⇒ bool"
where
"s0_is_state SES ≡ s0_SES ∈ S_SES"

definition ses_inputs_are_events :: "('s, 'e) SES_rec ⇒ bool"
where
"ses_inputs_are_events SES ≡ I_SES ⊆ E_SES"

definition ses_outputs_are_events :: "('s, 'e) SES_rec ⇒ bool"
where
"ses_outputs_are_events SES ≡ O_SES ⊆ E_SES"

definition ses_inputs_outputs_disjoint :: "('s, 'e) SES_rec ⇒ bool"
where
"ses_inputs_outputs_disjoint SES ≡ I_SES ∩ O_SES = {}"
```

definition *correct_transition_relation :: "('s, 'e) SES_rec ⇒ bool"*
where
"correct_transition_relation SES ≡
∀x y z. x y$\longrightarrow_{SES}$ z ⟶ ((x ∈ S_{SES}) ∧ (y ∈ E_{SES}) ∧ (z ∈ S_{SES}))"

Based on these definitions, state-event systems in *I-MAKS* are formalized as follows.

Definition 16. *A* STATE-EVENT SYSTEM *for a type of states* 's *and type of events* 'e *is a term of the record type* ('s, 'e) SES_rec *that satisfies the predicate* SES_valid*.* ◇

4.3 Translation from State-Event Systems to Event Systems

State-event systems can be translated to event systems. The translation is based on the extension from the small-step transition relation of a state-event system to a big-step transition relation induced by the state-event system.

Definition 17. *Let $SES = (S, s_0, E, I, O, T)$ be a state-event system. The* induced big-step transition relation of *SES (denoted by $\widehat{T}_{SES}$) is defined by the smallest set $\widehat{T}_{SES} \subseteq S \times E^* \times S$ satisfying the conditions*

1. $\forall s \in S.(s, \langle\rangle, s') \in \widehat{T}_{SES}$ *and*
2. $\forall s, s', s'' \in S. \forall e \in E. \forall \tau \in E^*.$
   $\big[\big((s, e, s') \in T_{SES} \wedge (s', \tau, s'') \in \widehat{T}\big) \Rightarrow (s, \langle e\rangle.\tau, s'') \in \widehat{T}_{SES}\big].$ ◇

That is, there is a big-step transition from a state s to another state s' with the trace τ if and only if there is a sequence of transitions in T from s to s' with the events of τ in their order of occurrence.

Based upon the induced big-step transition relation of a state-event system, the set of possible traces of a state-event system is defined as follows.

Definition 18. *Let $SES = (S, s_0, E, I, O, T)$ be a state-event system. The* set of possible traces of *SES (denoted by Tr_{SES}) is defined by $Tr_{SES} = \{\tau \in E^* \mid \exists s \in S. (s_0, \tau, s) \in \widehat{T}_{SES}\}$.* ◇

This means the set of possible traces of a state-event systems consists of all traces for which a big-step transition from the initial state to another state is possible.

Using the translation from the transition relation of a state-event system to the possible traces of a state-event system the event system induced by a state-event system is defined as follows.

Definition 19. *Let $SES = (S, s_0, E, I, O, T)$ be a state-event system. The* event system induced by *SES is the event system $ES = (E, I, O, Tr_{SES})$.* ◇

That is, the event system obtained by replacing the notion of state and the transition relation that modeled the behavior of the state-event system with the induced set of possible traces.

***I-MAKS* Formalization.** *I-MAKS* adopts the translation using three functions (see Theory `State-Event Systems`): the recursive function `path`, the function `possible_traces`, and the function `induceES`.

With the recursive function `path` *I-MAKS* provides the induced big-step transition relation as a partial function.

```
primrec path :: "('s, 'e) SES_rec ⇒ 's ⇒ 'e list ⇀ 's"
where
path_empt: "path SES s1 [] = (Some s1)" |
path_nonempt: "path SES s1 (e # t) =
  (if (∃s2. s1 e⟶SES s2)
  then (path SES (the (TSES s1 e)) t)
  else None)"
```

Based on this function `path`, the possible traces of a given record of type `('s 'e) SES_rec` are formalized by the function `possible_traces`.

```
definition possible_traces :: "('s, 'e) SES_rec ⇒ ('e list) set"
where
"possible_traces SES ≡ {t. (enabled SES s0SES t)}"
```

```
definition enabled :: "('s, 'e) SES_rec ⇒ 's ⇒ 'e list ⇒ bool"
where
"enabled SES s t ≡ (∃s'. s t⟹SES s')"
```

Using this function, *I-MAKS* formalizes the translation from a record of type `('s 'e) SES_rec` to a record of type `'e ES_rec` by the function `induceES`.

```
definition induceES :: "('s, 'e) SES_rec ⇒ 'e ES_rec"
where
"induceES SES ≡
 (|
  E_ES = ESES,
  I_ES = ISES,
  O_ES  = OSES,
  Tr_ES = possible_traces SES
 |)"
```

That is, the events, input events, and output events remain unchanged but the notion of state and the transition function is replaced by the set of possible traces.

As part of *I-MAKS*, it is proven that applying the function `induceES` to a STATE-EVENT SYSTEM evaluates to an EVENT SYSTEM.

```
lemma induceES_yields_ES:
  "SES_valid SES ⟹ ES_valid (induceES SES)"
```

Based on this lemma the EVENT SYSTEM induced by a given STATE-EVENT SYSTEM in *I-MAKS* is defined as follows.

Definition 20. *The* EVENT SYSTEM `ES` *for a type of events* `'e` induced by *a* STATE-EVENT SYSTEM `SES` *for a type of states* `'s` *and the type of events* `'e` *is defined by the result of* `induceES SES`*.* ◇

4.4 Parallel Composition of Event Systems

For the specification of complex systems, *MAKS* supports the composition of event systems to complex systems.

Based upon the interfaces of event systems, *MAKS* only considers certain event systems as composable.

Definition 21. *Let* $ES_1 = (E_1, I_1, O_1, Tr_1)$ *and* $ES_2 = (E_2, I_2, O_2, Tr_2)$ *be two event systems. The two event systems* ES_1 *and* ES_2 *are* composable *if and only if* $E_1 \cap E_2 \subseteq (O_1 \cap I_2) \cup (O_2 \cap I_1)$. $\diamond$

That is, two event systems are composable if each shared event is an input event of one event system and an output event of the other event system. Hence, event systems only communicate on their interfaces.

Considering two composable event systems, *MAKS* provides the following notion of parallel composition.

Definition 22. *Let* $ES_1 = (E_1, I_1, O_1, Tr_1)$ *and* $ES_2 = (E_2, I_2, O_2, Tr_2)$ *be two composable event systems. Then the* parallel composition of ES_1 and ES_2 *is the event system* $ES = (E, I, O, Tr)$ *defined by*

$$\begin{aligned}
E &= E_1 \cup E_2 \\
I &= (I_1 \setminus O_2) \cup (I_2 \setminus O_1) \\
O &= (O_1 \setminus I_2) \cup (O_2 \setminus I_1) \\
Tr &= \{\tau \in E^* \mid \tau \upharpoonright E_1 \in Tr_1 \wedge \tau \upharpoonright E_2 \in Tr_2\}.
\end{aligned} \qquad \diamond$$

The parallel composition of two event systems, the *components*, is the event system, *the composed event system*, modeling a system that can engage in the actions modeled by one of the two components. The parallel composition of event systems preserves the interfaces of the two components except that all events used for communication between the two components become internal events. Moreover, the set of possible traces of the composed event system consists of all traces that projected to the events of each component are possible traces of the components. Hence, in each possible trace of the composed event system its components agree on the occurrence of shared events. This means the events shared at the interface of the two components are means for communication, effectively, establishing a blocking message passing communication between the two components.

***I-MAKS* Formalization.** *I-MAKS* adopts *MAKS*' notion of parallel composition by a predicate `composable` and a function `composeES` (see Theory `Event Systems`.

The predicate `composable` transfers the notion of composable event systems in *MAKS* to *I-MAKS*, i.e. to terms of the record type `'e ES_rec`.

```
definition composable :: "'e ES_rec ⇒ 'e ES_rec ⇒ bool"
where
"composable ES1 ES2 ≡ E_ES1 ∩ E_ES2 ⊆ (O_ES1 ∩ I_ES2) ∪ (O_ES2 ∩ I_ES1)"
```

Hence, two EVENT SYSTEMs are composale iff the predicate `composable` evaluates to true for those EVENT SYSTEMs.

Definition 23. *Two* EVENT SYSTEMS *for a type of events* `'e` *are* COMPOSABLE *if they satisfy the predicate* `composable`*.* ◇

Likewise to the predicate `composable`, the function `composeES` transfers *MAKS*' parallel composition to *I-MAKS*, i.e. to terms of the record type `'e ES_rec` for a type of events `'e`.

```
definition composeES :: "'e ES_rec ⇒ 'e ES_rec ⇒ 'e ES_rec"
where
"composeES ES1 ES2 ≡  ⦇
  E_ES   = E_ES1 ∪ E_ES2,
  I_ES   = (I_ES1 - O_ES2) ∪ (I_ES2 - O_ES1),
  O_ES   = (O_ES1 - I_ES2) ∪ (O_ES2 - I_ES1),
  Tr_ES = {τ. (τ ↿ E_ES1) ∈ Tr_ES1 ∧ (τ ↿ E_ES2) ∈ Tr_ES2
              ∧ (set τ ⊆ E_ES1 ∪ E_ES2)} ⦈"
```

As syntactic abbreviation, *I-MAKS* supports the usage of `ES1 ‖ ES2` instead of `composeES ES1 ES2` for two terms `ES1` and `ES2` of the record type `'e ES_rec`.

As part of *I-MAKS*, it is proven that applying the function `composeES` to two COMPOSABLE EVENT SYSTEMS always evaluates to an EVENT SYSTEM.

