I-MAKS
A Framework for Information-Flow Security in Isabelle/HOL

April, 2018

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

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.

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 $\tau$ 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 $\alpha \in E^*$ and $\beta \in E^*$ (denoted by $\alpha.\beta$) is recursively defined by:

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

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 $\text{Tr} \subseteq E^*$ is closed under prefixes if each prefix $\alpha$ of each trace $\tau \in \text{Tr}$ is also contained in $\text{Tr}$, i.e. $\alpha \in \text{Tr}$.

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 $\tau$ to $E'$ (denoted by $\tau \upharpoonright E'$) is recursively defined by:

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

The event sequence $\tau \upharpoonright E'$ contains those occurrences of events in $\tau$ that are contained in the set $E'$. Note that the relative order of events in $\tau \upharpoonright E'$ is the same as in $\tau$. For instance, $\langle e_1.e_2.e_2.e_3 \rangle \upharpoonright \{ e_1, e_2 \}$ 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 $(E, \text{Tr})$ where $E$ is a set of events and $\text{Tr}$ is a set of traces over $E$ (i.e. $\text{Tr} \subseteq E^*$) that is closed under prefixes.

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

As convention, we refer to a trace $\tau \in \text{Tr}$ as possible trace of $\text{Tr}$ for a given set of traces $\text{Tr}$ modeling the behavior of a system. If the set of traces $\text{Tr}$ 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 $\tau$. We refer to a trace $\nu$ as a possible observation if there exists a trace $\tau \in Tr$ 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 $\nu$ by the set $\{\tau \in \text{Tr} \mid \tau \upharpoonright L = \nu\}$. That is, the set of all possible traces that potentially generated the observation $\nu$.

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 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 $(E, \text{Tr})$ 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 non-inference [O’H90, McL94, ZL97] is defined by:

$$NF \equiv \forall \tau \in \text{Tr}. \tau \upharpoonright L \in \text{Tr} \quad \Diamond$$

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 $(E, \text{Tr})$ 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:

$$SEP \equiv \forall \tau_L, \tau_H \in \text{Tr}. \forall t \in E^* . (t \upharpoonright L = \tau_L \upharpoonright L \land t \upharpoonright H = \tau_H \upharpoonright H) \Rightarrow t \in \text{Tr} \quad \Diamond$$

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 $\tau_H$. 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 $(E, \text{Tr})$ 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 \( P : \mathcal{P}(E^*) \rightarrow \mathcal{B} \) is a closure property on sets of traces if and only if for all \( \mathcal{Tr} \subseteq E^* \) there exists a set of traces such that \( \mathcal{Tr} \supseteq \mathcal{Tr} \) and \( P(\mathcal{Tr}) \).

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 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. 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:

```isabelle
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 ∪.

**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 \texttt{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:

\texttt{datatype \ texttt{'}a\ list = Nil \mid \texttt{Cons \ '}a \texttt{\ '}'a\ list\}}

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

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

\texttt{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 \texttt{field name = value} where comma is used as separator between list elements and the symbols \texttt{\{} and \texttt{\}} are used to mark the start and the end of the list, respectively. For instance, \texttt{\{}\ A = 3, B = 5, C = 7 \}} is a term of type \texttt{rec} from which the value of the second field can be retrieved using the field name of the second field, i.e. \texttt{\ B \ \{}\ A = 3, B = 5, C = 7 \}} is 5.

\textbf{Function Definitions.} Functions in Isabelle/HOL can be defined using the keywords \texttt{definition}, \texttt{primrec}, \texttt{fun}, and \texttt{function}.

The keyword \texttt{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 \textit{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 \texttt{square} for a natural number can be defined by:

\texttt{definition square :: "nat ⇒ nat"}
\texttt{where "square x ≡ x*x"}

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

\texttt{primrec set :: "'a list ⇒ 'a set"}
\texttt{where "set [] = {}" \mid "set (Cons x xs) = \{x\} \cup \texttt{set xs}"}

\texttt{primrec append :: "'a list ⇒ 'a list ⇒ 'a list"}
\texttt{where "[] @ ys = ys" \mid "(x#xs) @ ys = x \ #\ xs @ ys"}
Each of the function definitions consists of two equations (separated by \mid). 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 \texttt{primrec}, \texttt{fun}, or \texttt{function} define different classes of recursive functions. The keyword \texttt{primrec} defines recursive functions by giving exactly one reduction rule for each constructor. In contrast, \texttt{fun} and \texttt{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 patterns (each possible input is covered by exactly one equation), and termination. For functions defined using the keyword \texttt{fun}, these proofs are performed automatically. For functions defined using the keyword \texttt{function}, these proofs must be performed manually. Hence, \texttt{function} must be used instead of \texttt{fun} if the automatic proofs fail.

\textbf{Theorems, Lemmas, and Proofs.} Theorems and lemmas in Isabelle/HOL are defined using the keywords \texttt{theorem} and \texttt{lemma} followed by a formula or a proposition in Isabelle/HOL’s meta-logic. For instance, that from \( A \) and \( B \) it follows that \( A \land B \) holds is captured by the following theorem:

\texttt{theorem \"[A ; B ] \implies A \land 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 \texttt{apply}, proof tactics can be applied to transform the proof obligations until they are fully discharged. With the language \texttt{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

\textsc{I-MAKS} is an Isabelle/HOL formalization of MAKS in its version from [Man03]. The structure of theories in \textsc{I-MAKS} is depicted in Fig. 1. \textsc{I-MAKS} consists of two top-level components, a specification and a verification component. The specification component contains all parts of \textsc{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 \textsc{I-MAKS}. The latter contains everything related to the specification of information-flow properties in \textsc{I-MAKS}. The verification component contains all parts of \textsc{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
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 \( \tau \) 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 \( \text{prefix} \).

**definition** \( \text{prefix} :: \text{'e list} \Rightarrow \text{'e list} \Rightarrow \text{bool} \) (infixl “\( \preceq \)” 100)

where

\[
(11 \preceq 12) \equiv (\exists 13. 11 \oplus 13 = 12)
\]

As syntactic abbreviation for the predicate \( \text{prefix} \) \( 11 \ 12 \), I-MAKS supports the infix notation \( 11 \preceq 12 \).

The notion of closed under prefixes is formalized by the predicate \( \text{prefixclosed} \).

**definition** \( \text{prefixclosed} :: \text{('e list) set} \Rightarrow \text{bool} \)

where

\[
\text{prefixclosed}\ tr \equiv (\forall 11 \in\ tr. \forall 12. 12 \preceq 11 \rightarrow 12 \in\ tr)
\]

The notion of projection of \( l \) to a set \( E \) is formalized by the function \( \text{projection} \).

**definition** \( \text{projection} :: \text{('e list} \Rightarrow \text{'e set} \Rightarrow \text{'e list} \) (infixl “\( \upharpoonright \)” 100)

where

\[
1 \upharpoonright E \equiv \text{filter (} \lambda x . x \in E) \ 1
\]

As syntactic abbreviation for the function \( \text{projection} \) \( 1 \ E \), I-MAKS supports the infix notation \( 1 \upharpoonright E \).
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.

```plaintext
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 \(R\), \(F\ R\) 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 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 \(e\).

```plaintext
record \(\langle s, e \rangle\) SES_rec =
  S_SES :: \"s set\"
  s0_SES :: \"s\"
  E_SES :: \"e set\"
  I_SES :: \"e set\"
  O_SES :: \"e set\"
  T_SES :: \"s \rightarrow e \rightarrow s\"
```

Each term of the record type \(\langle s, e \rangle\) 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 \((T\_SES s e) = \text{Some } s'\), I-MAKS supports the usage of \(s \ x e \rightarrow SES \ s'\).

