Possibilistic Definitions of Security – An Assembly Kit –

Heiko Mantel

German Research Center for Artificial Intelligence (DFKI),
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
E-mail: mantel@dfki.de

Abstract

We present a framework in which different notions of security can be defined in a uniform and modular way. Each definition of security is formalized as a security predicate by assembling more primitive basic security predicates. A collection of such basic security predicates is defined and we demonstrate how well-known concepts like generalized non-interference or separability can be constructed from them. The framework is open and can be extended with new basic security predicates using a general schema. We investigate the compatibility of the assembled definitions with system properties apart from security and propose a new definition of security which does not restrict non-critical information flow. It turns out that the modularity of our framework simplifies these investigation. Finally, we discuss the stepwise development of secure systems.

1. Introduction

Non-interference has become a popular concept for formalizing security. Its main benefit over access control models like the one by Bell and LaPadula [1] or Biba [2] is that it provides a definition of security rather than only a mechanism for enforcing it. The initial work on non-interference by Goguen and Meseguer [6, 7] was limited to deterministic systems. Beginning with Sutherland [18], various generalizations of non-interference for non-deterministic systems have been proposed, e.g. [9, 12, 20]. In this article, we consider confidentiality aspects of security and focus on *possibilistic definitions of security*. The underlying idea of this approach is that information cannot be deduced by observing a system because the set of possible behaviours which may have generated a given observation is too large. For a discussion of benefits and limitations of the possibilistic approach we refer to [11, 13].

The variety of definitions of possibilistic security indicates that there might not be a definition which is optimal for all purposes. Rather, the choice of such a definition depends on the particular application. This demands for a uniform framework in which one can compare different definitions and choose the most appropriate one. McLean's selective interleaving functions [12, 14] provide such a framework in which security can be defined by closure conditions, but Zakinthinos and Lee pointed out that the expressiveness of his framework is too limited [20]. Their framework overcomes this limitation and allows them to define a perfect security property *PSP*. However, the general correspondence between closure conditions and security is lost. The framework presented in this article seeks to combine the expressiveness of the one by Zakinthinos and Lee with the elegance of McLean's framework while overcoming their limitations.

One novelty of our framework is its modular structure. It consists of a collection of *basic security predicates* which can be combined to *security predicates* in order to define a notion of security. Thus, our framework really is an *assembly kit* for such definitions in which basic security predicates are the building blocks. This modularity also reduces the complexity of reasoning about security, because many properties of the building blocks are preserved under combinations. New basic building blocks can be defined using a general schema which ensures that each basic security predicate corresponds to a closure condition.

This correspondence forms the theoretical basis for an investigation of the *compatibility of security with other system properties* which is an important criterion in the selection of a definition of security. As examples, we investigate properties which require certain types of non-critical information flow. We demonstrate that *PSP* is incompatible with some of these properties and derive the *pretty good security predicate PGSP* as a less restrictive definition which allows the respective types of information flow.

The correspondence to closure conditions also provides a basis for a *stepwise development of secure systems*. Although, security is not preserved under refinement [9] in

In Proceedings of the 13th IEEE Computer Security Foundations Workshop, Cambridge, United Kingdom, pp.185-199, July 3-5, 2000. IEEE Computer Society.

general, we show that certain definitions of security are preserved under *intersection* which is a special case of refinement.

After some basic definitions in Section 2, we introduce our framework in Section 3. We present a collection of basic security predicates in Section 4 and show how to assemble these basic predicates into well-known definitions of possibilistic security in Section 5. In Section 6 we investigate the compatibility of security with other system properties and derive *PGSP* in our framework in order to overcome certain incompatibilities of *PSP*. In Section 7 we consider the preservation of security in a stepwise development process. We discuss our approach in relation to previous work in Section 8. Finally, in Section 9 we summarize our achievements and remark on future plans.