```
lemma composeES_yields_ES :
 "(ES_valid ES1 ∧ ES_valid ES2) ⇒ ES_valid (ES1 ‖ ES2)"
```

Together with the insights of the lemma above, the predicate `composable` and the function `composeES`, *I-MAKS* defines the parallel composition of two COMPOSABLE EVENT SYSTEMs as follows.

Definition 24. *The* PARALLEL COMPOSITION *of two composable* EVENT SYSTEMS `ES1` *and* `ES2` *for a type of events* `'e` *is defined by the result of* `composeES ES1 ES2`*.* ◇

5 Security Specification Component

MAKS supports the specification of information-flow properties in a modular fashion. Concretely, the definition of an information-flow property is split in two parts: (1) the specification of the attacker's perspective and (2) the specification of the information-flow property w.r.t. an arbitrary attacker's perspective.

To this end, *MAKS* introduces the notion of views to model the perspective of an attacker on an event-based system model and a *MAKS*-specific notion of information-flow properties that are constructed using building blocks, so called basic security predicates (BSPs).

Remark. As in the previous section, the following subsections first introduce the notions of *MAKS* using mathematical notation and then provide the corresponding formalization of the notions in *I-MAKS* using the syntax of Isabelle/HOL.

5.1 Views

The perspective of an attacker that passively observes the visible behavior of a system, a so called observer, is formalized by the notion of a view on a set of events modeling the actions of the system.

Definition 25. *A* view $\mathcal{V}$ *is a triple* (V, N, C) *such that* $V \cap N = \emptyset$*,* $V \cap C = \emptyset$ *and* $N \cap C = \emptyset$*. A view* $\mathcal{V} = (V, N, C)$ *is a* view on E *where* E *is a set of events if* $V \cup N \cup C = E$*.* ◇

That is, a view on a set of events E is a disjoint partition of E into the set of *visible events* V, the set of *confidential events* C, and the set of *don't care events* N. The set of visible events models what actions an attacker can observe and, thus, must contain all events modeling actions that are observable to the attacker. The set of confidential events models what actions shall remain secret to an attacker, i.e. the attacker can neither observe these actions nor shall he be able to infer information about them. Hence, C must contain all events modeling secret actions. Finally, the set of don't care events models what actions are neither observable to the attacker nor shall remain secret to an attacker. Hence, N may contain all remaining events that must be contained in neither V nor C.

Note that a view provides a more fine-grained definition of an attacker's interface than the traditional partition of E into L and H used, e.g. in Definition 10 and 11.

***I-MAKS* Formalization.** *I-MAKS* formalizes views by a combination of the record type `'e V_rec` for a type of events `'e` and a corresponding predicate `V_valid` (see Theory `Views`). The fields `V`, `N`, `C` correspond to respective elements of a view, i.e. the visible events, the don't care events, and the confidential events.

definition *V_valid :: "'e V_rec ⇒ bool"*
where
"V_valid v ≡ VN_disjoint v ∧ VC_disjoint v ∧ NC_disjoint v"

definition *VN_disjoint :: "'e V_rec ⇒ bool"*
where
"VN_disjoint v ≡ $V_v \cap N_v$ *= {}"*

definition *VC_disjoint :: "'e V_rec ⇒ bool"*
where
"VC_disjoint v ≡ $V_v \cap C_v$ *= {}"*

definition *NC_disjoint :: "'e V_rec ⇒ bool"*
where
"NC_disjoint v ≡ $N_v \cap C_v$ *= {}"*

The predicate `V_valid` formalizes the side conditions on the fields of a term of record type `'e V_rec` to represent a view. Based on this predicate views in *I-MAKS* are formalized as follows.

Definition 26. *A* VIEW *for a type of events* `'e` *is a term of the record type* `'e V_rec` *that satisfies the predicate* `V_valid`*.* ◇

$$\tau \in Tr \xrightarrow[\text{modifies occurrences of events in } C]{\text{perturbation}} t \in E^* \xrightarrow[\text{modifes occurrences of events in } N]{\text{correction}} \tau' \in Tr$$

Figure 2: Pattern underlying the definition of all BSPs [Man03].

The notion of a view on a set of events E that models the perspective of an observer on the actions modeled by E is formalized by VIEWs satisfying the required side conditions. The side conditions are formalized as the predicate `isViewOn`.

definition *isViewOn :: "'e V_rec ⇒ 'e set ⇒ bool"*
where
"isViewOn $\mathcal{V}$ E ≡ V_valid $\mathcal{V}$ ∧ $V_{\mathcal{V}}$ ∪ $N_{\mathcal{V}}$ ∪ $C_{\mathcal{V}}$ = E"

The predicate `isViewOn` requires that the term of record type `'e V_rec` forms a disjoint partition of a set of events `E` of type `'e`.

Based upon this predicate, *I-MAKS* formalizes the perspective of an observer on a set of events as follows.

Definition 27. *Let* `E` *be a set of events of a type* `'e`*. A* VIEW ON `E` *is a* VIEW *for the type of events* `'e` *that satisfies the predicate* `isViewOn` *for* `E`*.* ◇

5.2 Basic Security Predicates

For the specification of information-flow properties in a modular and uniform fashion, *MAKS* provides a set of building blocks, the *basic security predicates* (*BSPs*). Each BSP is a closure property on sets of traces and is parametric in a view.

Definition 28. *A* basic security predicate BSP *is a function that maps a view on a set of events E to a closure property on sets of traces over E.* ◇

Hence, a BSP defines a information-flow requirement on systems modeled by a set of traces that is parametric in the attacker's perspective that observes the system.

As notational convention $BSP_{\mathcal{V}}$ denotes the closure property on sets of traces obtained by applying a BSP BSP to a view $\mathcal{V}$.

All BSPs are defined in a *perturbation and correction pattern* (cf. Fig. 2). The *perturbation* defines the information about confidential events that shall remain secret in terms of modifications to the occurrences of confidential events. The *correction* defines the permitted modifications to the occurrences of don't care events. For instance, consider the BSP *Backwards-Strict Deletion (BSD)*.

Definition 29. *Let E be a set of events. Let Tr be set of traces over E. Let $\mathcal{V} = (V, N, C)$ be a view on E. The basic security predicate* Backwards-Strict Deletion *(denoted by BSD) is defined by:*

$$\begin{aligned}
&BSD_{\mathcal{V}}(Tr) \equiv \\
&\quad \forall \alpha, \beta \in E^*. \forall c \in C. \\
&\qquad [(\beta.\langle c \rangle.\alpha \in Tr \wedge \alpha \restriction C = \langle\rangle) \\
&\qquad\quad \Rightarrow (\exists \alpha' \in E^*.\ \beta.\alpha' \in Tr \wedge \alpha' \restriction V = \alpha \restriction V \wedge \alpha' \restriction C = \langle\rangle)]
\end{aligned}$$
◇

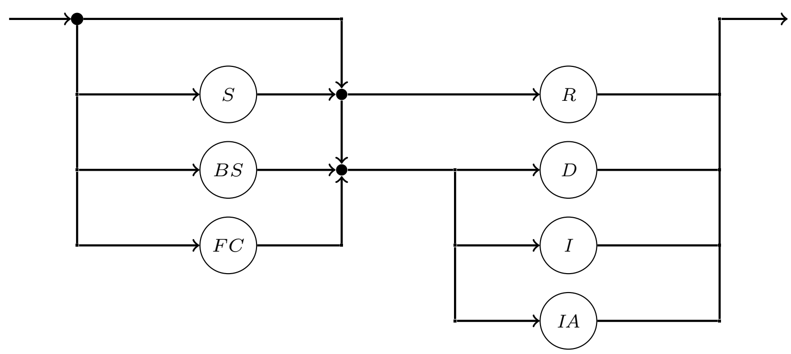

Figure 3: Names of BSPs in *MAKS* [Man03].

For BSD, the perturbation is the deletion of the last occurrence of an confidential event and the permitted correction are changes to the occurrences of don't care events after the point where the confidential event is deleted. Intuitively, BSD requires that each trace containing at least one occurrence of a confidential event can be explained by an alternative trace where the last occurrence of an confidential event is deleted. Hence, an attacker cannot be certain about the occurrence of the deleted confidential event based on his observation.

Naming Convention of BSPs. The name of a BSP indicates its perturbation and correction pattern. For BSD, the prefix BS indicates that backwards-strict corrections are permitted, i.e. corrections that affect the future wrt. the point of the perturbation of a trace but not the past. The suffix D indicates the perturbation *Deletion*, i.e. the removal of the last occurrence of a confidential event in a trace.

Overall *MAKS* provides four perturbations, identified by the symbols R, D, I and IA. There are also four corrections, three of them are identified by the symbols S, BS and FC, while the fourth is identified implicitly by not using one of these symbols. Within the name of a BSP, the correction identifier appears before the perturbation identifier. The syntax diagram in Fig. 3 visualizes all possible combinations of the symbols for perturbations and corrections. The diagram shall be read as follows: Starting from the top left, each sequence of correction and perturbation symbols on a possible path ending in the top right is a possible name of a BSP.

The blank prefix means that *arbitrary* modifications in don't care events are permitted. The prefix S for *strict* means that no corrections at all are permitted. The prefixes BS for *backwards-strict* and FC for *forward correctable* mean that only *causal* corrections are permitted, i.e. modifications that affect the future w.r.t. the point in the perturbation of the system run but not the past. The prefix FC restricts the permitted correction even further than BS. Because FC is not relevant in this report, we omit an explanation here and refer the reader to [Man03, Page 50f].

BSPs with the suffix R for *removal* or the suffix D for *deletion* prevent deductions about the occurrence of confidential events in a trace. More precisely, BSPs with the suffix R in their name ensure that an observer is unable to infer that a given trace occurred or an alternative possible trace without any confidential events yielding the same observation occurred. Similarly, BSPs with the suffix *D* ensure that an observer is unable to infer that a confidential event that may potentially occur in a trace did occur in this trace.

In contrast, BSPs with the suffix *I* for *insertion* or the suffix *IA* for *insertion of admissible events* prevent deductions about the non-occurrences of events in a trace. More precisely, BSPs with the suffix *I* ensure that an observer is unable to infer whether a confidential event did not occur in a trace. Similarly, BSPs with the suffix *IA* ensure that an observer is unable to infer whether a confidential event that may potentially occur in a trace did not occur in this trace.

Overall *MAKS* provides 14 BSPs derived in the naming pattern presented in Fig. 3. We omit the definitions of the remaining 13 BSPs and instead provide them in their *I-MAKS* representation in the following or in Appendix A.1.

I-MAKS **Formalization.** *I-MAKS* adopts the concept of BSPs in Isabelle/HOL in Theory `Basic Security Predicates`. As part of the theory, it formalizes all 14 BSPs presented in [Man03] in Isabelle/HOL.

I-MAKS formalizes BSPs as a combination of a type for BSPs and a predicate ensuring the closure property requirement on the binary predicate.

```
type_synonym 'e BSP = "'e V_rec ⇒ (('e list) set) ⇒ bool"
```

```
definition BSP_valid :: "'e BSP ⇒ bool"
where
"BSP_valid bsp ≡
  ∀𝒱 Tr E. ( isViewOn 𝒱 E ∧ areTracesOver Tr E )
              ⟶ (∃ Tr'. Tr' ⊇ Tr  ∧ bsp 𝒱 Tr')"
```

In combination *I-MAKS* formalizes BSPs as follows.

Definition 30. *A* BASIC SECURITY PREDICATE (BSP) *for a type of events* `'e` *is a term of the type* `'e BSP` *that satisfies the predicate* `BSP_valid`*.* ◇

Note that, in addition to VIEWs, BSPs can also depend on additional parameters to express more complex security requirements.

In the following, we present three examples of BSPs formalized in *I-MAKS*. We extracted these examples from the theory `Basic Security Predicates` where the remaining 11 BSPs are also formalized and lemmas that `BSP_valid` is satisfied for each of the 14 BSPs are proven.

The first example, *removal (R)*, is formalized by the predicate `R`.

```
definition R :: "'e BSP"
where
"R 𝒱 Tr ≡
  ∀τ∈Tr. ∃τ'∈Tr. τ' ↿ C𝒱 = [] ∧ τ' ↿ V𝒱 = τ ↿ V𝒱"
```

`R` intuitively requires that removing *all* occurrences of confidential events from a possible trace τ can be corrected to another possible trace τ'. To obtain τ', the perturbed trace $\tau \upharpoonright V_{\mathcal{V}} \cup N_{\mathcal{V}}$ may be corrected by inserting or removing don't care events at arbitrary points, while the visible events are left untouched. Thus, if R holds, an attacker with VIEW $\mathcal{V}$ cannot infer from a possible observation whether the trace that occurred contained confidential events or not.

Permitting corrections at arbitrary points can be too liberal for some systems, i.e. might not detect all intuitively insecure behavior. Moreover, requiring only alternative traces without any confidential events might not capture the desired information-flow property. The BSP BSD (cf. Definition 29) is less liberal and permits only causal corrections. It is formalized in *I-MAKS* by the predicate BSD.

```
definition BSD :: "'e BSP"
where
"BSD 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ [c] @ α) ∈ Tr ∧ α↿C𝒱 = [])
    ⟶ (∃α'. ((β @ α') ∈ Tr ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []))"
```

The two BSPs shown so far require that deleting all or only the last occurrence(s) of confidential events in a trace can be corrected to another trace with the same observation for the attacker. *I-MAKS* also defines BSPs requiring that the insertion of confidential events somewhere after the last confidential event of the initial trace can be corrected accordingly. An example for such a BSP is BSI.

```
definition BSI :: "'e BSP"
where
"BSI 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ α) ∈ Tr ∧ α↿C𝒱 = [])
    ⟶ (∃α'. ((β @ [c] @ α') ∈ Tr ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []))"
```

BSI requires that inserting a confidential event c somewhere after the last occurrence of a confidential event in a possible trace β @ α can be corrected to another possible trace β @ [c] @ α'. Likewise to BSD, BSI permits only causal corrections after the perturbation of the considered trace, i.e. corrections in the perturbed trace β @ [c] @ α may only affect α. Intuitively, all BSPs inserting occurrences of confidential events in a possible trace ensure that an attacker is unable to infer that a confidential event did not occur during a trace.

Beyond the formalization of all 14 BSPs presented in [Man03] in Isabelle/HOL, *I-MAKS* also provides a complete Isabelle/HOL formalization including proofs for the taxonomy of the 14 BSPs in Theory BSPTaxonomy. The complete list of taxonomy results formalized and proven in *I-MAKS* can be found in Appendix A.2.

5.3 Information-Flow Properties

In *MAKS* more complex information-flow properties than the BSPs themselves are defined as conjunction of multiple BSPs for a set of views on some set of events.

Definition 31. *Let E be a set of events. An* information-flow property *is a pair* $(\mathcal{VS}, BSPS)$ *where* $\mathcal{VS}$ *is a set of views on* E *and* $BSPS$ *is a set of BSPs.*
An information-flow property is satisfied for a set of traces $Tr \subseteq E^*$ *iff* $BSP_{\mathcal{V}}(Tr)$ *holds for each* $\mathcal{V} \in \mathcal{VS}$ *and each* $BSP \in BSPS$. $\diamond$

That is, an information-flow property consists of the perspective of the attackers against whom the system shall be secure and the building blocks that conjoined specify the desired notion of information-flow security. A system is considered secure wrt. the defined information-flow property if each of the BSPs is satisfied for each view and set of possible traces modeling the system.

Examples for how *MAKS*' notion of information-flow properties and BSPs can be used to define information-flow properties such as the properties in Section 2.2 are omitted here and instead given in their Isabelle/HOL formalization in the following.

***I-MAKS* Formalization.** Like in *MAKS*, information-flow properties in *I-MAKS* are structurally formalized as a pair of a set of VIEWs and a set of BSPs.

type_synonym *'e IFP_type = "('e V_rec set) × 'e SP"*

type_synonym *'e SP = "('e BSP) set"*

The corresponding predicate `IFP_valid` formalizes the semantic side conditions of information-flow properties on terms of the type `'e IFP_type`.

```
definition IFP_valid :: "'e set ⇒ 'e IFP_type ⇒ bool"
where
"IFP_valid E ifp ≡
  ∀𝒱 ∈ (fst ifp). isViewOn 𝒱 E
                    ∧ (∀BSP ∈ (snd ifp). BSP_valid BSP)"
```

Combining the type `'e IFP_type` and the predicate `IFP_valid`, *I-MAKS* formalizes information-flow properties as follows.

Definition 32. *An* INFORMATION-FLOW PROPERTY *for a type of events* `'e` *is a term of* `'e IFP_type` *that satisfies the predicate* `IFP_valid`*.* ◇

I-MAKS adopts the notion of satisfaction for information-flow properties utilizing the predicate `IFPIsSatisfied`.