Similar to \(ES\_valid\), the predicate \(SES\_valid\) defines the semantic side conditions on terms of the record type \(\langle s, e \rangle\) SES_rec.

```plaintext
definition SES_valid :: \ \langle s, e \rangle\ SES_rec \Rightarrow bool
where
"SES\_valid SES \equiv
s0_is_state SES \land ses_inputs_are_events SES
\land ses_outputs_are_events SES \land ses_inputs_outputs_disjoint SES \land
correct_transition_relation SES"
```

```plaintext
definition s0_is_state :: \ \langle s, e \rangle\ SES_rec \Rightarrow bool
where
"s0_is_state SES \equiv s0\_ses \in S\_ses"
```

```plaintext
definition ses_inputs_are_events :: \ \langle s, e \rangle\ SES_rec \Rightarrow bool
where
"ses_inputs_are_events SES \equiv I\_ses \subseteq E\_ses"
```

```plaintext
definition ses_outputs_are_events :: \ \langle s, e \rangle\ SES_rec \Rightarrow bool
where
"ses_outputs_are_events SES \equiv O\_ses \subseteq E\_ses"
```

```plaintext
definition ses_inputs_outputs_disjoint :: \ \langle s, e \rangle\ SES_rec \Rightarrow bool
where
"ses_inputs_outputs_disjoint SES \equiv I\_ses \cap O\_ses = \{\}"
```

14
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 \( \text{'s} \) and type of events \( \text{'e} \) is a term of the record type \((\text{'s}, \text{'e}) \text{ SES_rec}\) that satisfies the predicate \(\text{SES_valid}\).

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

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

That is, there is a big-step transition from a state \(s\) to another state \(s'\) with the trace \(\tau\) if and only if there is a sequence of transitions in \(T\) from \(s\) to \(s'\) with the events of \(\tau\) 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 \(\text{SES} = (S, s_0, E, I, O, T)\) be a state-event system. The set of possible traces of \(\text{SES}\) (denoted by \(\text{Tr}_{\text{SES}}\)) is defined by \(\text{Tr}_{\text{SES}} = \{\tau \in E^* \mid \exists s \in S. (s_0, \tau, s) \in \widehat{T}_{\text{SES}}\}\).

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 \(\text{SES} = (S, s_0, E, I, O, T)\) be a state-event system. The event system induced by \(\text{SES}\) is the event system \(\text{ES} = (E, I, O, \text{Tr}_{\text{SES}})\).

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.

```haskell
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 (T SES 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`.

```haskell
definition possible_traces :: "('s, 'e) SES_rec ⇒ ('e list) set"
where
"possible_traces SES ≡ \{t. (enabled SES s0 SES t)\}"
```

```haskell
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`.

```haskell
definition induceES :: "('s, 'e) SES_rec ⇒ 'e ES_rec"
where
"induceES SES ≡ 
  | E_ES = E SES,
  | I_ES = I SES,
  | O_ES = O SES,
  | 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.

```haskell
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`. 

\[\diamond\]
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 $E_1 \cap E_2 \subseteq (O_1 \cap I_2) \cup (O_2 \cap I_1)$.

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

$$E = E_1 \cup E_2$$
$$I = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$$
$$O = (O_1 \setminus I_2) \cup (O_2 \setminus I_1)$$
$$Tr = \{ \tau \in E^* \mid \tau\lceil E_1 \in Tr_1 \land \tau\lceil E_2 \in Tr_2 \}.$$  

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} \cap E_{ES2} \subseteq (O_{ES1} \cap I_{ES2}) \cup (O_{ES2} \cap I_{ES1})"

Hence, two EVENT SYSTEMS are composable 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.

\[\text{definition composeES :: } \text{‘e ES_rec ⇒ ‘e ES_rec ⇒ ‘e ES_rec} \]

where

\[
\begin{align*}
E_{ES} &= E_{ES1} \cup E_{ES2}, \\
I_{ES} &= (I_{ES1} - O_{ES2}) \cup (I_{ES2} - O_{ES1}), \\
O_{ES} &= (O_{ES1} - I_{ES2}) \cup (O_{ES2} - I_{ES1}), \\
\text{Tr}_{ES} &= \{ \tau. (\tau \uparrow E_{ES1}) \in \text{Tr}_{ES1} \land (\tau \uparrow E_{ES2}) \in \text{Tr}_{ES2} \\
&\quad \land (\text{set } \tau \subseteq E_{ES1} \cup E_{ES2}) \} \}
\end{align*}
\]

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.

\[\text{lemma composeES_yields_ES :} \]

\[
\begin{align*}
(\text{ES_valid ES1} \land \text{ES_valid ES2}) \Rightarrow \text{ES_valid (ES1 ∥ ES2)}
\end{align*}
\]

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 \( V \) is a triple \((V, N, C)\) such that \( V \cap N = \emptyset \), \( V \cap C = \emptyset \) and \( N \cap C = \emptyset \). A view \( V = (V, N, C) \) 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 \( \text{'e V_rec} \) for a type of events \( \text{'e} \) and a corresponding predicate \( \text{V_valid} \) (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 26. A view for a type of events \( \text{'e} \) is a term of the record type \( \text{'e V_rec} \) that satisfies the predicate \( \text{V_valid} \).
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 $\text{isViewOn}$.

**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 $\text{isViewOn}$ 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 $\text{BSP}_V$ denotes the closure property on sets of traces obtained by applying a BSP $\text{BSP}$ to a view $V$.

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 $V = (V, N, C)$ be a view on $E$. The basic security predicate Backwards-Strict Deletion (denoted by BSD) is defined by:

$$\text{BSD}_V(Tr) \equiv \forall \alpha, \beta \in E^*. \forall c \in C. \left[ (\beta, \langle c \rangle. \alpha \in Tr \land \alpha \land c \subset V = \langle \rangle \right] \Rightarrow (\exists \alpha' \in E^*. \beta. \alpha' \in Tr \land \alpha' \land V = \alpha \land V \land \alpha' \land c \subset V = \langle \rangle)$$

Figure 2: Pattern underlying the definition of all BSPs [Man03].
Figure 3: Names of BSPs in MAKS [Man03].

For BSD, the perturbation is the deletion of the last occurrence of a 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 a 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 w.r.t. 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 \textit{insertion} or the suffix \( IA \) for \textit{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.

\textbf{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 \( 'e BSP = 'e V_rec \Rightarrow ('e list) set \Rightarrow bool \)
definition BSP_valid :: \( 'e BSP \Rightarrow bool \)
where
\( BSP Valid bsp \equiv \forall V Tr E. ( isViewOn V E \land areTracesOver Tr E ) \rightarrow ( \exists Tr'. Tr' \supseteq Tr \land bsp V Tr') \)
perturbed trace $\tau \upharpoonright V \cup N$ 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 $\text{VIEW} V$ 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 29.** Let $V$ be a set of views. An information-flow property is a pair $(\forall S, B)$ where $\forall S$ is a set of views on $E$ and $B$ is a set of BSPs. An information-flow property is satisfied for a set of traces $Tr \subseteq E^*$ iff $BSP_V(Tr)$ holds for each $V \in \forall S$ and each $BSP \in B$.

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 $(\forall S, B)$ where $\forall S$ is a set of views on $E$ and $B$ is a set of BSPs. An information-flow property is satisfied for a set of traces $Tr \subseteq E^*$ iff $BSP_V(Tr)$ holds for each $V \in \forall S$ and each $BSP \in B$.\phantom{a}
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 \( \text{‘e IFP_type} = (\text{‘e V_rec} \text{ set}) \times \text{‘e SP} \)

type synonym \( \text{‘e SP} = (\text{‘e BSP} \text{ set}) \)

The corresponding predicate IFP_valid formalizes the semantic side conditions of information-flow properties on terms of the type \( \text{‘e IFP_type} \).

definition IFP_valid :: \( \text{‘e set} \Rightarrow \text{‘e IFP_type} \Rightarrow \text{bool} \)

where
"IFP_valid E ifp ≡ ∀ V ∈ (fst ifp). isViewOn V E ∧ (∀ BSP ∈ (snd ifp). BSP_valid BSP)"

Combining the type \( \text{‘e IFP_type} \) and the predicate IFP_valid, I-MAKS formalizes information-flow properties as follows.

Definition 32. An INFORMATION-FLOW PROPERTY for a type of events \( \text{‘e} \) is a term of \( \text{‘e IFP_type} \) that satisfies the predicate IFP_valid.