2. Preliminaries

Following [20] we model non-deterministic systems using event systems. Computation steps and interactions are modeled by *events*, i.e. actions without duration, like e.g. assigning a value to a variable or sending a message. We distinguish between *input*, *internal*, and *output events*. Input events are not controlled by the system but rather by some external environment while all other events are controlled by the system. The interface of a system is modeled by the input and output events and its executions by *traces*, i.e. sequences of events. Thus, a system can be specified by a set of traces which models its possible behaviours.

Definition 1. An *event system* ES is a tuple (E, I, O, Tr) where E is a set of events, $I \subseteq E$, $O \subseteq E$ respectively are the input and output events, and $Tr \in \mathcal{P}(E^*)$ is the set of traces. We denote the set of finite sequence over E by E^*. Each trace $t \in Tr$ is a finite sequence of events in E and Tr must be closed under prefixes.

For the definition of security policies we assume a set $\mathcal{D}$ of *security domains* such that each such domain is an abstraction from concrete entities, like e.g. individual users, groups of users, processes, or collections of files. A security domain $dom(e)$ is associated with each event e using a function $dom : E \to \mathcal{D}$. A *non-interference relation*, an irreflexive relation $\not\rightsquigarrow : \mathcal{D} \times \mathcal{D}$, can then be used to specify which information flows are restricted between domains, e.g. $D_1 \not\rightsquigarrow D_2$ states that D_1 *must not interfere with* D_2 for domains D_1, D_2. Finally, a *security policy Pol* is a triple $(\mathcal{D}, dom, \not\rightsquigarrow)$. *Pol* is called *transitive* if $\rightsquigarrow$, the complement of $\not\rightsquigarrow$, is transitive. Unless explicitely stated otherwise, we consider transitive security policies in this article.

For transitive security policies, in principle, it suffices to consider two domains H and L only, a high and a low-level. In the subsequent sections we will make use of this abstraction and consider the *two-level security policy* Pol_{HL} with domains H and L and the non-interference relation which demands $H \not\rightsquigarrow L$.

In order to prove that a systems satisfies a given security policy it is necessary to define formally what is meant by "D_1 does not interfere with D_2", i.e. $\not\rightsquigarrow$ must be given a semantics in terms of event systems. The formal definition of this phrase can be regarded as a *definition of security*. Consequently a *security property* is defined by a security policy together with a definition of security. *Non-inference*, *generalized non-interference*, and *restrictiveness* are well-known examples among the various possibilistic security properties which have been proposed for non-deterministic systems.

When proving that a system satisfies a given security policy one shows for each domain that the non-interference relation is respected. Since the security policy Pol_{HL} is fixed we can focus on the different definitions of security in the remainder of this article. Therefore, we use the term "definition of security" instead of "security property".

2.1. Notational Conventions

We assume that ES is an event system (E, I, O, Tr) where E denotes a set of events, I and O, respectively, the sets of input and output events in E, and Tr a set of traces over E. Individual events are denoted by e, sequences of events by α or β and by τ or t – if they are traces. A dot concatenates events to form sequences, e.g. $e_1.e_2.e_3$. We deliberately use a dot also for appending sequences, e.g. $\alpha.\beta$, $\alpha.e$, and $\alpha.e.\beta$. The empty sequence is denoted by $\langle\rangle$ and the *projection* of a given trace t to a set of events $E' \subseteq E$ by $t|_{E'}$. $t|_{E'}$ results from t by deleting all events not in E'.

We use H and L, the names of the high- and low-level domains, also to denote the subsets of events in E with domain H and L. HI, LI, HO, and LO denote the corresponding sets of input and output events. Events in L, H, HI, and HO are denoted respectively by l, h, hi, and ho. Subscripts and primes are used in combination with all of these denotations.