```
definition IFPIsSatisfied :: "'e IFP_type ⇒ ('e list) set  ⇒ bool"
where
"IFPIsSatisfied ifp Tr ≡
  ∀ 𝒱∈(fst ifp). ∀ BSP∈(snd ifp). BSP 𝒱 Tr"
```

Hence, a set of traces satisfies an INFORMATION-FLOW PROPERTY if and only if `IFPIsSatisfied` is satisfied.

Definition 33. *Let* `ifp` *be an* INFORMATION-FLOW PROPERTY*. Let* `Tr` *be a set of traces. the* INFORMATION-FLOW PROPERTY `ifp` *is satisfied for* `Tr` *if and only if* `IFPIsSatisfied ifp Tr` *is satisfied.* ◇

Using INFORMATION-FLOW PROPERTIES *I-MAKS* provide formalizations of several possibilistic information-flow properties from the literature in Theory `Property Library`. For instance, noninference (cf. Definition 10) is formalized as follows.

definition *NF :: "'e set ⇒ 'e set ⇒ 'e IFP_type"*
where
"NF L H ≡ ({HighConfidential L H}, {R})"

definition *HighConfidential :: "'e set ⇒ 'e set ⇒ 'e V_rec"*
where
"HighConfidential L H ≡ ⦇ V=L, N={}, C=H ⦈"

This formalization of noninference is equivalent to the formalization of noninference in *MAKS* (see [Man03]). This also holds for the other information-flow properties formalized in Theory `Property Library`, i.e. they are also equivalent to their formalizations in *MAKS*. We provide the *I-MAKS* formalization of all information-flow properties expressed in *MAKS* presented in [Man03] in Appendix A.4.

6 Verification Component

MAKS provides support for reasoning by unwinding on state-event systems in the form of unwinding results. That is, a sound proof-technique for the verification of BSPs reasoning about local requirements on single transitions and adjacent states instead of reasoning about the set of all possible traces of a state-event system.

In addition, *MAKS* provides support for compositional reasoning by compositionality results for the different BSPs. These compositionality results allow one to reason about the security of a system's components separately and then establish security for the overall system.

In this section, we provide the *I-MAKS* representation of the unwinding result for BSD and provide the compositionality result for BSD. We provide the representation of the remaining unwinding results and compositionality results of *MAKS* in *I-MAKS* in Appendix A.5 and Appendix A.6.

6.1 Unwinding Results

Reasoning by unwinding reduces the verification of BSPs for a given state-event system to the verification of two so called *unwinding conditions* for a suitable *unwinding relation*. An unwinding relation is a binary relation on the states of a state-event system. Intuitively, it captures an indistinguishability relation on states of a state-event system that shall relate all states indistinguishable for an attacker observing only transitions of the state-event system with a visible event. If one can provide an unwinding relation for a state-event system satisfying the two unwinding conditions for a BSP, one can conclude that the state-event system satisfies this BSP.

Overall *MAKS* provides an unwinding theorem for each of the 14 BSPs allowing one to verify a BSP for a state-event system by verifying two unwinding conditions.

I-MAKS Formalization. In the following, we present the unwinding theorem for BSD in *I-MAKS* and the two unwinding conditions *locally-respects forwards* and *output-step consistency* used in this theorem. For this purpose, let SES be a STATE-EVENT SYSTEM and $\mathcal{V}$ be a VIEW ON E_{SES}.

All unwinding conditions only reason about the reachable states of a STATE-EVENT SYSTEM. The predicate `reachable` characterizes theses states in *I-MAKS*.

definition *reachable :: "('s, 'e) SES_rec ⇒ 's ⇒ bool"*
where
"reachable SES s ≡ (∃t. $s0_{SES}$ $t \Longrightarrow_{SES}$ s)"

That is, all states in the image of the big-step transition function are reachable.

Based on this predicate, the unwinding condition locally-respects forwards is formalized in *I-MAKS* utilizing the predicate `lrf` on binary relations of states.

definition *lrf :: "'s rel ⇒ bool"*
where
"lrf ur ≡
∀s ∈ S_{SES}. ∀s' ∈ S_{SES}. ∀c ∈ $C_{\mathcal{V}}$.
((reachable SES s ∧ s $c \longrightarrow_{SES}$ s') ⟶ (s', s) ∈ ur)"

Intuitively, the predicate `lrf` ensures that the states before and after any occurrence of a confidential event are indistinguishable.

Using the predicate `lrf`, *I-MAKS* formalizes the unwinding condition locally-respects forwards as follows.

Definition 34. *Let SES be a* STATE-EVENT SYSTEM*. Let $\mathcal{V}$ be a* VIEW ON E_{SES}*. The unwinding condition* LOCALLY-RESPECTS FORWARDS *is satisfied for a binary relation ur on states of type 's if and only if lrf ur is satisfied.* ◇

The unwinding condition output-step consistency is formalized in *I-MAKS* utilizing the predicate `osc` on binary relations of states.

definition *osc :: "'s rel ⇒ bool"*
where
"osc ur ≡
∀s1 ∈ S_{SES}. ∀s1' ∈ S_{SES}. ∀s2' ∈ S_{SES}. ∀e ∈ (E_{SES} - $C_{\mathcal{V}}$).
(reachable SES s1 ∧ reachable SES s1'
∧ s1' $e \longrightarrow_{SES}$ s2' ∧ (s1', s1) ∈ ur)
⟶ (∃s2 ∈ S_{SES}. ∃δ. δ ↿ $C_{\mathcal{V}}$ = [] ∧ δ ↿ $V_{\mathcal{V}}$ = [e] ↿ $V_{\mathcal{V}}$
∧ s1 $\delta \Longrightarrow_{SES}$ s2 ∧ (s2', s2) ∈ ur)"

Intuitively, the predicate `osc` captures the following requirement (cf. Fig. 4): If any two states `s1'` and `s1` are indistinguishable, then a possible transition with a confidential event `e` from `s1'` to `s2'` has to be matched by a sequence of transitions with a trace `delta` from `s1` to `s2`. Thereby, the trace δ can differ from the trace `[e]` in at most the occurrence of don't care events. Moreover, the states resulting after the transitions have to be indistinguishable.

Using the predicate `osc`, *I-MAKS* formalizes the unwinding condition output-step consistency as follows.

Definition 35. *Let SES be a* STATE-EVENT SYSTEM*. Let $\mathcal{V}$ be a* VIEW ON E_{SES}*. The unwinding condition* OUTPUT-STEP CONSISTENCY *is satisfied for a binary relation ur on states of type 's if and only if osc ur is satisfied.* ◇

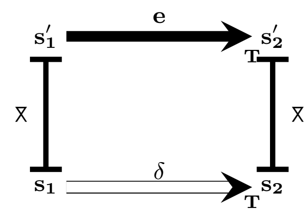

Figure 4: Illustration of the unwinding condition `osc` [Man03].

In the context of BSPs, the unwinding condition `lrf` matches the intuition of D because the observer cannot distinguish states connected by a transition with a confidential event, i.e. he cannot recognize that a confidential event occurred. Moreover, the unwinding condition `osc` matches the intuition of causal corrections in don't care events. That is, because only corrections in don't care event are permitted but are also restricted to future transitions.

Following this intuition, *I-MAKS* provides an unwinding theorem for BSD.

theorem *unwinding_theorem_BSD:*
"⟦ *lrf ur; osc ur* ⟧ $\Longrightarrow$ *BSD* $\mathcal{V}$ *Tr*$_{(inducеES\ SES)}$"

Hence, one can directly conclude BSD for a state-event system after providing an unwinding relation `ur` such that the two unwinding conditions LOCALLY-RESPECTS FORWARDS and OUTPUT-STEP CONSISTENCY are satisfied. That is, if there is a unwinding relation `ur` on the states of SES such that the two unwinding conditions LOCALLY-RESPECTS FORWARDS and OUTPUT-STEP CONSISTENCY are satisfied for `ur`, then the set of possible traces induced by SES satisfies BSD for $\mathcal{V}$.

As mentioned beforehand, we list the 13 remaining unwinding theorems and the unwinding conditions used in these theorems in Appendix A.5. Note that in these unwinding theorems, the unwinding condition OUTPUT-STEP CONSISTENCY is always reused and combined with a suitable variant of locally-respects.

6.2 Compositionality Results

To scale to larger systems, *MAKS* provides support for modular reasoning in the form of compositionality results. The compositionality results allow one to conclude information-flow properties for a composed system from the information-flow properties satisfied by its components.

Overall, *MAKS* provides 11 compositionality results. These compositionality results cover all strict, backwards-strict, and forward-correctable BSPs of *MAKS* as well as the BSP R.

I-MAKS Formalization. In the following, we present the compositionality result for *BSD* in *I-MAKS* and the two conditions for its application: *proper separation of views* and *well-behaved composition.* For this purpose, let ES1 and ES2 be two

EVENT SYSTEMS that are COMPOSABLE and let ES be the PARALLEL COMPOSITION of ES1 and ES2. Furthermore, let $\mathcal{V}1$ be a VIEW ON E_{ES1}, $\mathcal{V}2$ be a VIEW ON E_{ES2}, and $\mathcal{V}$ be a VIEW ON E_{ES}.

The first condition, proper separation of views, ensures that the attacker's perspective on the two components is compatible with the attacker's perspective on the composed system. The predicate `properSeparationOfViews` formalizes this condition in *I-MAKS*.

definition
properSeparationOfViews ::
"'e ES_rec ⇒ 'e ES_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ bool"
where
"properSeparationOfViews ES1 ES2 $\mathcal{V}$ $\mathcal{V}1$ $\mathcal{V}2$ ≡

$$
\begin{aligned}
& V_{\mathcal{V}} \cap E_{ES1} = V_{\mathcal{V}1} \\
& \wedge\ V_{\mathcal{V}} \cap E_{ES2} = V_{\mathcal{V}2} \\
& \wedge\ C_{\mathcal{V}} \cap E_{ES1} \subseteq C_{\mathcal{V}1} \\
& \wedge\ C_{\mathcal{V}} \cap E_{ES2} \subseteq C_{\mathcal{V}2} \\
& \wedge\ N_{\mathcal{V}1} \cap N_{\mathcal{V}2} = \{\}\text{"}
\end{aligned}
$$

The predicate ensures that the attacker's perspective on the two components and the attacker's perspective on the composed system are *compatible* in the following sense: Firstly, the view on the composed system does not change the attacker's capabilities on observing the components, i.e. everything the attacker is able to observe for the components he can also observe for the composed system (captured by the first two conjuncts). Secondly, the view on the composed system does not consider events confidential that are not confidential for the components (captured by the second two conjuncts). Finally, the last conjunct ensures that a correction in don't care events only changes don't care events in one component.

Using the predicate `properSeparationOfViews`, *I-MAKS* captures the condition proper separation of views as follows.

Definition 36. *The views $\mathcal{V}1$ and $\mathcal{V}2$ constitute a* PROPER SEPARATION OF $\mathcal{V}$ *for the* EVENT SYSTEMS *ES1 and ES2 if and only if* `properSeparationOfViews ES1 ES2` $\mathcal{V}$ $\mathcal{V}1$ $\mathcal{V}2$ *is satisfied.* ◇

The second condition, well-behaved composition, ensures that corrections in one component affecting the shared events can be handled by the other component. The predicate `wellBehavedComposition` formalizes this condition in *I-MAKS*.

definition
wellBehavedComposition ::
"'e ES_rec ⇒ 'e ES_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ bool"
where
"wellBehavedComposition ES1 ES2 $\mathcal{V}$ $\mathcal{V}1$ $\mathcal{V}2$ ≡

$$
\begin{aligned}
& (\ N_{\mathcal{V}1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap E_{ES1} = \{\}\ ) \\
& \quad \vee\ (\exists \varrho 1.\ (\ N_{\mathcal{V}1} \cap E_{ES2} = \{\} \wedge \mathit{total}\ ES1\ (C_{\mathcal{V}1} \cap N_{\mathcal{V}2}) \\
& \qquad\qquad \wedge\ BSIA\ \varrho 1\ \mathcal{V}1\ Tr_{ES1}\ )) \\
& \quad \vee\ (\exists \varrho 2.\ (\ N_{\mathcal{V}2} \cap E_{ES1} = \{\} \wedge \mathit{total}\ ES2\ (C_{\mathcal{V}2} \cap N_{\mathcal{V}1}) \\
& \qquad\qquad \wedge\ BSIA\ \varrho 2\ \mathcal{V}2\ Tr_{ES2}\ ))
\end{aligned}
$$

$\vee$ $(\exists\, \varrho 1\ \varrho 2\ \Gamma 1\ \Gamma 2.\ ($
$\nabla_{\Gamma 1} \subseteq E_{ES1} \wedge \Delta_{\Gamma 1} \subseteq E_{ES1} \wedge \Upsilon_{\Gamma 1} \subseteq E_{ES1}$
$\wedge\ \nabla_{\Gamma 2} \subseteq E_{ES2} \wedge \Delta_{\Gamma 2} \subseteq E_{ES2} \wedge \Upsilon_{\Gamma 2} \subseteq E_{ES2}$
$\wedge$ BSIA $\varrho 1\ \mathcal{V}1\ Tr_{ES1} \wedge$ BSIA $\varrho 2\ \mathcal{V}2\ Tr_{ES2}$
$\wedge$ 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 $\varrho 1\ \Gamma 1\ \mathcal{V}1\ Tr_{ES1} \wedge$ FCIA $\varrho 2\ \Gamma 2\ \mathcal{V}2\ Tr_{ES2}$
$\wedge\ V_{\mathcal{V}1} \cap V_{\mathcal{V}2} \subseteq \nabla_{\Gamma 1} \cup \nabla_{\Gamma 2}$
$\wedge\ C_{\mathcal{V}1} \cap N_{\mathcal{V}2} \subseteq \Upsilon_{\Gamma 1} \wedge C_{\mathcal{V}2} \cap N_{\mathcal{V}1} \subseteq \Upsilon_{\Gamma 2}$
$\wedge\ N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\}\ ))$"

definition *total :: "'e ES_rec ⇒ 'e set ⇒ bool"*
where
$"total\ ES\ E \equiv E \subseteq E_{ES} \wedge (\forall \tau \in Tr_{ES}.\ \forall e \in E.\ \tau\ @\ [e] \in Tr_{ES})"$

The four disjuncts of the predicate `wellBehavedComposition` establish a case distinction on the truth values of $N_{\mathcal{V}1} \cap E_{ES2} = \{\}$ and $N_{\mathcal{V}2} \cap E_{ES1} = \{\}$. If $N_{\mathcal{V}1} \cap E_{ES2} = \{\}$ holds, no shared events are affected by corrections in the first component. Likewise, if $N_{\mathcal{V}2} \cap E_{ES1} = \{\}$ holds, no shared events are affected by corrections in the second component. Hence, the first disjunct ensures that corrections do not have an effect on other components. The second disjunct ensures that corrections in the second component can be handled without leaking information about secrets by the first component. In the other direction, no shared events are affected by corrections. The third disjunct is the counterpart to the second disjunct for the opposite direction. Finally, the fourth disjunct ensures that corrections in either component can be handled by the components without leaking information about secrets.

Using the predicate `wellBehavedComposition`, *I-MAKS* captures the condition well-behaved composition as follows.

Definition 37. *Suppose that $\mathcal{V}1$ and $\mathcal{V}2$ constitute a* PROPER SEPARATION OF $\mathcal{V}$ *for the* EVENT SYSTEMS *ES1 and ES2. The composition of ES1 and ES2 is a* WELL-BEHAVED COMPOSITION *wrt. $\mathcal{V}1$ and $\mathcal{V}2$ if and only if `wellBehavedComposition` ES1 ES2 $\mathcal{V}$ $\mathcal{V}1$ $\mathcal{V}2$ is satisfied.* ◇