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

definition IFPIsSatisfied :: \( \text{‘e IFP_type} \Rightarrow (\text{‘e list} \text{ set}) \Rightarrow \text{bool} \)

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.
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 V be a VIEW ON SES.
All unwinding conditions only reason about the reachable states of a state-event system. The predicate \texttt{reachable} characterizes these states in \textit{I-MAKS}.

\textbf{Definition} \texttt{reachable} :: "\('s, \ 'e) \ SES\_rec \Rightarrow \ 's \Rightarrow \ bool" \ where 
\texttt{reachable \ SES \ s \ \equiv \ \(\exists \ t. \ s0_{\text{ses}} \ t_{\Rightarrow_{\text{ses}} s}\)"}

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 \textit{I-MAKS} utilizing the predicate \texttt{lrf} on binary relations of states.

\textbf{Definition} \texttt{lrf} :: "\('s \ rel \Rightarrow \ bool" \ where 
\texttt{lrf \ ur} \ \equiv \ \(\forall s \in S_{\text{ses}}. \ \forall s' \in S_{\text{ses}}. \ \forall c \in C_{\text{v}}. \ ((\text{reachable \ SES \ s} \ \land \ s \ c_{\Rightarrow_{\text{ses}} s'}) \ \Rightarrow \ (s', s) \in ur)\)"

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

Using the predicate \texttt{lrf}, \textit{I-MAKS} formalizes the unwinding condition locally-respects forwards as follows.

\textbf{Definition 34}. Let \textit{SES} be a state-event system. Let \textit{V} be a view on \textit{E}_{\textit{SES}}. The unwinding condition locally-respects forwards is satisfied for a binary relation \textit{ur} on states of type \textit{'s} if and only if \texttt{lrf \ ur} is satisfied.

The unwinding condition output-step consistency is formalized in \textit{I-MAKS} utilizing the predicate \texttt{osc} on binary relations of states.

\textbf{Definition} \texttt{osc} :: "\('s \ rel \Rightarrow \ bool" \ where 
\texttt{osc \ ur} \ \equiv \ \(\forall s1 \in S_{\text{ses}}. \ \forall s'1 \in S_{\text{ses}}. \ \forall s2' \in S_{\text{ses}}. \ \forall e \in (E_{\text{ses}} \ - \ C_{\text{v}}). \ ((\text{reachable \ SES \ s1} \ \land \ s1' e_{\Rightarrow_{\text{ses}} s2'} \ \land (s1', s1) \in ur) \ \Rightarrow \ (\exists s2 \in S_{\text{ses}}. \ \exists \delta. \ \delta \upharpoonright C_{\text{v}} = \{\} \ \land \ \delta \upharpoonright V_{\text{v}} = \{e\} \ \land V_{\text{v}} \ \land s1 \delta_{\Rightarrow_{\text{ses}} s2} \ \land (s2', s2) \in ur)\)"

Intuitively, the predicate \texttt{osc} captures the following requirement (cf. Fig. 4): If any two states \textit{s}1' and \textit{s}1 are indistinguishable, then a possible transition with a confidential event \textit{e} from \textit{s}1' to \textit{s}2' has to be matched by a sequence of transitions with a trace \textit{delta} from \textit{s}1 to \textit{s}2. Thereby, the trace \textit{delta} can differ from the trace \{\textit{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 \texttt{osc}, \textit{I-MAKS} formalizes the unwinding condition output-step consistency as follows.

\textbf{Definition 35}. Let \textit{SES} be a state-event system. Let \textit{V} be a view on \textit{E}_{\textit{SES}}. The unwinding condition output-step consistency is satisfied for a binary relation \textit{ur} on states of type \textit{'s} if and only if \texttt{osc \ ur} is satisfied.
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:**

\[ \left( lrf \ ur; \ osc \ ur \right) \implies BSD \ V \ Tr_{(induceES \ SES)} \]

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 \( V \).

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 V1 be a VIEW ON E_{ES1}, V2 be a VIEW ON E_{ES2}, and 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 36.** The views V1 and V2 constitute a proper separation of V for the EVENT SYSTEMS ES1 and ES2 if and only if properSeparationOfViews ES1 ES2 V V1 V2 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.
\[ \forall (\exists! g_1 g_2 \Gamma_1 \Gamma_2. \ ( \begin{array}{l} \nabla_{r_1} \subseteq E_{\text{esi}} \land \Delta_{r_1} \subseteq E_{\text{esi}} \land \Upsilon_{r_1} \subseteq E_{\text{esi}} \\ \nabla_{r_2} \subseteq E_{\text{esi}} \land \Delta_{r_2} \subseteq E_{\text{esi}} \land \Upsilon_{r_2} \subseteq E_{\text{esi}} \\ \text{BSIA} g_1 \nabla_1 \text{Tr}_{\text{esi}} \land \text{BSIA} g_2 \nabla_2 \text{Tr}_{\text{esi}} \\ \text{total} ES_1 \ (C_{\nu_1} \cap N_{\nu_2}) \land \text{total} ES_2 \ (C_{\nu_2} \cap N_{\nu_1}) \\ \text{FCIA} g_1 \nabla_1 \text{Tr}_{\text{esi}} \land \text{FCIA} g_2 \nabla_2 \text{Tr}_{\text{esi}} \\ \nabla_{\nu_1} \cap \nabla_{\nu_2} \subseteq \nabla_{r_1} \cup \nabla_{r_2} \\ C_{\nu_1} \cap N_{\nu_2} \subseteq \Upsilon_{r_1} \land C_{\nu_2} \cap N_{\nu_1} \subseteq \Upsilon_{r_2} \\ N_{\nu_1} \cap \Delta_{r_1} \cap E_{\text{esi}} = \emptyset \land N_{\nu_2} \cap \Delta_{r_2} \cap E_{\text{esi}} = \emptyset \end{array} ) ) \) ]

\text{definition total} :: "'e ES_rec \Rightarrow 'e set \Rightarrow \text{bool}"

\text{where}

"\text{total} ES E \equiv E \subseteq E_{\text{esi}} \land (\forall \tau \in \text{Tr}_{\text{esi}}. \ \forall e \in E. \ \tau \notin [e] \in \text{Tr}_{\text{esi}})"

The four disjuncts of the predicate \text{wellBehavedComposition} establish a case distinction on the truth values of \( N_{\nu_1} \cap E_{\text{esi}} = \emptyset \) and \( N_{\nu_2} \cap E_{\text{esi}} = \emptyset \). If \( N_{\nu_1} \cap E_{\text{esi}} = \emptyset \) holds, no shared events are affected by corrections in the first component. Likewise, if \( N_{\nu_2} \cap E_{\text{esi}} = \emptyset \) 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 \text{wellBehavedComposition}, I-MAKS captures the condition well-behaved composition as follows.

\textbf{Definition 37.} Suppose that \( \nabla_1 \) and \( \nabla_2 \) constitute a proper separation of \( \nabla \) for the event systems \( ES_1 \) and \( ES_2 \). The composition of \( ES_1 \) and \( ES_2 \) is a well-behaved composition \( \nabla \) \( ES_1 \) \( ES_2 \) \( \nabla \nabla_1 \nabla_2 \) if and only if \( \text{wellBehavedComposition} \)

\( \nabla \nabla_1 \nabla_2 \) \( ES_1 \) \( ES_2 \) 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

\textbf{theorem compositionality_BSD:}

"[ BSD \nabla_1 \text{Tr}_{\text{esi}}; BSD \nabla_2 \text{Tr}_{\text{esi}} ] \implies BSD \nabla \text{Tr}_{\text{esi} \parallel \text{esi}} "

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 \( \nabla_1 \) and \( \nabla_2 \) constitute a proper separation of \( \nabla \) for \( ES_1 \) and \( ES_2 \) as well as that the composition of \( ES_1 \) and \( ES_2 \) is a well-behaved composition \( \nabla \) \( ES_1 \) \( ES_2 \) \( \nabla \nabla_1 \nabla_2 \). 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) [CM+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


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.