3. A Framework for Possibilistic Security

The confidentiality of classified information can only be ensured if direct as well as indirect flows of information are restricted. An observer must neither directly observe information for which he does not possess the appropriate clearance nor be able to deduce such information from other observations. In order to prevent direct information flow certain aspects of the system behaviour must not be observable. Here, we assume that only low-level events are observable on the low-level, i.e. for a trace τ the sequence $\tau|_L$ can be observed. However, in the worst case an observer who has complete knowledge of the system, can construct all system behaviours which generate a given observation, and try

to deduce confidential information from this set. Formally, such an observer constructs the *low-level equivalence set* $LLES(Tr,\tau) = \{t \in Tr \mid t|_L = \tau|_L\}$ (introduced in [20]) from the observation $\tau|_L$ and the knowledge about the system.

The underlying idea of possibilistic security is to demand that $LLES(Tr,\tau)$ is so large that an observer cannot deduce confidential information from it. Any trace in $LLES(Tr,\tau)$ could have generated the observation $\tau|_L$ and, unless τ is the only element in $LLES(Tr,\tau)$, one cannot deduce that τ has actually occurred.[1] However, even if $LLES(Tr,\tau)$ has more than one element, the deduction of confidential information may still be possible. If certain high-level behaviours are compatible with a given observation but others are not then one deduces from this observation that one of the former high-level behaviours has occurred but not any of the latter ones. Such channels may cause a system to be insecure because they could be exploited by Trojan horses for example. In order to avoid this, a possibilistic security property demands that if a given high-level behaviour is compatible with some observation then certain other high-level behaviours must be compatible with it as well. Thus, $LLES(Tr,\tau)$ must be *closed* wrt. some criterion. This gives rise to a general correspondence between possibilistic security properties and closure conditions.

In our framework, possibilistic security properties are represented in a modular way as *security predicates*. Formally, a security predicate *SP* is either a single *basic security predicate BSP*, i.e. $SP \equiv BSP$ or a conjunction of basic security predicates, i.e. $SP \equiv BSP_1 \wedge \ldots \wedge BSP_n$. Each basic security predicate *BSP* demands that for any trace τ of the system there must be another trace τ' which is compatible with the same observation and which fulfills a condition Q, the *closure requirement of BSP*. The existence of τ', however, is only required if a condition R, the *restriction of BSP*, holds. This results in the following schema for the formal definition of basic security predicates:

$$\forall \tau \in Tr.R(Tr,\tau) \Rightarrow \exists \tau' \in LLES(Tr,\tau).Q(\tau,\tau') \ .$$

Since there is a general correspondence between possibilistic security properties and closure conditions, as explained above, it is desirable to establish such a correspondence for basic security predicates (abbreviated by BSP in the sequel) as well. In order to achieve this, we have to impose an additional requirement on the definition of BSPs. This *satisfiability condition* demands that $R(Tr,\tau) \Rightarrow \exists \tau' \in E^*.\tau'|_L = \tau|_L \wedge Q(\tau,\tau')$ is satisfiable for any $\tau \in Tr$. Note that τ' need not be in $LLES(Tr,\tau)$, not even in *Tr*, but it must yield the same observation as τ. The condition ensures that a BSP can be made valid for *Tr* by adding elements to *Tr*, i.e. by constructing a closure of *Tr*. This is essential for the construction of closure operations from BSPs in Subsection 4.3. The definition of a BSP with our schema states when a set of traces is closed wrt. some criterion, while the satisfiability condition ensures that for any set of traces such a closure can be constructed.

Technically we use a slightly more complicated schema for the definition of BSPs which allows for inductive definitions. This is achieved by additional variables $\alpha, \beta \in E^*$ and $e \in E$ in the schema which are universally quantified. These auxiliary variables are used to divide τ into subsequences, like $\beta.e.\alpha$ or $\beta.\alpha$, in the restriction R and to use the same division in the closure requirement Q.