Assuming the two conditions proper separation of views and well-behaved composition, the compositionality result for *BSD* is formalized by the following theorem proven in *I-MAKS*.[3]

theorem *compositionality_BSD:*
$"[\![\ BSD\ \mathcal{V}1\ Tr_{ES1};\ BSD\ \mathcal{V}2\ Tr_{ES2}\ ]\!] \Longrightarrow BSD\ \mathcal{V}\ Tr_{(ES1\ \|\ ES2)}"$

That is, the verification of *BSD* for the composed system ES can be reduced to the verification of *BSD* for the components.

We present the remaining 10 compositionality results formalized and proven in *I-MAKS* in Appendix A.6. For all of these compositionality results, it is assumed that $\mathcal{V}1$ and $\mathcal{V}2$ constitute a PROPER SEPARATION OF $\mathcal{V}$ for ES1 and ES2 as well as that the composition of ES1 and ES2 is a WELL-BEHAVED COMPOSITION wrt. $\mathcal{V}1$ and $\mathcal{V}2$. With these results, *I-MAKS* covers all compositionality results from *MAKS*.

[3] The two conditions proper separation of views and well-behaved composition are not directly stated in the theorem, instead the conditions are required by the context of the theorem in Isabelle/HOL.

7 Related Work

In the area of information-flow security, one distinguishes between language-based information-flow security and information-flow security at the specification level. In the former the security of programs is investigated. In the latter the security of systems modeled at a higher level of abstraction than programs is investigated. In this report we focus on information-flow security at the specification level. We refer to [SM03] for an overview on language-based information-flow security.

Frameworks for Specification-Level Information-Flow Security. Besides *MAKS*, there are a couple of other frameworks that were developed for analyzing and comparing different possibilistic information-flow properties.

The framework of selective interleaving functions is proposed in [McL94], the process algebra SPA in [FG95] and a representation of information-flow properties based on low-level equivalence sets in [ZL97]. More recently, three frameworks supporting the specification of possibilistic information-flow properties were developed inspired by *MAKS*: A general schema for the specification of trace-based information-flow properties is presented in [SS09]. A similar schema for the specification of information-flow properties of programs with UTP semantics [HH98] is provided in [BJ10]. Finally, a variant of *MAKS* supporting non-terminating systems is presented in [MC12]. However, we are not aware of any formalization of these frameworks in a proof assistant such as Isabelle/HOL.

The probably closest work to ours is the Bounded-Deducibility Security (BD-Security) framework [PL14] also formalized in Isabelle/HOL. The framework enables the specification of possibilistic information-flow properties (incl. declassification) for I/O automata. In contrast to *I-MAKS*, it does not provide building blocks for the definition of custom information-flow properties.

Verification of Information-Flow Properties using Theorem Provers. Interactive theorem provers have been applied in the verification of information-flow properties in several case studies. For instance, in [ACL03] Coq and in [vOLW05] Isabelle/HOL are used for the verification of information-flow properties expressing memory isolation on smartcards. In [KSBR13] KIV has been used for the verification of information-flow properties, e.g. intransitive noninterference [vdM07]. More recently, utilizing the BD-Security framework Isabelle/HOL has been used for the verification of information-flow properties for an conference management system [KLP14] and for a social-media platform [BPPR16, BPPR17]. While *I-MAKS* has not been used in comparable case studies with Isabelle/HOL, we are confident that it can be used for similar case studies in the future.

Tools for the Verification of Information-Flow Properties. With *I-MAKS* we enable users to verify possibilistic information-flow properties in Isabelle/HOL. Related to this general support for the verification of information-flow properties are special purpose tools that permit the verification of specific information-flow properties for specific system specification languages.

There exist several tools in this direction. For process algebras there are, for instance, the *Checker of Persistent Security (CoPS)* [PPR04] targeting the process algebra SPA for the properties SBNDC, PBNDC, and PPBNDS, the *Pi-calculus Non-interference checker (PicNIc)* [CMM+08] to verify four information-flow properties for the Pi-calculus, aiming in particular at controlled declassification, and the CSP refinement checker *FDR2* [For10] for properties based on the idea of low-determinism. The *Petri Net Security Checker* [FGF09] verifies the property PBNI+ for Petri Net specifications. The *Automated Non-Interference Check Assistant (Anica)* [Leh11] targets the verification of PBNI+ and PBNID for Petri Net specifications. The UMLsec-Tool and its successor CARiSMA [WWB+13] can verify UML specifications with respect to certain information-flow properties including some *MAKS* BSPs. These tools enable an automatic verification of the specific information-flow properties, but they do not provide the freedom to specify and verify custom information-flow properties.

8 Conclusion

We presented *I-MAKS*, an Isabelle/HOL formalization of *MAKS*. *I-MAKS* transfers the pen-and-paper framework into the proof assistant Isabelle/HOL. With this transfer, we reverified the soundness of the pen-and-paper framework utilizing the machine-supported rigor of the proof assistant Isabelle/HOL. In addition this transfer, enables the usage of the general purpose proof techniques offered by Isabelle/HOL when using *I-MAKS*.

We see *I-MAKS* as step towards developing a tool for the specification and verification of possibilistic information-flow properties at the specification level. That is, *I-MAKS* provides the necessary basis to develop front-ends for the specification of system models in common specification languages and adding further support for (semi-)automatic verification of information-flow properties. For instance, the development of a CSP front-end and the integration of the model-checking techniques from [DHRS11] are interesting future directions. In its current form *I-MAKS* already allows one to use the machine-checked framework as a tool in Isabelle/HOL.

We hope that *I-MAKS* also encourages the development of further machine-checked extensions of *MAKS* integrating already existing extensions (e.g. [HS04, MC12]) or adding novel extensions to the framework.

Acknowledgments

The authors thank Christoph Feller from the University of Kaiserslautern for his valuable feedback regarding the usage of a preliminary version of *I-MAKS* and Oliver Bračevac for his contributions to the proofs in the theories related to unwinding and compositionality.

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

[ACL03] J. Andronick, B. Chetali, and O. Ly. Using Coq to Verify Java Card Applet Isolation Properties. In *Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, LNCS 2758, pages 335–351, 2003.

[BFPR03] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Refinement Operators and Information Flow Security. In *Proceedings of the International Conference on Software Engineering and Formal Methods (SEFM)*, pages 44–53, 2003.

[BJ10] M. J. Banks and J. L. Jacob. Unifying Theories of Confidentiality. In *Proceedings of the Third International Symposium on Unifying Theories of Programming (UTP)*, LNCS 6445, pages 120–136, 2010.

[BPPR16] T. Bauereiß, A. Pesenti Gritti, A. Popescu, and F. Raimondi. CoSMed: A Confidentiality-Verified Social Media Platform. In *Proceedings of the 7th International Conference on Interactive Theorem Proving*, pages 87–106, 2016.

[BPPR17] T. Bauereiß, A. Pesenti Gritti, A. Popescu, and F. Raimondi. CoSMeDis: A Distributed Social Media Platform with Formally Verified Confidentiality Guarantees. In *Proceedings of the 2017 IEEE Symposium on Security and Privacy*, pages 729–748, 2017.

[CMM+08] S. Crafa, M. Mio, M. Miculan, C. Piazza, and S. Rossi. PicNIc - Pi-Calculus Non-Interference Checker. In *International Conference on Application of Concurrency to System Design (ACSD)*, pages 33–38, 2008.

[DHRS11] D. D'Souza, R. Holla, R. K. Ramesh, and B. Sprick. Model-Checking Trace-Based Information Flow Properties. *Journal of Computer Security*, 19(1):101–138, 2011.

[FG95] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[FGF09] S. Frau, R. Gorrieri, and C. Ferigato. Petri Net Security Checker: Structural Non-interference at Work. In *Workshop on Formal Aspects in Security and Trust (FAST'08)*, LNCS 5491, pages 210–225, 2009.

[For10] Formal Systems (Europe) Ltd and Oxford University Computing Laboratory. *FDR2 User Manual*, 2010.

[GM82] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proceedings of the 3rd IEEE Symposium on Security and Privacy (S&P)*, pages 11–20, 1982.

[Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[HH98] C.A.R. Hoare and J. He. *Unifying Theories of Programming*. Prentice Hall, 1998.

[HS04] D. Hutter and A. Schairer. Possibilistic Information Flow Control in the Presence of Encrypted Communication. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, LNCS 3193, pages 209–224, 2004.

[JT88] D. M. Johnson and F. J. Thayer. Security and the Composition of Machines. In *Proceedings of the Computer Security Foundations Workshop*, pages 72–89, 1988.

[KLP14] S. Kanav, P. Lammich, and A. Popescu. A Conference Management System with Verified Document Confidentiality. In *Proceedings of the 26th International Conference on Computer Aided Verification*, pages 167–183, 2014.

[KSBR13] K. Katkalov, K. Stenzel, M. Borek, and W. Reif. Model-Driven Development of Information Flow-Secure Systems with IFlow. In *Proceedings of the 5th ASE/IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*, 2013.

[Leh11] A. Lehmann. Automated Non-Interference Check Assistant (Anica). `http://service-technology.org/anica`, September 2011.

[Man00a] H. Mantel. Possibilistic Definitions of Security – An Assembly Kit. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 185–199, 2000.

[Man00b] H. Mantel. Unwinding Possibilistic Security Properties. In *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS)*, LNCS 1895, pages 238–254, 2000.

[Man02] H. Mantel. On the Composition of Secure Systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 88–104, 2002.

[Man03] H. Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Saarland University, Saarbrücken, Germany, 2003.

[Man11] H. Mantel. Information Flow and Noninterference. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 605–607. Springer, 2011.

[MC12] D. Milushev and D. Clarke. Coinductive Unwinding of Security-Relevant Hyperproperties. In *Proceedings of the 17th Nordic Conference on Secure IT Systems (NordSec)*, LNCS 7617, pages 121–136, 2012.

[McC87] D. McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In *Proceedings of the 8th IEEE Symposium on Security and Privacy (S&P)*, pages 161–166, 1987.

[McL94] J. D. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy (S&P)*, pages 79–93, 1994.

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. LNCS 2283. Springer, 2002.

[O'H90] C. O'Halloran. A Calculus of Information Flow. In *Proceedings of the 1st European Symposium on Research in Computer Security (ESORICS)*, pages 147–159, 1990.

[PL14] A. Popescu and P. Lammich. Bounded-Deducibility Security. *Archive of Formal Proofs*, 2014.

[PPR04] C. Piazza, E. Pivato, and S. Rossi. CoPS - Checker of Persistent Security. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 2988, pages 144–152, 2004.

[PWK96] R. V. Peri, W. A. Wulf, and D. M. Kienzle. A Logic of Composition for Information Flow Predicates. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–93, 1996.

[SM03] A. Sabelfeld and A. C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[SS09] F. Seehusen and K. Stolen. Information Flow Security, Abstraction and Composition. *Information Security, IET*, 3(1):9–33, 2009.

[Sut86] D. Sutherland. A Model of Information. In *Proceedings of the 9th National Computer Security Conference*, Baltimore, MD, USA, 1986.

[UT18a] University of Cambridge and Technische Universität München. Isabelle Documentation. `http://isabelle.in.tum.de/documentation.html`, April 2018.

[UT18b] University of Cambridge and Technische Universität München. Theory List. http://isabelle.in.tum.de/library/HOL/HOL/List.html, April 2018.

[vdM07] R. van der Meyden. What, Indeed, is Intransitive Noninterference? (extended abstract). In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, LNCS 4734, pages 235–250, 2007.

[vOLW05] D. von Oheimb, V. Lotz, and G. Walter. Analyzing SLE 88 Memory Management Security using Interacting State Machines. *International Journal of Information Security*, 4(3):155–171, 2005.

[WWB+13] S. Wenzel, D. Warzecha, B. Berghoff, J. Bürger, L. Kaltchev, J. Kowald, K. Mensah, M. Michel, and K. Rudack. CARiSMA. http://vm4a003.itmc.tu-dortmund.de/carisma/web/doku.php, October 2013.

[ZL95] A. Zakinthinos and E. S. Lee. The Composability of Non-Interference. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop (CSFW)*, pages 2–8, 1995.

[ZL97] A. Zakinthinos and E. S. Lee. A General Theory of Security Properties. In *Proceedings of the 18th IEEE Symposium on Security and Privacy (S&P)*, pages 94–102, 1997.

A I-MAKS Definitions and Theorems

A.1 Definitions of Basic Security Predicates

In the following, we provide the definitions of all BSPs as defined in *I-MAKS*, corresponding lemmas proven in *I-MAKS* that state the validity of the BSPs, and necessary supplementary definitions. We extracted all of these definitions and lemmas from the theory `BasicSecurityPredicates`.