\textbf{type synonym} \texttt{'}e BSP = "'e V_rec ⇒ ('e list) set) ⇒ bool"

\textbf{definition} \texttt{BSP_valid :: 'e BSP ⇒ bool}

\textbf{where}

\texttt{"BSP_valid bsp ≡ ∀V Tr E. ( isViewOn V E ∧ areTracesOver Tr E ) → (∃ Tr'. Tr' ⊇ Tr ∧ bsp V Tr')"}

\textbf{Supplementary Definitions.} We provide definitions of the types for \(\rho\) and \(\Gamma\) that are used as additional parameters in the BSPs IA, SIA, BSIA, FCD, FCI, and FCIA below.

\textbf{type synonym} \texttt{'}e Rho = "'e V_rec ⇒ 'e set"

\textbf{record} \texttt{'}e Gamma =

\texttt{Nabla :: 'e set

Delta :: 'e set

Upsilon :: 'e set}

In addition, we provide the definition of \(\rho\)-admissibility in I-MAKS which is used in the definitions of IA, SIA, BSIA, and FCIA.

\textbf{definition} \texttt{Adm :: 'e V_rec ⇒ 'e Rho ⇒ ('e list) set ⇒ 'e list ⇒ 'e ⇒ bool}

\textbf{where}

\texttt{"Adm V ϱ Tr β e ≡ \exists γ. ((γ @ [e]) ∈ Tr ∧ γ↾(ϱ V) = β↾(ϱ V))"}

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

\textbf{definition} \texttt{R :: 'e BSP}

\textbf{where}

\texttt{"R V Tr ≡ ∀τ ∈ Tr. (∃τ' ∈ Tr. τ ↾ C_v = [] ∧ τ' ↾ V_v = τ ↾ V_v)"}

\textbf{lemma} \texttt{BSP_valid_R: "BSP_valid R"}

\textbf{definition} \texttt{D :: 'e BSP}

\textbf{where}

\texttt{"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 \textit{BSP\_valid\textunderscore D}: "BSP\_valid\textunderscore D"

definition \textit{I} :: "'e BSP"
where
"\textit{I} \ V \ Tr \equiv \forall \alpha \beta. \forall c \in C_{V}. ((\beta \in \alpha) \in Tr \land \alpha|C_{V} = [])
\rightarrow (\exists \alpha' \beta'. \((\beta' \in [c] \in \alpha') \in Tr \land \alpha'|V_{\neg} = \alpha|V_{\neg} \land \alpha'|C_{V} = []
\land \beta'|1(V_{\neg} \cup C_{V}) = \beta|1(V_{\neg} \cup C_{V}))"

lemma \textit{BSP\_valid\textunderscore I}: "BSP\_valid\textunderscore I"

definition \textit{IA} :: "'e Rho \Rightarrow 'e BSP"
where
"\textit{IA} \varrho \ V \ Tr \equiv \forall \alpha \beta. \forall c \in C_{V}. ((\beta \in \alpha) \in Tr \land \alpha|C_{V} = []) \land (\text{Adm}_{V} \varrho \ Tr \beta \ c)
\rightarrow (\exists \alpha' \beta'. \((\beta' \in [c] \in \alpha') \in Tr) \land \alpha'|V_{\neg} = \alpha|V_{\neg}
\land \alpha'|C_{V} = []) \land \beta'|1(V_{\neg} \cup C_{V}) = \beta|1(V_{\neg} \cup C_{V}))"

lemma \textit{BSP\_valid\textunderscore IA}: "BSP\_valid\textunderscore (IA \varrho) "

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

definition \textit{SR} :: "'e BSP"
where
"\textit{SR} \ V \ Tr \equiv \forall \tau \in Tr. \tau \downarrow (V_{\neg} \cup N_{\neg}) \in Tr"

lemma "BSP\_valid SR"

definition \textit{SD} :: "'e BSP"
where
"\textit{SD} \ V \ Tr \equiv \forall \alpha \beta. \forall c \in C_{V}. ((\beta \in \alpha) \in Tr \land \alpha|C_{V} = []) \land (\Adm_{V} \beta \ c)
\rightarrow \beta \in \alpha \in Tr"

lemma "BSP\_valid SD"

definition \textit{SI} :: "'e BSP"
where
"\textit{SI} \ V \ Tr \equiv \forall \alpha \beta. \forall c \in C_{V}. ((\beta \in \alpha) \in Tr \land \alpha|C_{V} = []) \rightarrow \beta \in [c] \in \alpha \in Tr"

lemma "BSP\_valid SI"

definition \textit{SIA} :: "'e Rho \Rightarrow 'e BSP"
where
"\textit{SIA} \varrho \ V \ Tr \equiv \forall \alpha \beta. \forall c \in C_{V}. ((\beta \in \alpha) \in Tr \land \alpha|C_{V} = []) \land (\text{Adm}_{V} \varrho \ Tr \beta \ c)
\rightarrow (\beta \in [c] \in \alpha \in Tr)"

lemma "BSP\_valid\textunderscore (SIA \varrho) "

36
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 ∈ C V. (β @ [c] @ α) ∈ Tr ∧ α C V = []
→ (∃ α'. ((β @ α') ∈ Tr ∧ α' V V = α V V ∧ α' C V = []))"

lemma BSP_valid_BSD: "BSP_valid BSD"

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

lemma BSP_valid_BSI: "BSP_valid BSI"

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

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 ∈ (C V ∩ Γ). ∀ v ∈ (V V ∩ Δ Γ).
((β @ [c] @ v) @ α) ∈ Tr ∧ α C V = [] ∧ (Adm Γ V Tr β c)
→ (∃ α'. ((β @ [c] @ v) @ α') ∈ Tr ∧ α' V V = α V V ∧ α' C V = []))"

lemma BSP_valid_FCD: "BSP_valid (FCD Γ) "

definition FCI :: "'e Gamma ⇒ 'e BSP"
where "FCI Γ V Tr ≡
∀ α β. ∀ c ∈ (C V ∩ Γ). ∀ v ∈ (V V ∩ Δ Γ).
((β @ [v] @ α) ∈ Tr ∧ α C V = []
→ (∃ α'. (set v') ⊆ (N V ∩ Δ Γ)
∧ ((β @ v' @ [v] @ α') ∈ Tr
∧ α' V V = α V V ∧ α' C V = []))"

lemma BSP_valid_FCI: "BSP_valid (FCI Γ) "

lemma BSP_valid_FCI: "BSP_valid (FCI Γ)"

definition FCIA :: 'e Rho ⇒ 'e Gamma ⇒ 'e BSP
where "FCIA ϱ Γ V Tr ≡ ∀ α β. ∀ c ∈ (C V ∩ Γ). ∀ v ∈ (V V ∩ Γ).
(∃ α'. ∃ δ'. (set δ') ⊆ (N V ∩ Γ).
∧ (∃ β. (β @ [v] @ α ∈ Tr ∧ α↿ C V = {} ∧ (Adm V g Tr β c))
→ (∃ α'. ∃ δ'. (set δ') ⊆ (N V ∩ Γ).
∧ ((β @ [c] @ δ' @ [v] @ α') ∈ Tr
∧ α'↾V V = α↾V V ∧ α' ↾ C 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 J-MAKS. We extracted all of these results from the theory BSPTaxonomy.
For all of these results, it is assumed ES_valid ES, isViewOn V ES, isViewOn V 1 ES,
and isViewOn V 2 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 V Tr ES ⇒ R V Tr ES"

lemma BSD_implies_D:
"BSD V Tr ES ⇒ D V Tr ES"

lemma SD_implies_BSD :
"(SD V Tr ES) ⇒ BSD V Tr ES "

lemma SD_implies_SR:
"SD V Tr ES ⇒ SR V Tr ES"

Taxonomy Results for Modified Views: For the following taxonomy results it is assumed
that for the two views V 1 and V 2, we have that V V 2 ⊆ V V 1, N V 2 ⊇ N V 1, and C V 2 ⊆ C V 1
hold.

lemma R_implies_R_for_modified_view:
"R V 1 Tr ES ⇒ R V 2 Tr ES"

lemma D_implies_D_for_modified_view:
"D V 1 Tr ES ⇒ D V 2 Tr ES"

lemma BSD_implies_BSD_for_modified_view:
"BSD V 1 Tr ES⇒ BSD V 2 Tr ES"

lemma SD_implies_FCD:
"(SD V Tr ES) ⇒ FCD Γ V Tr ES"
Further Taxonomy Results: For the following taxonomy results it is assumed that for the two views $V_1$ and $V_2$, we have that $V_{V_2} \subseteq V_{V_1}$, $N_{V_2} \supseteq N_{V_1}$, and $C_{V_2} = C_{V_1}$ hold. Furthermore, is assumed that for $\Gamma_1$ and $\Gamma_2$, we have that $V_{V_2} \cap \nabla_{V_2} \subseteq V_{V_1} \cap \nabla_{V_1}$, $C_{V_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{V_1} \cap \Upsilon_{\Gamma_1}$, and $N_{V_2} \cap \Delta_{\Gamma_2} \supseteq N_{V_1} \cap \Delta_{\Gamma_1}$ hold.