Definition 2. The *basic security predicate* BSP_{RQ} for the *restriction* R and the *closure requirement* Q is defined by

$$\forall \tau \in Tr.\forall \alpha, \beta \in E^*.\forall e \in E.$$
$$[R(Tr,\tau,\alpha,\beta,e) \Rightarrow \exists \tau' \in LLES(Tr,\tau).Q(\tau,\tau',\alpha,\beta,e)].$$

We require that $R(Tr,\tau,\alpha,\beta,e) \Rightarrow \exists \tau' \in E^*.\tau'|_L = \tau|_L \wedge Q(\tau,\tau',\alpha,\beta,e)$, the *satisfiability condition*, can be satisfied for all $Tr \in \mathcal{P}(E^*)$, $\tau \in Tr$, $\alpha, \beta \in E^*$, and $e \in E$. Note that τ' need not be in *Tr*.

Definition 3. A *security predicate SP* is either a single basic security predicate or a conjunction of multiple basic security predicates.

Possibilistic security properties are represented as security predicates within our framework in a modular way. Definition 2 and 3 enforce a certain structure for such representations. In particular, this structure ensures that the correspondence between possibilistic security and closure conditions also exists in the framework. The schema for the definition of BSPs allows for inductive definitions. This enables us to distinguish two dimensions of BSPs when we instantiate the schema for BSPs (cf. Section 4). The expressiveness of our framework allows for a representation of the well-known possibilistic security properties (cf. Section 5). The correspondence to closure conditions is the basis for the investigation of the compatibility of such security properties with other system properties (cf. Section 6) and for a stepwise development of secure systems (cf. Section 7).

4. Basic Security Predicates

In this section, we illustrate how to instantiate our schema for the definition of BSPs and present a collection of BSPs which will be used as examples in the remainder of the article. Two dimensions of BSPs are distinguished in Subsection 4.1 and 4.2 and, if possible, the BSPs in each dimension are ordered by implication. BSPs in the first dimension express that it is confidential that an event has

[1] The possibilistic approach prevents certainty about deduced information and abstracts from probabilities.

occurred. Formally this corresponds to the possibility to delete this event without changing the observation. In the second dimension it is confidential that an event has *not* occurred which corresponds to the possibility to insert events. All BSPs presented, correspond to closure conditions. Our schema for BSPs ensures this correspondence for BSPs in general as we will show in Subsection 4.3. The distinction of two dimensions together with the ordering of BSPs helps to compare the various BSPs easily and provides useful orientation when BSPs are selected for the construction of security predicates.

Recall that we assume that only low-level events are observable for a low-level user. However, not all high-level events need to be confidential. Depending on the particular application, a BSP may be appropriate which ensures the confidentiality of only some high-level events and does not care if an occurrence of the others can be deduced. Nevertheless, in other cases it may be necessary to consider all high-level events as confidential. In this article, we restrict our considerations to two scenarios in which either all high-level events or (only) high-level inputs are confidential. An adaption of the definitions to other scenarios, where e.g. high-level outputs are confidential, is possible.

BSPs based on removal of events:

$RE(Tr) \equiv \tau'|_H = \langle\rangle$

$RI(Tr) \equiv \tau'|_{HI} = \langle\rangle$

$SRI(Tr) \equiv \tau'|_{HI} = \langle\rangle \wedge \tau'|_{(E\setminus HI)} = \tau|_{(E\setminus HI)}$

BSPs based on stepwise deletion of events:

$DE(Tr) \equiv (e \in H \wedge \tau = \beta.e.\alpha \wedge \alpha|_H = \langle\rangle) \Rightarrow (\tau' = \beta.\alpha)$

$DI(Tr) \equiv (e \in HI \wedge \tau = \beta.e.\alpha \wedge \alpha|_{HI} = \langle\rangle) \Rightarrow (\tau' = \beta'.\alpha' \wedge \beta'|_{L\cup HI} = \beta|_{L\cup HI} \wedge \alpha'|_{L\cup HI} = \alpha|_{L\cup HI})$

$BSDI(Tr) \equiv (e \in HI \wedge \tau = \beta.e.\alpha \wedge \alpha|_{HI} = \langle\rangle) \Rightarrow (\tau' = \beta.\alpha' \wedge \alpha'|_{L\cup HI} = \alpha|_{L\cup HI})$

$SDI(Tr) \equiv (e \in HI \wedge \tau = \beta.e.\alpha \wedge \alpha|_{HI} = \langle\rangle) \Rightarrow (\tau' = \beta.\alpha)$

Unprimed variables (τ, α, β, e) are universally and primed ones (τ', α', β') are existentially quantified. τ' must be in $LLES(Tr, \tau)$.