```
type_synonym 'e BSP = "'e V_rec ⇒ (('e list) set) ⇒ bool"
```

```
definition BSP_valid :: "'e BSP ⇒ bool"
where
"BSP_valid bsp ≡
  ∀𝒱 Tr E. ( isViewOn 𝒱 E ∧ areTracesOver Tr E )
              ⟶ (∃ Tr'. Tr' ⊇ Tr  ∧ bsp 𝒱 Tr')"
```

Supplementary Definitions. We provide definitions of the types for ρ and Γ that are used as additional parameters in the BSPs `IA`, `SIA`, `BSIA`, `FCD`, `FCI`, and `FCIA` below.

```
type_synonym 'e Rho = "'e V_rec ⇒ 'e set"
```

```
record 'e Gamma =
  Nabla :: "'e set"
  Delta :: "'e set"
  Upsilon :: "'e set"
```

In addition, we provide the definition of ρ-admissibility in *I-MAKS* which is used in the definitions of `IA`, `SIA`, `BSIA`, and `FCIA`.

```
definition
Adm :: "'e V_rec ⇒ 'e Rho ⇒ ('e list) set ⇒ 'e list ⇒ 'e ⇒ bool"
where
"Adm 𝒱 ϱ Tr β e ≡
   ∃γ. ((γ @ [e]) ∈ Tr ∧ γ↿(ϱ 𝒱) = β↿(ϱ 𝒱))"
```

Unrestricted Basic Security Predicates. We provide the definitions of all BSPs that permit arbitrary corrections together with their validity lemmas below.

```
definition R :: "'e BSP"
where
"R 𝒱 Tr ≡
  ∀τ∈Tr. ∃τ'∈Tr. τ' ↿ C_𝒱 = [] ∧ τ' ↿ V_𝒱 = τ ↿ V_𝒱"
```

```
lemma BSP_valid_R: "BSP_valid R"
```

```
definition D :: "'e BSP"
where
"D 𝒱 Tr ≡
  ∀α β. ∀c∈C_𝒱. ((β @ [c] @ α) ∈ Tr ∧ α↿C_𝒱 = [])
    ⟶ (∃α' β'. ((β' @ α') ∈ Tr ∧ α'↿V_𝒱 = α↿V_𝒱 ∧ α'↿C_𝒱 = []
                    ∧ β'↿(V_𝒱 ∪ C_𝒱) = β↿(V_𝒱 ∪ C_𝒱)))"
```

```
lemma BSP_valid_D: "BSP_valid D"

definition I :: "'e BSP"
where
"I 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ α) ∈ Tr ∧ α↿C𝒱 = [])
    ⟶ (∃α' β'. ((β' @ [c] @ α') ∈ Tr ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []
                       ∧ β'↿(V𝒱 ∪ C𝒱) = β↿(V𝒱 ∪ C𝒱)))"

lemma BSP_valid_I: "BSP_valid I"

definition IA :: "'e Rho ⇒ 'e BSP"
where
"IA ϱ 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ α) ∈ Tr ∧ α↿C𝒱 = [] ∧ (Adm 𝒱 ϱ Tr β c))
    ⟶ (∃ α' β'. ((β' @ [c] @ α') ∈ Tr) ∧ α'↿V𝒱 = α↿V𝒱
                      ∧ α'↿C𝒱 = [] ∧ β'↿(V𝒱 ∪ C𝒱) = β↿(V𝒱 ∪ C𝒱))"

lemma BSP_valid_IA: "BSP_valid (IA ϱ) "
```

Strict Basic Security Predicates. We provide the definitions of all BSPs that permit no corrections together with their validity lemmas below.

```
definition SR :: "'e BSP"
where
"SR 𝒱 Tr ≡ ∀τ∈Tr. τ ↿ (V𝒱 ∪ N𝒱) ∈ Tr"

lemma "BSP_valid SR"

definition SD :: "'e BSP"
where
"SD 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ [c] @ α) ∈ Tr ∧ α↿C𝒱 = []) ⟶ β @ α ∈ Tr"

lemma "BSP_valid SD"

definition SI :: "'e BSP"
where
"SI 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ α) ∈ Tr ∧ α ↿ C𝒱 = []) ⟶ β @ [c] @ α ∈ Tr"

lemma "BSP_valid SI"

definition SIA :: "'e Rho ⇒ 'e BSP"
where
"SIA ϱ 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ α) ∈ Tr ∧ α ↿ C𝒱 = [] ∧ (Adm 𝒱 ϱ Tr β c))
    ⟶ (β @ [c] @ α) ∈ Tr"

lemma "BSP_valid (SIA ϱ) "
```

Backwards-Strict Basic Security Predicates. We provide the definitions of all BSPs that permit corrections after the point of perturbation together with their validity lemmas below.

```
definition BSD :: "'e BSP"
where
"BSD 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ [c] @ α) ∈ Tr ∧ α↿C𝒱 = [])
    ⟶ (∃α'. ((β @ α') ∈ Tr ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []))"

lemma BSP_valid_BSD: "BSP_valid BSD"

definition BSI :: "'e BSP"
where
"BSI 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ α) ∈ Tr ∧ α↿C𝒱 = [])
    ⟶ (∃α'. ((β @ [c] @ α') ∈ Tr ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []))"

lemma BSP_valid_BSI: "BSP_valid BSI"

definition BSIA :: "'e Rho ⇒ 'e BSP"
where
"BSIA ϱ 𝒱 Tr ≡
  ∀α β. ∀c∈C𝒱. ((β @ α) ∈ Tr ∧ α↿C𝒱 = [] ∧ (Adm 𝒱 ϱ Tr β c))
    ⟶ (∃α'. ((β @ [c] @ α') ∈ Tr ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []))"

lemma BSP_valid_BSIA: "BSP_valid (BSIA ϱ) "
```

Forward-Correctable Basic Security Predicates. We provide the definitions of all BSPs that permit perturbations only directly before a visible event and permit corrections after the point of perturbation together with their validity lemmas below.

```
definition FCD :: "'e Gamma ⇒ 'e BSP"
where
"FCD Γ 𝒱 Tr ≡
  ∀α β. ∀c∈(C𝒱 ∩ ϒΓ). ∀v∈(V𝒱 ∩ ∇Γ).
    ((β @ [c,v] @ α) ∈ Tr ∧ α ↿ C𝒱 = [])
      ⟶ (∃α'. ∃δ'. (set δ') ⊆ (N𝒱 ∩ ΔΓ)
                      ∧ ((β @ δ' @ [v] @ α') ∈ Tr
                      ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []))"

lemma BSP_valid_FCD: "BSP_valid (FCD Γ)"