**lemma** $\text{FCD_implies_FCD_for_modified_view_gamma}$:

"$[\text{FCD } \Gamma_1 V_1 \text{ Tr}_{\text{es}}; \quad V_{V_2} \cap \nabla_{V_2} \subseteq V_{V_1} \cap \nabla_{V_1}, \quad N_{V_2} \cap \Delta_{\Gamma_2} \supseteq N_{V_1} \cap \Delta_{\Gamma_1}, \quad C_{V_2} \cap \Upsilon_{\Gamma_2} \subseteq C_{V_1} \cap \Upsilon_{\Gamma_1}] \implies \text{FCD } \Gamma_2 V_2 \text{ Tr}_{\text{es}}$"

Trivial Satisfaction Results:

**lemma** $\text{Trivially_fulfilled_D_C_empty}$:

"$C_{V} = \{} \implies D \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_BSD_C_empty}$:

"$C_{V} = \{} \implies BSD \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_R_C_empty}$:

"$C_{V} = \{} \implies R \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_SD_C_empty}$:

"$C_{V} = \{} \implies SD \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_FCD_C_empty}$:

"$C_{V} = \{} \implies FCD \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_R_V_empty}$:

"$V_{V} = \{} \implies R \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_D_V_empty}$:

"$V_{V} = \{} \implies D \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_BSD_V_empty}$:

"$V_{V} = \{} \implies BSD \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_FCD_V_empty}$:

"$V_{V} = \{} \implies FCD \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_FCD_Nabla_\Upsilon_empty}$:

"$[\nabla_{\Gamma} = \{} \lor \Upsilon_{\Gamma} = \{}] \implies \text{FCD } \Gamma \text{ V Tr}_{\text{es}}$"

**lemma** $\text{Trivially_fulfilled_FCD_N_subseteq_\Delta_and_BSD}$:

"$[N_{V} \subseteq \Delta_{\Gamma} ; BSD \text{ V Tr}_{\text{es}}] \implies \text{FCD } \Gamma \text{ V Tr}_{\text{es}}$"
Taxonomy of BSPs in the Second Dimension.

**Taxonomy Results for the Same View:**

**lemma** \( \text{SI_implies_BSI} : (SI \lor Tr_{es}) \implies BSI \lor Tr_{es} \)**

**lemma** \( \text{BSI_implies_I} : (BSI \lor Tr_{es}) \implies (I \lor Tr_{es}) \)**

**lemma** \( \text{SIA_implies_BSI} : (SIA \lor Tr_{es}) \implies (BSIA \lor Tr_{es}) \)**

**lemma** \( \text{SI_implies_SIA} : (SI \lor Tr_{es}) \implies (SIA \lor Tr_{es}) \)**

**lemma** \( \text{BSI_implies_BSI} : (BSI \lor Tr_{es}) \implies (BSIA \lor Tr_{es}) \)**

**lemma** \( \text{BSIA_implies_BSIA_for_modified_view} : (BSIA \lor Tr_{es}) \implies (BSIA \lor Tr_{es}) \)**

**lemma** \( \text{I_implies_I_for_modified_view} : (I \lor Tr_{es}) \implies (I \lor Tr_{es}) \)**

**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 \( \mathcal{V}_2 \subseteq \mathcal{V}_1 \), \( N_{V_2} \supseteq N_{V_1} \), and \( C_{V_2} = C_{V_1} \) hold.

**lemma** \( \text{IA_implies_I_for_modified_view} : (I \lor Tr_{es}) \implies (I \lor Tr_{es}) \)**

**lemma** \( \text{BSI_implies_BSIA_for_modified_view} : (BSI \lor Tr_{es}) \implies (BSIA \lor Tr_{es}) \)**

**lemma** \( \text{SI_implies_SI_for_modified_view} : (SI \lor Tr_{es}) \implies (SI \lor Tr_{es}) \)**

**lemma** \( \text{SIA_implies_SIA_for_modified_view} : (SIA \lor Tr_{es}) \implies (SIA \lor Tr_{es}) \)**

**lemma** \( \text{BSIA_implies_BSIA_for_modified_view} : (BSIA \lor Tr_{es}) \implies (BSIA \lor Tr_{es}) \)**

**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 \( \mathcal{V}_2 \subseteq \mathcal{V}_1 \), \( N_{V_2} \supseteq N_{V_1} \), and \( C_{V_2} = C_{V_1} \) hold. Furthermore, is assumed that for \( \Gamma_1 \) and \( \Gamma_2 \), we have that \( \mathcal{V}_{V_2} \cap \nabla_{r_2} \subseteq \mathcal{V}_{V_1} \cap \nabla_{r_1} \), \( C_{V_2} \cap r_2 \subseteq C_{V_1} \cap r_1 \), and \( N_{V_2} \cap \Delta_{r_2} \supseteq N_{V_1} \cap \Delta_{r_1} \) hold.

**lemma** \( \text{FCI_implies_FCI_for_modified_view_gamma} : (FCI \lor \Gamma_1 \lor Tr_{es}; \ g_1(V_1) \supseteq g_2(V_2), \ g_1(V_1) \supseteq g_2(V_2) \lor Tr_{es} \)**

**lemma** \( \text{FCIA_implies_FCIA_for_modified_view_rho_gamma} : (FCIA \lor \Gamma_1 \lor Tr_{es}; \ g_1(V_1) \supseteq g_2(V_2), \ g_1(V_1) \supseteq g_2(V_2) \lor Tr_{es} \)**
Trivial Satisfaction Results:

**Lemma** Trivially fulfilled I_C_empty:
\[ C^V = {} \implies I^V Tr_{es} \]

**Lemma** Trivially fulfilled IA_C_empty:
\[ C^V = {} \implies IA^V Tr_{es} \]

**Lemma** Trivially fulfilled BSI_C_empty:
\[ C^V = {} \implies BSI^V Tr_{es} \]

**Lemma** Trivially fulfilled BSIA_C_empty:
\[ C^V = {} \implies BSIA^V Tr_{es} \]

**Lemma** Trivially fulfilled SI_C_empty:
\[ C^V = {} \implies SI^V Tr_{es} \]

**Lemma** Trivially fulfilled SIA_C_empty:
\[ C^V = {} \implies SIA^V Tr_{es} \]

**Lemma** Trivially fulfilled FCI_C_empty:
\[ C^V = {} \implies FCI^V Tr_{es} \]

**Lemma** Trivially fulfilled FCIA_C_empty:
\[ C^V = {} \implies FCIA^V Tr_{es} \]

**Lemma** Trivially fulfilled I_V_empty:
\[ V^V = {} \implies I^V Tr_{es} \]

**Lemma** Trivially fulfilled IA_V_empty:
\[ V^V = {} \implies IA^V Tr_{es} \]

**Lemma** Trivially fulfilled BSI_V_empty:
\[ V^V = {} \implies BSI^V Tr_{es} \]

**Lemma** Trivially fulfilled BSIA_V_empty:
\[ V^V = {} \implies BSIA^V Tr_{es} \]

**Lemma** Trivially fulfilled SI_V_empty:
\[ V^V = {} \implies SI^V Tr_{es} \]

**Lemma** Trivially fulfilled SIA_V_empty:
\[ V^V = {} \implies SIA^V Tr_{es} \]

**Lemma** Trivially fulfilled FCI_V_empty:
\[ V^V = {} \implies FCI^V Tr_{es} \]

**Lemma** Trivially fulfilled FCIA_V_empty:
\[ V^V = {} \implies FCIA^V Tr_{es} \]

**Lemma** Trivially fulfilled IA_V_empty_rho_subseteq_C_N:
\[ [V^V = {}; \rho^V \supseteq (C^V \cup N^V)] \implies IA^V \rho^V Tr_{es} \]