Figure 1. BSPs based on removal or stepwise deletion of high-level events

4.1. Removal and Stepwise Deletion of Events

BSPs which prevent a low-level user from inferring that certain high-level events have occurred are based on the possibility to remove these events from traces while preserving the resulting observation. In the sequel, we distinguish between BSPs which require the *removal* of all these events at once and BSPs which require their *stepwise deletion*. Each of these BSPs is defined using the schema from Definition 2 such that the satisfiability condition is fulfilled.

When defining BSPs, we employ a uniform naming scheme. If e.g. a BSP is based on removal or deletion of events then an 'R' or 'D' respectively occurs in its name. If only high-level inputs are affected then 'I' is used and if all high-level events are affected then 'E' is used instead.

The basic security predicates *RE* (Removal of Events), *RI* (Removal of Inputs), and *SRI* (Strict Removal of Inputs) require a global removal of events. The BSP *RE* demands the removal of high-level events. Given $\tau \in Tr$, it assures the existence of a low-level equivalent trace τ' such that $\tau'|_H$ is empty. τ' results from τ by removing *all* high-level events, e.g. if τ is $l_1.ho_1.hi_1.l_2$, then $\tau' = l_1.l_2$ must be a trace. In Figure 1, we use a shorthand notation to abbreviate

$$RE(Tr) \equiv \forall \tau \in Tr.\forall \alpha, \beta \in E^*.\forall e \in E.\mathsf{TRUE} \Rightarrow \exists \tau' \in LLES(Tr, \tau).\tau'|_H = \langle\rangle \ .$$

In the shorthand notation, we omit quantifiers since they are clear from the schema for defining BSPs and just write $R \Rightarrow Q$. Unprimed variables are implicitly universally and primed ones existentially quantified. If R holds trivially we write Q instead of $\mathsf{TRUE} \Rightarrow Q$. For *RI* and *SRI*, τ' results from τ by removing all high-level inputs. While *RI* allows τ' to differ from τ in high-level internal and output events, *SRI* does not. Hence, *SRI* is called *strict*. For the example trace, *RI* would be satisfied if any of $l_1.ho_1.l_2$ or $l_1.l_2$ occurs in *Tr* while *SRI* is satisfied only by the first trace.

The basic security predicates *DE* (Deletion of Events), *DI* (Deletion of Inputs), *SDI* (Strict Deletion of Inputs), and *BSDI* (Backwards Strict Deletion of Inputs) are based on the stepwise deletion of events. These BSPs are inductively defined. A trace τ requires the existence of a trace τ' which results from τ by deleting a single event, τ' may require the existence of a trace τ'' which results by the deletion of another event, and so on. In the definition of these BSPs the auxiliary variables α, β, and e are used in order to divide τ into subsequences. Similarly, τ' is divided using additional auxiliary variables $\alpha', \beta' \in E^*$ which are existentially quantified in the respective closure requirement. In Figure 1 the quantifiers for the latter auxiliary variables are also omitted. Note that these variables do not occur in the restrictions of the BSPs.