definition FCI :: "'e Gamma ⇒ 'e BSP"
where
"FCI Γ 𝒱 Tr ≡
  ∀α β. ∀c∈(C𝒱 ∩ ϒΓ). ∀v∈(V𝒱 ∩ ∇Γ).
    ((β @ [v] @ α) ∈ Tr ∧ α↿C𝒱 = [])
      ⟶ (∃α'. ∃δ'. (set δ') ⊆ (N𝒱 ∩ ΔΓ)
                      ∧ ((β @ [c] @ δ' @ [v] @ α') ∈ Tr
                      ∧ α'↿V𝒱 = α↿V𝒱 ∧ α'↿C𝒱 = []))"
```

lemma *BSP_valid_FCI: "BSP_valid (FCI Γ)"*

definition *FCIA :: "'e Rho $\Rightarrow$ 'e Gamma $\Rightarrow$ 'e BSP"*
where
"FCIA ϱ Γ $\mathcal{V}$ Tr $\equiv$
$\forall\alpha$ β. $\forall c\in(C_{\mathcal{V}} \cap \Upsilon_{\Gamma})$. $\forall v\in(V_{\mathcal{V}} \cap \nabla_{\Gamma})$.
((β @ [v] @ α) $\in$ Tr $\wedge$ $\alpha\upharpoonright C_{\mathcal{V}}$ = [] $\wedge$ (Adm $\mathcal{V}$ ϱ Tr β c))
$\longrightarrow$ ($\exists\alpha'$. $\exists\delta'$. (set δ') $\subseteq$ ($N_{\mathcal{V}} \cap \Delta_{\Gamma}$)
$\wedge$ ((β @ [c] @ δ' @ [v] @ α') $\in$ Tr
$\wedge$ $\alpha'\upharpoonright V_{\mathcal{V}}$ = $\alpha\upharpoonright V_{\mathcal{V}}$ $\wedge$ $\alpha'\upharpoonright C_{\mathcal{V}}$ = []))"

lemma *BSP_valid_FCIA: "BSP_valid (FCIA ϱ Γ) "*

A.2 Taxonomy Results

In the following, we provide a complete list of the taxonomy results for BSPs formalized and proven in *I-MAKS*. We extracted all of theses results from the theory `BSPTaxonomy`. For all of these results, it is assumed `ES_valid ES`, `isViewOn` $\mathcal{V}$ E_{ES}, `isViewOn` $\mathcal{V}_1$ E_{ES}, and `isViewOn` $\mathcal{V}_2$ E_{ES} hold. We declare any further assumptions in the respective sections.

Taxonomy of BSPs in the First Dimension.

Taxonomy Results for the Same View:

lemma *D_implies_R:*
"D $\mathcal{V}$ Tr_{ES} $\Longrightarrow$ R $\mathcal{V}$ Tr_{ES}"

lemma *BSD_implies_D:*
"BSD $\mathcal{V}$ Tr_{ES} $\Longrightarrow$ D $\mathcal{V}$ Tr_{ES}"

lemma *SD_implies_BSD :*
"(SD $\mathcal{V}$ Tr_{ES}) $\Longrightarrow$ BSD $\mathcal{V}$ Tr_{ES} "

lemma *SD_implies_SR:*
"SD $\mathcal{V}$ Tr_{ES} $\Longrightarrow$ SR $\mathcal{V}$ Tr_{ES}"

Taxonomy Results for Modified Views: For the following taxonomy results it is assumed that for the two views $\mathcal{V}_1$ and $\mathcal{V}_2$, we have that $V_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1}$, $N_{\mathcal{V}_2} \supseteq N_{\mathcal{V}_1}$, and $C_{\mathcal{V}_2} \subseteq C_{\mathcal{V}_1}$ hold.

lemma *R_implies_R_for_modified_view:*
"R $\mathcal{V}_1$ Tr_{ES} $\Longrightarrow$ R $\mathcal{V}_2$ Tr_{ES}"

lemma *D_implies_D_for_modified_view:*
"D $\mathcal{V}_1$ Tr_{ES} $\Longrightarrow$ D $\mathcal{V}_2$ Tr_{ES}"

lemma *BSD_implies_BSD_for_modified_view:*
"BSD $\mathcal{V}_1$ $Tr_{ES}$$\Longrightarrow$ BSD $\mathcal{V}_2$ Tr_{ES}"

lemma *SD_implies_FCD:*
"(SD $\mathcal{V}$ Tr_{ES}) $\Longrightarrow$ FCD Γ $\mathcal{V}$ Tr_{ES}"

Further Taxonomy Results: For the following taxonomy results it is assumed that for the two views $\mathcal{V}_1$ and $\mathcal{V}_2$, we have that $V_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1}$, $N_{\mathcal{V}_2} \supseteq N_{\mathcal{V}_1}$, and $C_{\mathcal{V}_2} = C_{\mathcal{V}_1}$ hold. Furthermore, is assumed that for Γ_1 and Γ_2, we have that $V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}$, $C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}$, and $N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}$ hold.

lemma *FCD_implies_FCD_for_modified_view_gamma:*
"$\llbracket FCD\ \Gamma_1\ \mathcal{V}_1\ Tr_{ES};$
$V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1};\ N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1};\ C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1} \rrbracket$
$\Longrightarrow FCD\ \Gamma_2\ \mathcal{V}_2\ Tr_{ES}$"

Trivial Satisfaction Results:

lemma *Trivially_fulfilled_D_C_empty:*
"$C_{\mathcal{V}} = \{\} \Longrightarrow D\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_BSD_C_empty:*
"$C_{\mathcal{V}} = \{\} \Longrightarrow BSD\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_R_C_empty:*
"$C_{\mathcal{V}} = \{\} \Longrightarrow R\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_SD_C_empty:*
"$C_{\mathcal{V}} = \{\} \Longrightarrow SD\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_FCD_C_empty:*
"$C_{\mathcal{V}} = \{\} \Longrightarrow FCD\ \Gamma\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fullfilled_R_V_empty:*
"$V_{\mathcal{V}}=\{\} \Longrightarrow R\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_D_V_empty:*
"$V_{\mathcal{V}} = \{\} \Longrightarrow D\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_BSD_V_empty:*
"$V_{\mathcal{V}} = \{\} \Longrightarrow BSD\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_FCD_V_empty:*
"$V_{\mathcal{V}} = \{\} \Longrightarrow FCD\ \Gamma\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_FCD_Nabla_Υ_empty:*
"$\llbracket \nabla_{\Gamma}=\{\} \vee \Upsilon_{\Gamma}=\{\} \rrbracket \Longrightarrow FCD\ \Gamma\ \mathcal{V}\ Tr_{ES}$"

lemma *Trivially_fulfilled_FCD_N_subseteq_Δ_and_BSD:*
"$\llbracket N_{\mathcal{V}} \subseteq \Delta_{\Gamma};\ BSD\ \mathcal{V}\ Tr_{ES} \rrbracket \Longrightarrow FCD\ \Gamma\ \mathcal{V}\ Tr_{ES}$"

Taxonomy of BSPs in the Second Dimension.

Taxonomy Results for the Same View:

lemma *SI_implies_BSI :*
$"(SI\ \mathcal{V}\ Tr_{ES}) \Longrightarrow BSI\ \mathcal{V}\ Tr_{ES}\ "$

lemma *BSI_implies_I:*
$"(BSI\ \mathcal{V}\ Tr_{ES}) \Longrightarrow (I\ \mathcal{V}\ Tr_{ES})"$

lemma *SIA_implies_BSIA:*
$"(SIA\ \varrho\ \mathcal{V}\ Tr_{ES}) \Longrightarrow (BSIA\ \varrho\ \mathcal{V}\ Tr_{ES})"$

lemma *SI_implies_SIA:*
$"SI\ \mathcal{V}\ Tr_{ES} \Longrightarrow SIA\ \varrho\ \mathcal{V}\ Tr_{ES}"$

lemma *BSI_implies_BSIA:*
$"BSI\ \mathcal{V}\ Tr_{ES} \Longrightarrow BSIA\ \varrho\ \mathcal{V}\ Tr_{ES}"$

lemma *I_implies_IA:*
$"I\ \mathcal{V}\ Tr_{ES} \Longrightarrow IA\ \varrho\ \mathcal{V}\ Tr_{ES}"$

Taxonomy Results for Modified Views: For the following taxonomy results it is assumed that for the two views $\mathcal{V}_1$ and $\mathcal{V}_2$, we have that $V_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1}$, $N_{\mathcal{V}_2} \supseteq N_{\mathcal{V}_1}$, and $C_{\mathcal{V}_2} = C_{\mathcal{V}_1}$ hold.

lemma *I_implies_I_for_modified_view :*
$"I\ \mathcal{V}_1\ Tr_{ES} \Longrightarrow I\ \mathcal{V}_2\ Tr_{ES}"$

lemma *BSI_implies_BSI_for_modified_view :*
$"BSI\ \mathcal{V}_1\ Tr_{ES} \Longrightarrow BSI\ \mathcal{V}_2\ Tr_{ES}"$

lemma *SI_implies_SI_for_modified_view :*
$"SI\ \mathcal{V}_1\ Tr_{ES} \Longrightarrow SI\ \mathcal{V}_2\ Tr_{ES}"$

lemma *IA_implies_IA_for_modified_view :*
$"[\![IA\ \varrho_1\ \mathcal{V}_1\ Tr_{ES};\ \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1)\ ]\!] \Longrightarrow IA\ \varrho_2\ \mathcal{V}_2\ Tr_{ES}"$

lemma *BSIA_implies_BSIA_for_modified_view :*
$"[\![BSIA\ \varrho_1\ \mathcal{V}_1\ Tr_{ES};\ \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1)\ ]\!] \Longrightarrow BSIA\ \varrho_2\ \mathcal{V}_2\ Tr_{ES}"$

lemma *SIA_implies_SIA_for_modified_view :*
$"[\![SIA\ \varrho_1\ \mathcal{V}_1\ Tr_{ES};\ \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1)\ ]\!] \Longrightarrow SIA\ \varrho_2\ \mathcal{V}_2\ Tr_{ES}"$

Further Taxonomy Results: For the following taxonomy results it is assumed that for the two views $\mathcal{V}_1$ and $\mathcal{V}_2$, we have that $V_{\mathcal{V}_2} \subseteq V_{\mathcal{V}_1}$, $N_{\mathcal{V}_2} \supseteq N_{\mathcal{V}_1}$, and $C_{\mathcal{V}_2} = C_{\mathcal{V}_1}$ hold. Furthermore, is assumed that for Γ_1 and Γ_2, we have that $V_{\mathcal{V}_2} \cap \nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1} \cap \nabla_{\Gamma_1}$, $C_{\mathcal{V}_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1} \cap \Upsilon_{\Gamma_1}$, and $N_{\mathcal{V}_2} \cap \Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1} \cap \Delta_{\Gamma_1}$ hold.

lemma *FCI_implies_FCI_for_modified_view_gamma:*
$$"[\![FCI\ \Gamma_1\ \mathcal{V}_1\ Tr_{ES};\ V_{\mathcal{V}_2}\cap\nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1}\cap\nabla_{\Gamma_1};\ N_{\mathcal{V}_2}\cap\Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1}\cap\Delta_{\Gamma_1};\ C_{\mathcal{V}_2}\cap\Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1}\cap\Upsilon_{\Gamma_1}\ ]\!] \Longrightarrow FCI\ \Gamma_2\ \mathcal{V}_2\ Tr_{ES}"$$

lemma *FCIA_implies_FCIA_for_modified_view_rho_gamma:*
$$"[\![FCIA\ \varrho_1\ \Gamma_1\ \mathcal{V}_1\ Tr_{ES};\ \varrho_2(\mathcal{V}_2) \supseteq \varrho_1(\mathcal{V}_1);\ V_{\mathcal{V}_2}\cap\nabla_{\Gamma_2} \subseteq V_{\mathcal{V}_1}\cap\nabla_{\Gamma_1};\ N_{\mathcal{V}_2}\cap\Delta_{\Gamma_2} \supseteq N_{\mathcal{V}_1}\cap\Delta_{\Gamma_1};\ C_{\mathcal{V}_2}\cap\Upsilon_{\Gamma_2} \subseteq C_{\mathcal{V}_1}\cap\Upsilon_{\Gamma_1}\ ]\!] \Longrightarrow FCIA\ \varrho_2\ \Gamma_2\ \mathcal{V}_2\ Tr_{ES}"$$

Trivial Satisfaction Results:

lemma *Trivially_fulfilled_I_C_empty:*
"$C_{\mathcal{V}}$ = {} $\Longrightarrow$ I $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_IA_C_empty:*
"$C_{\mathcal{V}}$ = {} $\Longrightarrow$ IA ϱ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_BSI_C_empty:*
"$C_{\mathcal{V}}$ = {} $\Longrightarrow$ BSI $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_BSIA_C_empty:*
"$C_{\mathcal{V}}$ = {} $\Longrightarrow$ BSIA ϱ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_SI_C_empty:*
"$C_{\mathcal{V}}$ = {} $\Longrightarrow$ SI $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_SIA_C_empty:*
"$C_{\mathcal{V}}$ = {} $\Longrightarrow$ SIA ϱ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_FCI_C_empty:*
"$C_{\mathcal{V}}$ = {} $\Longrightarrow$ FCI Γ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_FCIA_C_empty:*
"$C_{\mathcal{V}}$ = {} $\Longrightarrow$ FCIA Γ ϱ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_FCI_V_empty:*
"$V_{\mathcal{V}}$ = {} $\Longrightarrow$ FCI Γ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_FCIA_V_empty:*
"$V_{\mathcal{V}}$ = {} $\Longrightarrow$ FCIA ϱ Γ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_IA_V_empty_rho_subseteq_C_N:*
"$[\![$ $V_{\mathcal{V}}$ = {}; ϱ $\mathcal{V}$ $\supseteq$ ($C_{\mathcal{V}}$ $\cup$ $N_{\mathcal{V}}$) $]\!]$ $\Longrightarrow$ IA ϱ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_BSIA_V_empty_rho_subseteq_C_N:*
"$[\![$ $V_{\mathcal{V}}$ = {}; ϱ $\mathcal{V}$ $\supseteq$ ($C_{\mathcal{V}}$ $\cup$ $N_{\mathcal{V}}$) $]\!]$ $\Longrightarrow$ BSIA ϱ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_BSI_V_empty_total_ES_C:*
"$[\![$ $V_{\mathcal{V}}$ = {}; total ES $C_{\mathcal{V}}$ $]\!]$ $\Longrightarrow$ BSI $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_I_V_empty_total_ES_C:*
"$[\![$ $V_{\mathcal{V}}$ = {}; total ES $C_{\mathcal{V}}$ $]\!]$ $\Longrightarrow$ I $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_FCI_Nabla_Υ_empty:*
"$[\![$ ∇_{Γ}={} $\vee$ Υ_{Γ}={} $]\!]$ $\Longrightarrow$ FCI Γ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_FCIA_Nabla_Υ_empty:*
"$[\![$ ∇_{Γ}={} $\vee$ Υ_{Γ}={} $]\!]$ $\Longrightarrow$ FCIA ϱ Γ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_FCI_N_subseteq_Δ_and_BSI:*
"$[\![$ $N_{\mathcal{V}}$ $\subseteq$ Δ_{Γ}; BSI $\mathcal{V}$ Tr_{ES} $]\!]$ $\Longrightarrow$ FCI Γ $\mathcal{V}$ Tr_{ES}"

lemma *Trivially_fulfilled_FCIA_N_subseteq_Δ_and_BSIA:*
"$[\![$ $N_{\mathcal{V}}$ $\subseteq$ Δ_{Γ}; BSIA ϱ $\mathcal{V}$ Tr_{ES} $]\!]$ $\Longrightarrow$ FCIA ϱ Γ $\mathcal{V}$ Tr_{ES}"

A.3 Information-Flow Properties

In the following, we provide the complete definition of the notion of information-flow properties as defined in *I-MAKS*. We extracted all of these definitions from the theory `InformationFlowProperties`.

```
type_synonym 'e SP = "('e BSP) set"

type_synonym 'e IFP_type = "('e V_rec set) × 'e SP"

definition IFP_valid :: "'e set ⇒ 'e IFP_type ⇒ bool"
where
"IFP_valid E ifp ≡
  ∀𝒱 ∈ (fst ifp). isViewOn 𝒱 E
                  ∧ (∀BSP ∈ (snd ifp). BSP_valid BSP)"

definition IFPIsSatisfied :: "'e IFP_type ⇒ ('e list) set  ⇒ bool"
where
"IFPIsSatisfied ifp Tr ≡
  ∀ 𝒱∈(fst ifp). ∀ BSP∈(snd ifp). BSP 𝒱 Tr"
```

A.4 Property Library

In the following, we provide the definitions of information-flow properties from the literature that can be expressed using *I-MAKS* together with their representation in *I-MAKS*. We extracted all of the definitions from the theory `PropertyLibrary`.

Supplementary Definitions. In the definitions that we present in the following, we use the views `HighConfidential` and `HighInputsConfidential` as well as the definition of all interleavings of two traces.

```
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)}"
```

Generalized Noninterference.

```
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)"
```

```
definition litGNI :: "'e set ⇒ 'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool"
where
"litGNI L H IE Tr ≡
  ∀ t1 t2 t3.
    t1 @ t2 ∈ Tr ∧ t3 ↿ (L ∪ (H - IE)) = t2 ↿ (L ∪ (H - IE))
     ⟶ (∃ t4. t1 @ t4 ∈ Tr ∧ t4↿(L ∪ (H ∩ IE)) = t3↿(L ∪ (H ∩ IE)))"
```

Interleaving-based Generalized Noninterference.

```
definition IBGNI :: "'e set ⇒ 'e set ⇒ 'e set ⇒ 'e IFP_type"
where "IBGNI L H IE ≡ ( {HighInputsConfidential L H IE}, {D, I})"
```

```
lemma IBGNI_valid: "L ∩ H = {} ⟹ IFP_valid (L ∪ H) (IBGNI L H IE)"
```

```
definition
litIBGNI :: "'e set ⇒ 'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool"
where
"litIBGNI L H IE Tr ≡
  ∀ τ_l ∈ Tr. ∀ t_hi t.
    (set t_hi) ⊆ (H ∩ IE)  ∧ t ∈ interleaving t_hi (τ_l ↿ L)
      ⟶ (∃ τ' ∈ Tr. τ' ↿ (L ∪ (H ∩ IE)) = t)"
```

Forward Correctability.