**Lemma** Trivially fulfilled BSIA_V_empty_rho_subseteq_C_N:
\[ [V^V = {}; \rho^V \supseteq (C^V \cup N^V)] \implies BSIA^V \rho^V Tr_{es} \]

**Lemma** Trivially fulfilled BSI_V_empty_total_ES_C:
\[ [V^V = {}; total ES C^V] \implies BSI^V Tr_{es} \]

**Lemma** Trivially fulfilled I_V_empty_total_ES_C:
\[ [V^V = {}; total ES C^V] \implies I^V Tr_{es} \]

**Lemma** Trivially fulfilled FCI_Na_o_\(\Upsilon\)_empty:
\[ [\nabla_{\Gamma} = {} \lor \Upsilon_{\Gamma} = {}] \implies FCI^\Gamma Tr_{es} \]

**Lemma** Trivially fulfilled FCIA_Na_o_\(\Upsilon\)_empty:
\[ [\nabla_{\Gamma} = {} \lor \Upsilon_{\Gamma} = {}] \implies FCIA^\rho \Gamma Tr_{es} \]

**Lemma** Trivially fulfilled FCI_N_subseteq_\(\Delta\)_and_BSI:
\[ [N^V \subseteq \Delta_{\Gamma}; BSI^V \rho^V Tr_{es}] \implies FCI^\Gamma \rho^V Tr_{es} \]

**Lemma** Trivially fulfilled FCIA_N_subseteq_\(\Delta\)_and_BSIA:
\[ [N^V \subseteq \Delta_{\Gamma}; BSIA^V \rho^V Tr_{es}] \implies FCIA^\rho \Gamma \rho^V Tr_{es} \]
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.

\[
\text{type synonym } e\ SP = \{e\ BSP\} \text{ set}
\]

\[
\text{type synonym } e\ IFP\_type = \{e\ V\_rec\ set\} \times e\ SP
\]

\[
\text{definition } IFP\_valid :: e\ set \Rightarrow e\ IFP\_type \Rightarrow \text{bool}
\]
where
"\text{IFP\_valid } E \text{ ifp } \equiv \forall V \in (\text{fst ifp}). \text{isViewOn } V \ W \land \forall BSP \in (\text{snd ifp}). \text{BSP\_valid } BSP"

\[
\text{definition } IFPIsSatisfied :: e\ IFP\_type \Rightarrow (e\ list) \text{ set} \Rightarrow \text{bool}
\]
where
"\text{IFPIsSatisfied } ifp \ Tr \equiv \forall V \in (\text{fst ifp}). \forall BSP \in (\text{snd ifp}). \text{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.

\[
\text{definition } HighInputsConfidential :: e\ set \Rightarrow e\ set \Rightarrow e\ V\_rec
\]
where
"\text{HighInputsConfidential } L \ H \ IE \equiv \{ | V=L, N=H-IE, C=H \cap IE \}"

\[
\text{definition } HighConfidential :: e\ set \Rightarrow e\ set \Rightarrow e\ V\_rec
\]
where
"\text{HighConfidential } L \ H \equiv \{ | V=L, N={}, C=H \}"

\[
\text{fun } interleaving :: e\ list \Rightarrow e\ list \Rightarrow (e\ list) \text{ set}
\]
where
"interleaving t1 [] = \{t1\} | \text{interleaving } [] t2 = \{t2\} | \text{interleaving } (e1 \# t1) \ (e2 \# t2) = \{t. (\exists t'. t=(e1 \# t') \land t' \in \text{interleaving } t1 \ (e2 \# t2))\}
\cup \{t. (\exists t'. t=(e2 \# t') \land t' \in \text{interleaving } (e1 \# t1) \ t2)\}"

42
Generalized Noninterference.

definition GNI :: 
  "\'e set \Rightarrow \'e set \Rightarrow \'e set \Rightarrow \'e IFP_type"
where
"GNI L H IE \equiv (\text{HighInputsConfidential} L H IE), (\text{BSD}, \text{BSI})"

lemma GNI_valid: 
  "L \cap H = {} \implies \text{IFP_valid} (L \cup H) (\text{GNI} L H IE)"

definition litGNI :: 
  "\'e set \Rightarrow \'e set \Rightarrow \'e set \Rightarrow (\'e list) set \Rightarrow \text{bool}"
where
"litGNI L H IE Tr \equiv 
  \forall t_1 t_2 t_3. 
  t_1 \not\in Tr \land t_2 \uparrow (L \cup (H - IE)) = t_2 \uparrow (L \cup (H - IE))
  \implies (\exists t_4. t_1 \not\in Tr \land t_4 \uparrow (L \cup (H \cap IE)) = t_3 \uparrow (L \cup (H \cap IE)))"

Interleaving-based Generalized Noninterference.

definition IBGNI :: 
  "\'e set \Rightarrow \'e set \Rightarrow \'e set \Rightarrow \'e IFP_type"
where
"IBGNI L H IE \equiv (\text{HighInputsConfidential} L H IE), (D, I)"

lemma IBGNI_valid: 
  "L \cap H = {} \implies \text{IFP_valid} (L \cup H) (\text{IBGNI} L H IE)"

definition litIBGNI :: 
  "\'e set \Rightarrow \'e set \Rightarrow \'e set \Rightarrow (\'e list) set \Rightarrow \text{bool}"
where
"litIBGNI L H IE Tr \equiv 
  \forall \tau _l \in Tr. \forall t_hi t. 
  (\text{set} t_hi) \subseteq (H \cap IE) \land t \in \text{interleaving} t_hi (\tau _l \uparrow L)
  \implies (\exists \tau ' \in Tr. \tau ' \uparrow (L \cup (H \cap IE)) = t)"

Forward Correctability.

definition FC :: 
  "\'e set \Rightarrow \'e set \Rightarrow \'e set \Rightarrow \'e IFP_type"
where
"FC L H IE \equiv 
  (\text{HighInputsConfidential} L H IE), 
  (\text{BSD}, \text{BSI}, (\text{FCD} \{\text{Nabla=IE, Delta=}, \text{Upsilon=IE}\}), 
  (\text{FCI} \{\text{Nabla=IE, Delta=}, \text{Upsilon=IE}\}))"

lemma FC_valid: 
  "L \cap H = {} \implies \text{IFP_valid} (L \cup H) (\text{FC} L H IE)"

definition litFC :: 
  "\'e set \Rightarrow \'e set \Rightarrow \'e set \Rightarrow (\'e list) set \Rightarrow \text{bool}"
where
"litFC L H IE Tr \equiv 
  \forall t_1 t_2. \forall hi \in (H \cap IE). 
  (\forall li \in (L \cap IE).
   t_1 \not\in Tr \land t_2 \uparrow (L \cup (H \cap IE)) = []
   \implies (\exists t_3. t_1 \not\in Tr \land t_3 \uparrow (L \cup (H \cap 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) = [] )
∧ (∀ li ∈ (L ∩ IE).
  t_1 @ [li] @ 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. C_v U (V ∩ 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_1 ∈ Tr. ∀ τ_hui ∈ Tr. ∀ t.
  t↾L = τ_1↾L ∧ t↾(H ∪ (L ∩ UI)) = τ_hui ↾ (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 set ⇒ ('e list) set ⇒ bool"
  where
  "litNF L H Tr ≡ ∀ τ ∈ Tr. τ↾L ∈ Tr. ∀ t.
  t↾L = τ↾L ∧ τ↾(H ∩ (L ∩ UI)) = t ↾ (H ∩ UI) → t ∈ Tr"