```
definition FC :: "'e set ⇒ 'e set ⇒ 'e set ⇒ 'e IFP_type"
where
"FC L H IE ≡
  ( {HighInputsConfidential L H IE},
  {BSD, BSI, (FCD ⦇ Nabla=IE, Delta={}, Upsilon=IE ⦈),
             (FCI ⦇ Nabla=IE, Delta={}, Upsilon=IE ⦈ )})"
```

```
lemma FC_valid: "L ∩ H = {} ⟹ IFP_valid (L ∪ H) (FC L H IE)"
```

```
definition litFC :: "'e set ⇒ 'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool"
where
"litFC L H IE Tr ≡
  ∀t_1 t_2. ∀ hi ∈ (H ∩ IE).
  (
    (∀ li ∈ (L ∩ IE).
      t_1 @ [li] @ t_2 ∈ Tr ∧ t_2 ↿ (H ∩ IE) = []
      ⟶ (∃ t_3. t_1 @ [hi] @ [li] @ t_3 ∈ Tr
                 ∧ t_3 ↿ L = t_2 ↿ L ∧ t_3 ↿ (H ∩ IE) = [] ))
```

```
      ∧ (t_1 @ t_2 ∈ Tr ∧ t_2 ↿ (H ∩ IE) = []
         ⟶ (∃ t_3. t_1 @ [hi]  @ t_3 ∈ Tr
                          ∧ t_3 ↿ L = t_2 ↿ L ∧ t_3 ↿ (H ∩ IE) = [] ))
    ∧ (∀ li ∈ (L ∩ IE).
         t_1 @ [hi] @ [li] @ t_2 ∈ Tr ∧ t_2 ↿ (H ∩ IE) = []
          ⟶ (∃ t_3. t_1 @ [li] @ t_3 ∈ Tr
                          ∧ t_3 ↿ L = t_2 ↿ L ∧ t_3 ↿ (H ∩ IE) = [] ))
         ∧ (t_1 @ [hi]  @ t_2 ∈ Tr ∧ t_2 ↿ (H ∩ IE) = []
            ⟶ (∃ t_3. t_1  @ t_3 ∈ Tr
                          ∧ t_3 ↿ L = t_2 ↿ L ∧ t_3 ↿ (H ∩ IE) = [] ))
  )"
```

Nondeducibility for Outputs.

definition `NDO :: "'e set ⇒ 'e set ⇒ 'e set ⇒ 'e IFP_type"`
where
```
"NDO UI L H ≡
  ( {HighConfidential L H}, {BSD, (BSIA (λ 𝒱. C𝒱 ∪ (V𝒱 ∩ UI)))})"
```

lemma `NDO_valid: "L ∩ H = {} ⟹ IFP_valid (L ∪ H) (NDO UI L H)"`

definition `litNDO :: "'e set ⇒ 'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool"`
where
```
"litNDO UI L H Tr ≡
  ∀τ_l ∈ Tr. ∀ τ_hlui ∈ Tr.  ∀ t.
    t↿L = τ_l↿L ∧ t↿(H ∪ (L ∩ UI)) = τ_hlui↿(H ∪ (L ∩ UI)) ⟶ t ∈ Tr"
```

Noninference.

definition `NF :: "'e set ⇒ 'e set ⇒ 'e IFP_type"`
where
```
"NF L H ≡ ( {HighConfidential L H}, {R})"
```

lemma `NF_valid: "L ∩ H = {} ⟹ IFP_valid (L ∪ H) (NF L H)"`

definition `litNF :: "'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool"`
where
```
"litNF L H Tr ≡ ∀τ ∈ Tr. τ ↿ L ∈ Tr"
```

Generalized Noninference.

definition `GNF :: "'e set ⇒ 'e set ⇒ 'e set ⇒ 'e IFP_type"`
where
```
"GNF L H IE ≡ ( {HighInputsConfidential L H IE}, {R})"
```

lemma `GNF_valid: "L ∩ H = {} ⟹ IFP_valid (L ∪ H) (GNF L H IE)"`

definition `litGNF :: "'e set ⇒ 'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool"`
where
```
"litGNF L H IE Tr ≡
  ∀τ ∈ Tr. ∃τ' ∈ Tr. τ'↿ (H ∩ IE) = [] ∧ τ'↿ L = τ ↿ L"
```

Separability.

definition *SEP :: "'e set ⇒ 'e set ⇒ 'e IFP_type"*
where
```
"SEP L H ≡ ( {HighConfidential L H}, {BSD, (BSIA (λ 𝒱. C_𝒱))})"
```

lemma *SEP_valid: "L ∩ H = {} ⟹ IFP_valid (L ∪ H) (SEP L H)"*

definition *litSEP :: "'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool"*
where
```
"litSEP L H Tr ≡
  ∀τ_l ∈ Tr. ∀ τ_h ∈ Tr.
    interleaving (τ_l ↿ L) (τ_h ↿ H) ⊆ {τ ∈ Tr . τ ↿ L = τ_l ↿ L} "
```

Perfect Security Property.

definition *PSP :: "'e set ⇒ 'e set ⇒ 'e IFP_type"*
where
```
"PSP L H ≡
  ( {HighConfidential L H}, {BSD, (BSIA (λ 𝒱. C_𝒱 ∪ N_𝒱 ∪ V_𝒱))})"
```

lemma *PSP_valid: "L ∩ H = {} ⟹ IFP_valid (L ∪ H) (PSP L H)"*

definition *litPSP :: "'e set ⇒ 'e set ⇒ ('e list) set ⇒ bool"*
where
```
"litPSP L H Tr ≡
  (∀τ ∈ Tr. τ ↿ L ∈ Tr)
   ∧ (∀ α β. (β @ α) ∈ Tr ∧ (α ↿ H) = []
              ⟶ (∀ h ∈ H. β @ [h] ∈ Tr ⟶ β @ [h] @ α ∈ Tr))"
```

A.5 Unwinding

In the following, it is assumed that `SES_valid SES` and `isViewOn` $\mathcal{V}$ $\mathtt{E}_{\mathtt{SES}}$ hold.

Unwinding Conditions. In the following, we provide the definition of all unwinding conditions as defined in *I-MAKS*. We extracted all of these definitions from the theory `UnwindingConditions`.

Auxiliary Definitions:

definition *En :: "'e Rho ⇒ 's ⇒ 'e ⇒ bool"*
where
```
"En ϱ s e ≡
  ∃β γ. ∃s' ∈ S_SES. ∃s'' ∈ S_SES.
    s0_SES β⟹_SES s ∧ (γ ↿ (ϱ 𝒱) = β ↿ (ϱ 𝒱))
     ∧ s0_SES γ⟹_SES s' ∧ s' e⟶_SES s''"
```

Locally Respects:

definition `lrf :: "'s rel ⇒ bool"`
where
```
"lrf ur ≡
  ∀s ∈ S_SES. ∀s' ∈ S_SES. ∀c ∈ C_𝒱.
   ((reachable SES s ∧ s c⟶_SES s') ⟶ (s', s) ∈ ur)"
```

definition `lrb :: "'s rel ⇒ bool"`
where
```
"lrb ur ≡ ∀s ∈ S_SES. ∀c ∈ C_𝒱.
   (reachable SES s ⟶ (∃s' ∈ S_SES. (s c⟶_SES s' ∧ ((s, s') ∈ ur))))"
```

definition `fcrf :: "'e Gamma ⇒ 's rel ⇒ bool"`
where
```
"fcrf Γ ur ≡
  ∀c ∈ (C_𝒱 ∩ Υ_Γ). ∀v ∈ (V_𝒱 ∩ ∇_Γ). ∀s ∈ S_SES. ∀s' ∈ S_SES.
     ((reachable SES s ∧ s ([c] @ [v])⟹_SES s')
       ⟶ (∃s'' ∈ S_SES. ∃δ. (∀d ∈ (set δ). d ∈ (N_𝒱 ∩ Δ_Γ)) ∧
              s (δ @ [v])⟹_SES s'' ∧ (s', s'') ∈ ur))"
```

definition `fcrb :: "'e Gamma ⇒ 's rel ⇒ bool"`
where
```
"fcrb Γ ur ≡
  ∀c ∈ (C_𝒱 ∩ Υ_Γ). ∀v ∈ (V_𝒱 ∩ ∇_Γ). ∀s ∈ S_SES. ∀s'' ∈ S_SES.
   ((reachable SES s ∧ s v⟶_SES s'')
     ⟶ (∃s' ∈ S_SES. ∃δ. (∀d ∈ (set δ). d ∈ (N_𝒱 ∩ Δ_Γ)) ∧
           s ([c] @ δ @ [v])⟹_SES s' ∧ (s'', s') ∈ ur))"
```

definition `lrbe :: "'e Rho ⇒ 's rel ⇒ bool"`
where
```
"lrbe ϱ ur ≡
  ∀s ∈ S_SES. ∀c ∈ C_𝒱 .
   ((reachable SES s ∧ (En ϱ s c))
     ⟶ (∃s' ∈ S_SES. (s c⟶_SES s' ∧ (s, s') ∈ ur)))"
```

definition `fcrbe :: "'e Gamma ⇒ 'e Rho ⇒ 's rel ⇒ bool"`
where
```
"fcrbe Γ ϱ ur ≡
  ∀c ∈ (C_𝒱 ∩ Υ_Γ). ∀v ∈ (V_𝒱 ∩ ∇_Γ). ∀s ∈ S_SES. ∀s'' ∈ S_SES.
   ((reachable SES s ∧ s v⟶_SES s'' ∧ (En ϱ s c))
     ⟶ (∃s' ∈ S_SES. ∃δ. (∀d ∈ (set δ). d ∈ (N_𝒱 ∩ Δ_Γ)) ∧
            s ([c] @ δ @ [v])⟹_SES s' ∧ (s'', s') ∈ ur))"
```

Output-Step Consistency:

definition `osc :: "'s rel ⇒ bool"`
where
```
"osc ur ≡
  ∀s1 ∈ S_SES. ∀s1' ∈ S_SES. ∀s2' ∈ S_SES. ∀e ∈ (E_SES - C_𝒱).
     (reachable SES s1 ∧ reachable SES s1'
        ∧ s1' e⟶_SES s2' ∧ (s1', s1) ∈ ur)
     ⟶ (∃s2 ∈ S_SES. ∃δ. δ ↿ C_𝒱 = [] ∧ δ ↿ V_𝒱 = [e] ↿ V_𝒱
           ∧ s1 δ⟹_SES s2 ∧ (s2', s2) ∈ ur)"
```

Unwinding Results. In the following, we provide all unwinding theorems specified and proven in *I-MAKS*. We extracted all of these theorems from the theory `UnwindingResults`.

Unwinding Results for Non-Strict BSPs:

theorem *unwinding_theorem_R:*
"⟦ lrf ur; osc ur ⟧ ⟹ R 𝒱 ($Tr_{(induceES\ SES)}$)"

theorem *unwinding_theorem_D:*
"⟦ lrf ur; osc ur ⟧ ⟹ D 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_I:*
"⟦ lrb ur; osc ur ⟧ ⟹ I 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_IA:*
"⟦ lrbe ϱ ur; osc ur ⟧ ⟹ IA ϱ 𝒱 $Tr_{(induceES\ SES)}$"

Unwinding Results for Strict BSPs:

theorem *unwinding_theorem_SR:*
"⟦ 𝒱′ = ⦇ V = ($V_\mathcal{V}$ ∪ $N_\mathcal{V}$), N = {}, C = $C_\mathcal{V}$ ⦈;
Unwinding.lrf SES 𝒱′ ur; Unwinding.osc SES 𝒱′ ur ⟧
⟹ SR 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_SD:*
"⟦ 𝒱′ = ⦇ V = ($V_\mathcal{V}$ ∪ $N_\mathcal{V}$), N = {}, C = $C_\mathcal{V}$ ⦈;
Unwinding.lrf SES 𝒱′ ur; Unwinding.osc SES 𝒱′ ur ⟧
⟹ SD 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_SI:*
"⟦ 𝒱′ = ⦇ V = ($V_\mathcal{V}$ ∪ $N_\mathcal{V}$), N = {}, C = $C_\mathcal{V}$ ⦈;
Unwinding.lrb SES 𝒱′ ur; Unwinding.osc SES 𝒱′ ur ⟧
⟹ SI 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_SIA:*
"⟦ 𝒱′ = ⦇ V = ($V_\mathcal{V}$ ∪ $N_\mathcal{V}$), N = {}, C = $C_\mathcal{V}$ ⦈; ϱ 𝒱 = ϱ 𝒱′;
Unwinding.lrbe SES 𝒱′ ϱ ur; Unwinding.osc SES 𝒱′ ur ⟧
⟹ SIA ϱ 𝒱 $Tr_{(induceES\ SES)}$"

Unwinding Results for Backwards-Strict and Forward-Correctable BSPs:

theorem *unwinding_theorem_BSD:*
"⟦ lrf ur; osc ur ⟧ ⟹ BSD 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_BSI:*
"⟦ lrb ur; osc ur ⟧ ⟹ BSI 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_BSIA:*
"⟦ lrbe ϱ ur; osc ur ⟧ ⟹ BSIA ϱ 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_FCD:*
"⟦ fcrf Γ ur; osc ur ⟧ ⟹ FCD Γ 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_FCI:*
"⟦ fcrb Γ ur; osc ur ⟧ ⟹ FCI Γ 𝒱 $Tr_{(induceES\ SES)}$"

theorem *unwinding_theorem_FCIA:*
"⟦ fcrbe Γ ϱ ur; osc ur ⟧ ⟹ FCIA ϱ Γ 𝒱 $Tr_{(induceES\ SES)}$"

A.6 Compositionality

Auxiliary Definitions. In the following we provide the Isabelle/HOL definitions of the two predicates `properSeparationOfViews` and `wellBehavedComposition` that are used in the assumptions for the compositionality results. We extracted these definitions from the theory `CompositionBase`.

definition
```
properSeparationOfViews ::
"'e ES_rec ⇒ 'e ES_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ bool"
```
where
```
"properSeparationOfViews ES1 ES2 𝒱 𝒱1 𝒱2 ≡
   V𝒱 ∩ E_ES1 = V𝒱1
   ∧ V𝒱 ∩ E_ES2 = V𝒱2
   ∧ C𝒱 ∩ E_ES1 ⊆ C𝒱1
   ∧ C𝒱 ∩ E_ES2 ⊆ C𝒱2
   ∧ N𝒱1 ∩ N𝒱2 = {}"
```

definition
```
wellBehavedComposition ::
"'e ES_rec ⇒ 'e ES_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ bool"
```
where
```
"wellBehavedComposition ES1 ES2 𝒱 𝒱1 𝒱2 ≡
( N𝒱1 ∩ E_ES2 = {} ∧ N𝒱2 ∩ E_ES1 = {} )
  ∨ (∃ϱ1. ( N𝒱1 ∩ E_ES2 = {} ∧ total ES1 (C𝒱1 ∩ N𝒱2)
            ∧ BSIA ϱ1 𝒱1 Tr_ES1 ))
  ∨ (∃ϱ2. ( N𝒱2 ∩ E_ES1 = {} ∧ total ES2 (C𝒱2 ∩ N𝒱1)
            ∧ BSIA ϱ2 𝒱2 Tr_ES2 ))
  ∨ (∃ϱ1 ϱ2 Γ1 Γ2. (
       ∇Γ1 ⊆ E_ES1 ∧ ΔΓ1 ⊆ E_ES1 ∧ ΥΓ1 ⊆ E_ES1
       ∧ ∇Γ2 ⊆ E_ES2 ∧ ΔΓ2 ⊆ E_ES2 ∧ ΥΓ2 ⊆ E_ES2
       ∧ BSIA ϱ1 𝒱1 Tr_ES1 ∧ BSIA ϱ2 𝒱2 Tr_ES2
       ∧ total ES1 (C𝒱1 ∩ N𝒱2) ∧ total ES2 (C𝒱2 ∩ N𝒱1)
       ∧ FCIA ϱ1 Γ1 𝒱1 Tr_ES1 ∧ FCIA ϱ2 Γ2 𝒱2 Tr_ES2
       ∧ V𝒱1 ∩ V𝒱2 ⊆ ∇Γ1 ∪ ∇Γ2
       ∧ C𝒱1 ∩ N𝒱2 ⊆ ΥΓ1 ∧ C𝒱2 ∩ N𝒱1 ⊆ ΥΓ2
       ∧ N𝒱1 ∩ ΔΓ1 ∩ E_ES2 = {} ∧ N𝒱2 ∩ ΔΓ2 ∩ E_ES1 = {} ))"
```

Compositionality Results. In the following, we provide all compositionality results of *MAKS* in their Isabelle/HOL formalization. We have extracted these compositionality results from the theory `CompositionalityResults`.
For the compositionality results, it is assumed that `ES_valid ES1` and `ES_valid ES2` holds. Furthermore, it is assumed that `composable ES1 ES2` hold. Finally, it is also assumed that `isViewOn` $\mathcal{V}$ $E_{ES1\|ES2}$, `isViewOn` $\mathcal{V}1$ E_{ES1}, and `isViewOn` $\mathcal{V}2$ E_{ES2} hold.

Compositionality Results for Non-Strict BSPs:

theorem *compositionality_R:*
```
"⟦ R 𝒱1 Tr_ES1; R 𝒱2 Tr_ES2 ⟧ ⟹ R 𝒱 (Tr_(ES1 ∥ ES2))"
```

Compositionality Results for Strict BSPs:

theorem *compositionality_SR:*

$$\text{"}\llbracket\ SR\ \mathcal{V}1\ Tr_{ES1};\ SR\ \mathcal{V}2\ Tr_{ES2}\ \rrbracket \Longrightarrow SR\ \mathcal{V}\ (Tr_{(ES1\ \|\ ES2)})\text{"}$$

theorem *compositionality_SD:*

$$\text{"}\llbracket\ SD\ \mathcal{V}1\ Tr_{ES1};\ SD\ \mathcal{V}2\ Tr_{ES2}\ \rrbracket \Longrightarrow SD\ \mathcal{V}\ (Tr_{(ES1\ \|\ ES2)})\text{"}$$

theorem *compositionality_SI:*

$$\begin{array}{l}\text{"}\llbracket SD\ \mathcal{V}1\ Tr_{ES1};\ SD\ \mathcal{V}2\ Tr_{ES2};\ SI\ \mathcal{V}1\ Tr_{ES1};\ SI\ \mathcal{V}2\ Tr_{ES2}\ \rrbracket\\ \quad \Longrightarrow SI\ \mathcal{V}\ (Tr_{(ES1\ \|\ ES2)})\text{"}\end{array}$$

theorem *compositionality_SIA:*

$$\begin{array}{l}\text{"}\llbracket SD\ \mathcal{V}1\ Tr_{ES1};\ SD\ \mathcal{V}2\ Tr_{ES2};\ SIA\ \varrho 1\ \mathcal{V}1\ Tr_{ES1};\ SIA\ \varrho 2\ \mathcal{V}2\ Tr_{ES2};\\ \quad (\varrho 1\ \mathcal{V}1) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES1};\ (\varrho 2\ \mathcal{V}2) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES2}\ \rrbracket\\ \quad \Longrightarrow SIA\ \varrho\ \mathcal{V}\ (Tr_{(ES1\ \|\ ES2)})\text{"}\end{array}$$

Compositionality Results for Backwards-Strict and Forward Correctable BSPs:

theorem *compositionality_BSD:*

$$\text{"}\llbracket\ BSD\ \mathcal{V}1\ Tr_{ES1};\ BSD\ \mathcal{V}2\ Tr_{ES2}\ \rrbracket \Longrightarrow BSD\ \mathcal{V}\ Tr_{(ES1\ \|\ ES2)}\text{"}$$

theorem *compositionality_BSI:*

$$\begin{array}{l}\text{"}\llbracket\ BSD\ \mathcal{V}1\ Tr_{ES1};\ BSD\ \mathcal{V}2\ Tr_{ES2};\ BSI\ \mathcal{V}1\ Tr_{ES1};\ BSI\ \mathcal{V}2\ Tr_{ES2}\ \rrbracket\\ \quad \Longrightarrow BSI\ \mathcal{V}\ Tr_{(ES1\ \|\ ES2)}\text{"}\end{array}$$

theorem *compositionality_BSIA:*

$$\begin{array}{l}\text{"}\llbracket\ BSD\ \mathcal{V}1\ Tr_{ES1};\ BSD\ \mathcal{V}2\ Tr_{ES2};\ BSIA\ \varrho 1\ \mathcal{V}1\ Tr_{ES1};\ BSIA\ \varrho 2\ \mathcal{V}2\ Tr_{ES2};\\ \quad (\varrho 1\ \mathcal{V}1) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES1};\ (\varrho 2\ \mathcal{V}2) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES2}\ \rrbracket\\ \quad \Longrightarrow BSIA\ \varrho\ \mathcal{V}\ (Tr_{(ES1\ \|\ ES2)})\text{"}\end{array}$$

theorem *compositionality_FCD:*

$$\begin{array}{l}\text{"}\llbracket\ BSD\ \mathcal{V}1\ Tr_{ES1};\ BSD\ \mathcal{V}2\ Tr_{ES2};\\ \quad \nabla_{\Gamma} \cap E_{ES1} \subseteq \nabla_{\Gamma 1};\ \nabla_{\Gamma} \cap E_{ES2} \subseteq \nabla_{\Gamma 2};\\ \quad \Upsilon_{\Gamma} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1};\ \Upsilon_{\Gamma} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2};\\ \quad (\ \Delta_{\Gamma 1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma 2} \cap N_{\mathcal{V}2}\ ) \subseteq \Delta_{\Gamma};\\ \quad N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\};\ N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\};\\ \quad FCD\ \Gamma 1\ \mathcal{V}1\ Tr_{ES1};\ FCD\ \Gamma 2\ \mathcal{V}2\ Tr_{ES2}\ \rrbracket\\ \quad \Longrightarrow FCD\ \Gamma\ \mathcal{V}\ (Tr_{(ES1\ \|\ ES2)})\text{"}\end{array}$$

theorem *compositionality_FCI:*

$$\begin{array}{l}\text{"}\llbracket\ BSD\ \mathcal{V}1\ Tr_{ES1};\ BSD\ \mathcal{V}2\ Tr_{ES2};\ BSIA\ \varrho 1\ \mathcal{V}1\ Tr_{ES1};\ BSIA\ \varrho 2\ \mathcal{V}2\ Tr_{ES2};\\ \quad total\ ES1\ (C_{\mathcal{V}1} \cap \Upsilon_{\Gamma 1});\ total\ ES2\ (C_{\mathcal{V}2} \cap \Upsilon_{\Gamma 2});\\ \quad \nabla_{\Gamma} \cap E_{ES1} \subseteq \nabla_{\Gamma 1};\ \nabla_{\Gamma} \cap E_{ES2} \subseteq \nabla_{\Gamma 2};\\ \quad \Upsilon_{\Gamma} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1};\ \Upsilon_{\Gamma} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2};\\ \quad (\ \Delta_{\Gamma 1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma 2} \cap N_{\mathcal{V}2}\ ) \subseteq \Delta_{\Gamma};\\ \quad (N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma 1})\\ \quad \vee\ (\ N_{\mathcal{V}2} \cap \Delta_{\Gamma 2} \cap E_{ES1} = \{\} \wedge N_{\mathcal{V}1} \cap \Delta_{\Gamma 1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma 2})\ \ ;\\ \quad FCI\ \Gamma 1\ \mathcal{V}1\ Tr_{ES1};\ FCI\ \Gamma 2\ \mathcal{V}2\ Tr_{ES2}\ \rrbracket\\ \quad \Longrightarrow FCI\ \Gamma\ \mathcal{V}\ (Tr_{(ES1\ \|\ ES2)})\text{"}\end{array}$$

theorem *compositionality_FCIA:*

"$[\![$ BSD $\mathcal{V}1$ Tr_{ES1}; BSD $\mathcal{V}2$ Tr_{ES2}; BSIA $\varrho1$ $\mathcal{V}1$ Tr_{ES1}; BSIA $\varrho2$ $\mathcal{V}2$ Tr_{ES2};
$(\varrho1\ \mathcal{V}1) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES1}$; $(\varrho2\ \mathcal{V}2) \subseteq (\varrho\ \mathcal{V}) \cap E_{ES2}$;
total ES1 $(C_{\mathcal{V}1} \cap \Upsilon_{\Gamma1} \cap N_{\mathcal{V}2} \cap \Delta_{\Gamma2})$; total ES2 $(C_{\mathcal{V}2} \cap \Upsilon_{\Gamma2} \cap N_{\mathcal{V}1} \cap \Delta_{\Gamma1})$;
$\nabla_{\Gamma} \cap E_{ES1} \subseteq \nabla_{\Gamma1}$; $\nabla_{\Gamma} \cap E_{ES2} \subseteq \nabla_{\Gamma2}$;
$\Upsilon_{\Gamma} \cap E_{ES1} \subseteq \Upsilon_{\Gamma1}$; $\Upsilon_{\Gamma} \cap E_{ES2} \subseteq \Upsilon_{\Gamma2}$;
$(\Delta_{\Gamma1} \cap N_{\mathcal{V}1} \cup \Delta_{\Gamma2} \cap N_{\mathcal{V}2}) \subseteq \Delta_{\Gamma}$;
$(N_{\mathcal{V}1} \cap \Delta_{\Gamma1} \cap E_{ES2} = \{\} \wedge N_{\mathcal{V}2} \cap \Delta_{\Gamma2} \cap E_{ES1} \subseteq \Upsilon_{\Gamma1})$
$\vee\ (N_{\mathcal{V}2} \cap \Delta_{\Gamma2} \cap E_{ES1} = \{\} \wedge N_{\mathcal{V}1} \cap \Delta_{\Gamma1} \cap E_{ES2} \subseteq \Upsilon_{\Gamma2})$;
FCIA $\varrho1$ $\Gamma1$ $\mathcal{V}1$ Tr_{ES1}; FCIA $\varrho2$ $\Gamma2$ $\mathcal{V}2$ Tr_{ES2} $]\!]$
$\Longrightarrow$ FCIA ϱ Γ $\mathcal{V}$ $(Tr_{(ES1\ \|\ ES2)})$"