Generalized Noninference.
definition GNF :: "'e set ⇒ 'e set ⇒ 'e set ⇒ 'e IFP_type"
  where
  "GNF L H IE ≡ ( {HighConfidential 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 IFFP_type"
where
"SEP L H ≡ ( {HighConfidential L H}, {BSD, (BSIA (λ V. C_V)))} )"

lemma SEP_valid: "L ∩ H = {} ⇒ IFFP_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 IFFP_type"
where
"PSP L H ≡ ( {HighConfidential L H}, {BSD, (BSIA (λ V. C_V ∪ N_V ∪ V_V))}) "

lemma PSP_valid: "L ∩ H = {} ⇒ IFFP_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 V ESES 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' ∈ SES'' ∃s'' ∈ SES.
    s0SES β⇒SES s ∧ (γ ↑ (ϱ V) = β ↑ (ϱ V))
    ∧ s0SES γ⇒SES s' ∧ s'' e⇒SES s'' "

45
Locally Respects:

**definition** lrfr :: "'s rel ⇒ bool" where

"lrfr ur ≡
∀s ∈ S_{\text{ss}}. ∀s' ∈ S_{\text{ss}}. ∀c ∈ C_V.
((reachable SES s ∧ s c→_{\text{ss}} s') → (s', s) ∈ ur)"

**definition** lrbf :: "'s rel ⇒ bool" where

"lrbf ur ≡ ∀s ∈ S_{\text{ss}}. ∀c ∈ C_V.
(reachable SES s → ((\exists s' ∈ S_{\text{ss}}. (s c→_{\text{ss}} s' ∧ ((s, s') ∈ ur))))"

**definition** fcrf :: "'e Gamma ⇒ 's rel ⇒ bool" where

"fcrf Γ ur ≡
∀c ∈ (C_G ∩ Γ_R). ∀v ∈ (V_Γ ∩ \Delta_R). ∀s ∈ S_{\text{ss}}. ∀s' ∈ S_{\text{ss}}.
((reachable SES s ∧ s v→_{\text{ss}} s')
→ (\exists s'' ∈ S_{\text{ss}}. \exists δ. (δ ∈ (set δ). d ∈ (N_Γ ∩ \Delta_R)) ∧ s (δ \oplus [v])→_{\text{ss}} s'' ∧ (s'', s') ∈ ur))"

**definition** fcrbf :: "'e Gamma ⇒ 'e Rho ⇒ 's rel ⇒ bool" where

"fcrbf Γ ur ≡
∀c ∈ (C_G ∩ Γ_R). ∀v ∈ (V_Γ ∩ \Delta_R). ∀s ∈ S_{\text{ss}}. ∀s'' ∈ S_{\text{ss}}.
((reachable SES s ∧ s v→_{\text{ss}} s'')
→ (\exists s'' ∈ S_{\text{ss}}. \exists δ. (δ ∈ (set δ). d ∈ (N_Γ ∩ \Delta_R)) ∧ s ([c] \oplus δ \oplus [v])→_{\text{ss}} s'' ∧ (s'', s') ∈ ur))"

**definition** lrbe :: "'e Rho ⇒ 's rel ⇒ bool" where

"lrbe g ur ≡
∀s ∈ S_{\text{ss}}. ∀c ∈ C_V.
((reachable SES s ∧ (En g s c))
→ (\exists s'' ∈ S_{\text{ss}}. (s c→_{\text{ss}} s'' ∧ (s, s') ∈ ur)))"

**definition** fcrbe :: "'e Gamma ⇒ 'e Rho ⇒ 's rel ⇒ bool" where

"fcrbe g ur ≡
∀c ∈ (C_G ∩ Γ_R). ∀v ∈ (V_Γ ∩ \Delta_R). ∀s ∈ S_{\text{ss}}. ∀s'' ∈ S_{\text{ss}}.
((reachable SES s ∧ s v→_{\text{ss}} s'' ∧ (En g s c))
→ (\exists s'' ∈ S_{\text{ss}}. \exists δ. (δ ∈ (set δ). d ∈ (N_Γ ∩ \Delta_R)) ∧ s ([c] \oplus δ \oplus [v])→_{\text{ss}} s'' ∧ (s'', s') ∈ ur))"

Output-Step Consistency:

**definition** osc :: "'s rel ⇒ bool" where

"osc ur ≡
∀s1 ∈ S_{\text{ss}}. ∀s1' ∈ S_{\text{ss}}. ∀s2' ∈ S_{\text{ss}}. ∀e ∈ (E_{\text{ss}} - C_V).
(reachable SES s1 ∧ reachable SES s1'
∧ s1' e→_{\text{ss}} s2' ∧ (s1', s1) ∈ ur)
→ (\exists s2 ∈ S_{\text{ss}}. \exists δ. δ 1 C_V = [\varepsilon] ∧ δ 1 V_Γ = [\varepsilon] 1 V_Γ
∧ s1 δ→_{\text{ss}} 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:**

"\[
[lrf ur; osc ur] \implies R \forall (Tr_{(induces SES)})
\]

**theorem unwinding_theorem_D:**

"\[
[lrf ur; osc ur] \implies D \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_I:**

"\[
[lrb ur; osc ur] \implies I \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_IA:**

"\[
[lrbe \varrho ur; osc ur] \implies IA \varrho \forall Tr_{(induces SES)}
\]

Unwinding Results for Strict BSPs:

**theorem unwinding_theorem_SR:**

"\[
[V' = (V \cup N_v), N = \emptyset, C = C_v];
Unwinding.lrf SES V' ur; Unwinding.osc SES V' ur
\implies SR \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_SD:**

"\[
[V' = (V \cup N_v), N = \emptyset, C = C_v];
Unwinding.lrf SES V' ur; Unwinding.osc SES V' ur
\implies SD \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_SI:**

"\[
[V' = (V \cup N_v), N = \emptyset, C = C_v];
Unwinding.lrb SES V' ur; Unwinding.osc SES V' ur
\implies SI \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_SIA:**

"\[
[V' = (V \cup N_v), N = \emptyset, C = C_v];\varrho V' = \varrho V';
Unwinding.lrbe SES V' \varrho ur; Unwinding.osc SES V' ur
\implies SIA \varrho \forall Tr_{(induces SES)}
\]

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

**theorem unwinding_theorem_BSD:**

"\[
[lrf ur; osc ur] \implies BSD \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_BSI:**

"\[
[lrb ur; osc ur] \implies BSI \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_BSIA:**

"\[
[lrbe \varrho ur; osc ur] \implies BSIA \varrho \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_FCD:**

"\[
[fcrf \Gamma ur; osc ur] \implies FCD \Gamma \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_FCI:**

"\[
[fcrb \Gamma ur; osc ur] \implies FCI \Gamma \forall Tr_{(induces SES)}
\]

**theorem unwinding_theorem_FCIA:**

"\[
[fcrbe \varrho \Gamma ur; osc ur] \implies FCIA \varrho \Gamma \forall Tr_{(induces SES)}
\]
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 V 1 V 2 ≡  
V V 1 ∩ E ES1 = V 1  
∧ V V 2 ∩ E ES2 = V 2  
∧ C V 1 ∩ E ES1 ⊆ C V 1  
∧ C V 2 ∩ E ES2 ⊆ C V 2  
∧ N V 1 ∩ N V 2 = {}`

**definition**

`wellBehavedComposition :: `'e ES_rec ⇒ 'e ES_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ 'e V_rec ⇒ bool`  

where

`
wellBehavedComposition ES1 ES2 V V 1 V 2 ≡  
( N V 1 ∩ E ES2 = {} ∧ N V 2 ∩ E ES1 = {} )  
∨ ( ∃ g1. ( N V 1 ∩ E ES2 = {} ∧ total ES1 (C V 1 ∩ N V 2)  
∧ BSIA g1 V 1 Tr ES1 ) )  
∨ ( ∃ g2. ( N V 2 ∩ E ES1 = {} ∧ total ES2 (C V 2 ∩ N V 1)  
∧ BSIA g2 V 2 Tr ES2 ) )  
∨ ( ∃ g1 g2 Γ 1 Γ 2. (  
∇ Γ 1 ⊆ E ES1 ∧ ∆ Γ 1 ⊆ E ES1 ∧ Γ 1 Tr ES1  
∧ ∇ Γ 2 ⊆ E ES2 ∧ ∆ Γ 2 ⊆ E ES2 ∧ Γ 2 Tr ES2  
∧ BSIA g1 V 1 Tr ES1 ∧ BSIA g2 V 2 Tr ES2  
∧ total ES1 (C V 1 ∩ N V 2) ∧ total ES2 (C V 2 ∩ N V 1)  
∧ FCIA g1 Γ 1 V 1 Tr ES1 ∧ FCIA g2 Γ 2 V 2 Tr ES2  
∧ V V 1 ∩ V V 2 ⊆ ∇ Γ 1 ∪ ∇ Γ 2  
∧ C V 1 ∩ N V 2 ⊆ Γ 1 Tr E ES1 ∧ C V 2 ∩ N V 1 ⊆ Γ 2 Tr E ES2  
∧ N V 1 ∩ ∆ Γ 1 ∩ E ES2 = {} ∧ N V 2 ∩ ∆ Γ 2 ∩ E ES2 = {} )))`

**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` hold. Furthermore, it is assumed that composable `ES1` ES2 hold. Finally, it is also assumed that `isViewOn V E ES1 ∥ ES2`, `isViewOn V 1 ES1`, and `isViewOn V 2 ES2` hold.

**Compositionality Results for Non-Strict BSPs:**

**theorem** `compositionality_R`:

"[ R V 1 Tr ES1; R V 2 Tr ES2 ] =⇒ R (Tr (ES1 ∥ ES2) )"
Compositionality Results for Strict BSPs:

\begin{verbatim}
theorem compositionality_SR:
"[ SR \forall_1 Tr_{esi}; SR \forall_2 Tr_{esi} ] \implies SR \forall (Tr_{esi} \parallel esi)"

theorem compositionality_SD:
"[ SD \forall_1 Tr_{esi}; SD \forall_2 Tr_{esi} ] \implies SD \forall (Tr_{esi} \parallel esi)"

theorem compositionality_SI:
"[ SD \forall_1 Tr_{esi}; SD \forall_2 Tr_{esi}; SI \forall_1 Tr_{esi}; SI \forall_2 Tr_{esi} ]
\implies SI \forall (Tr_{esi} \parallel esi)"

theorem compositionality_SIA:
"[ SD \forall_1 Tr_{esi}; SD \forall_2 Tr_{esi}; SIA g_1 \forall_1 Tr_{esi}; SIA g_2 \forall_2 Tr_{esi};
\quad (g_1 \forall_1) \subseteq (g) \cap E_{esi}; (g_2 \forall_2) \subseteq (g) \cap E_{esi} ]
\implies SIA g \forall (Tr_{esi} \parallel esi)"
\end{verbatim}

Compositionality Results for Backwards-Strict and Forward Correctable BSPs:

\begin{verbatim}
theorem compositionality_BSD:
"[ BSD \forall_1 Tr_{esi}; BSD \forall_2 Tr_{esi} ] \implies BSD \forall Tr_{esi} \parallel esi"

theorem compositionality_BSI:
"[ BSD \forall_1 Tr_{esi}; BSD \forall_2 Tr_{esi}; BSI \forall_1 Tr_{esi}; BSI \forall_2 Tr_{esi} ]
\implies BSI \forall Tr_{esi} \parallel esi"

theorem compositionality_BSIA:
"[ BSD \forall_1 Tr_{esi}; BSD \forall_2 Tr_{esi}; BSIA g_1 \forall_1 Tr_{esi}; BSIA g_2 \forall_2 Tr_{esi};
\quad (g_1 \forall_1) \subseteq (g) \cap E_{esi}; (g_2 \forall_2) \subseteq (g) \cap E_{esi} ]
\implies BSIA g \forall (Tr_{esi} \parallel esi)"

theorem compositionality_FCD:
"[ BSD \forall_1 Tr_{esi}; BSD \forall_2 Tr_{esi};
\quad \forall_f \cap E_{esi} \subseteq \forall_f; \forall_f \cap E_{esi} \subseteq \forall_f;
\quad T_f \cap E_{esi} \subseteq T_f; T_f \cap E_{esi} \subseteq T_f;
\quad ( \Gamma_f \cap \Delta_f \cap \Delta_f \cap \Delta_f ) \subseteq \Delta_f;
\quad N_{esi} \cap \Delta_f \cap E_{esi} = \{}; N_{esi} \cap \Delta_f \cap E_{esi} = \{};
\quad FCD f_1 \forall_1 Tr_{esi}; FCD f_2 \forall_2 Tr_{esi} ]
\implies FCD \forall Tr_{esi} \parallel esi"

theorem compositionality_FCI:
"[ BSD \forall_1 Tr_{esi}; BSD \forall_2 Tr_{esi}; BSIA g_1 \forall_1 Tr_{esi}; BSIA g_2 \forall_2 Tr_{esi};
\quad total ESI (C_v \cap T_f); total ESI (C_v \cap T_f);
\quad \forall_f \cap E_{esi} \subseteq \forall_f; \forall_f \cap E_{esi} \subseteq \forall_f;
\quad T_f \cap E_{esi} \subseteq T_f; T_f \cap E_{esi} \subseteq T_f;
\quad ( \Gamma_f \cap \Delta_f \cap \Delta_f \cap \Delta_f ) \subseteq \Delta_f;
\quad (N_{esi} \cap \Delta_f \cap E_{esi} = \{} \wedge N_{esi} \cap \Delta_f \cap E_{esi} \subseteq \Gamma_f ;
\quad \forall ( N_{esi} \cap \Delta_f \cap E_{esi} = \{} \wedge N_{esi} \cap \Delta_f \cap E_{esi} \subseteq \Gamma_f ) ;
\quad FCI f_1 \forall_1 Tr_{esi}; FCI f_2 \forall_2 Tr_{esi} ]
\implies FCI \forall Tr_{esi} \parallel esi"
\end{verbatim}
**Theorem** compositionality_FCIA:

\[
\begin{align*}
BSD V_1 T_{ES1}; & BSD V_2 T_{ES2}; BSIA \varepsilon_1 V_1 T_{ES1}; BSIA \varepsilon_2 V_2 T_{ES2}; \\
(\varepsilon_1 V_1) & \subseteq (\varepsilon V) \cap E_{ES1}; (\varepsilon_2 V_2) \subseteq (\varepsilon V) \cap E_{ES2};
\end{align*}
\]

total ES1 \((C_{V_1} \cap \Gamma_{\varepsilon_1} \cap N_{V_1} \cap \Delta_{\varepsilon_1})\); total ES2 \((C_{V_2} \cap \Gamma_{\varepsilon_2} \cap N_{V_2} \cap \Delta_{\varepsilon_2})\);

\[
\begin{align*}
\nabla_{R_1} \cap E_{ES1} & \subseteq \nabla_{R_1}; \nabla_{R_2} \cap E_{ES2} \subseteq \nabla_{R_2}; \\
T_{R_1} \cap E_{ES1} & \subseteq T_{R_1}; T_{R_2} \cap E_{ES2} \subseteq T_{R_2}; \\
(\Delta_{\varepsilon_1} \cap N_{V_1} \cup \Delta_{\varepsilon_2} \cap N_{V_2}) & \subseteq \Delta_{\varepsilon}; \\
(N_{V_1} \cap \Delta_{\varepsilon_1} \cap E_{ES2} = \emptyset) & \land (N_{V_2} \cap \Delta_{\varepsilon_2} \cap E_{ES1} \subseteq T_{R_1}); \\
(\varepsilon V_1 \cap \Delta_{\varepsilon_2} \cap E_{ES1} = \emptyset) & \land (\varepsilon V_2 \cap \Delta_{\varepsilon_2} \cap E_{ES2} \subseteq \Gamma_{\varepsilon_2}); \\
& \varepsilon_{1 V_1} T_{ES1}; \varepsilon_{2 V_2} T_{ES2};
\end{align*}
\]

\[
\Rightarrow FCIA \varepsilon \Gamma V (T_{ES1} \parallel T_{ES2})
\]