For *DE*, τ' results from τ by the deletion of the last high-level event, e.g. if $l_1.ho_1.hi_1.l_2 \in Tr$ then $l_1.ho_1.l_2$ as well as $l_1.l_2$ must also be in *Tr*. For *DI*, *BSDI*, and *SDI*, τ' results from τ by the deletion of the last high-level input event e. While in *DI*, α' and β' may differ, respectively, from α and β in high-level internal and output events (but not in high-level inputs or low-level events), in *BSDI*

only α' may differ, and in *SDI* none of them may differ. If e.g. $l_1.ho_1.hi_1.ho_2.l_2 \in Tr$ then *DI* would be satisfied if any of $l_1.ho_1.ho_2.l_2$, $l_1.ho_1.l_2$, or $l_1.l_2$ is in *Tr*, *BSDI* only for any of the first two, and *SDI* only for the first trace. Hence, *BSDI* is called *backwards strict* and *SDI* strict.

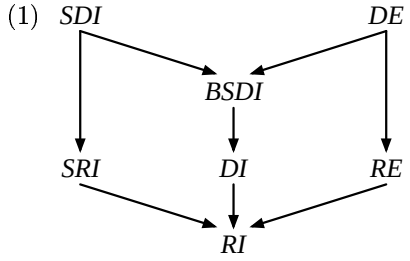

Figure 2. Ordering BSPs based on removal or stepwise deletion of high-level events

Theorem 4. *RE, RI, SRI, DE, DI, BSDI, and SDI are ordered by implication as depicted in Figure 2.*

Proof. *SDI* $\Rightarrow$ *SRI* and *DI* $\Rightarrow$ *RI* are proved by induction on the number of high-level input events, *DE* $\Rightarrow$ *RE* by induction on the number of high-level events, and *DE* $\Rightarrow$ *BSDI* by induction on the number of high-level output events in α. All other implications are trivial. □

Note that there is no link between *SDI*/*SRI* and *DE*/*RE* in Figure 2. Informally, the reason is that *SDI* and *SRI* require that high-level output and internal events are preserved when high-level input events are deleted/removed while *DE* and *RE* require that output and internal events are deleted/removed as well. This is reflected by the following counterexamples. For *DE* $\not\Rightarrow$ *SRI* consider the trace set $Tr_1 = \{hi.ho, hi, \langle\rangle\}$ for which $DE(Tr_1)$ holds but $SRI(Tr_1)$ does not hold because $ho \notin Tr_1$. For *SDI* $\not\Rightarrow$ *RE* consider the trace set $Tr_2 = \{ho.l, ho, \langle\rangle\}$ for which $SDI(Tr_2)$ holds but $RE(Tr_2)$ does not hold because $l \notin Tr_2$.

The BSPs presented in this subsection have different motivations. Some BSPs, like *RE* or *RI*, are building blocks of well-known definitions of security (cf. Section 5) while others, like *SRI*, *SDI*, or *DE*, help us to prove properties of security predicates (cf. Section 6). *BSDI* and *DI* have been added mainly for reasons of uniformity.

4.2. Stepwise Insertion of Events

BSPs which prevent the low-level user from inferring that certain high-level events have not occurred, are based on the possibility to insert these events into traces while preserving the resulting low-level observation. We define BSPs which require the *stepwise insertion* of these events.

Again, we use a uniform naming scheme for BSPs. For all BSPs based on the stepwise insertion of events, an 'I' occurs in the name. If only high-level inputs are inserted then an additional 'I' is used and if all kinds of high-level events are inserted, then 'E' is used instead.

BSPs based on insertion of events:

$$
\begin{aligned}
IE(Tr) &\equiv (e \in H \wedge \tau = \beta.\alpha \wedge \alpha|_H = \langle\rangle) \\
&\quad \Rightarrow (\tau' = \beta.e.\alpha) \\
II(Tr) &\equiv (e \in HI \wedge \tau = \beta.\alpha \wedge \alpha|_{HI} = \langle\rangle) \\
&\quad \Rightarrow (\tau' = \beta'.e.\alpha' \wedge \beta'|_{L \cup HI} = \beta|_{L \cup HI} \\
&\qquad \wedge \alpha'|_{L \cup HI} = \alpha|_{L \cup HI}) \\
BSII(Tr) &\equiv (e \in HI \wedge \tau = \beta.\alpha \wedge \alpha|_{HI} = \langle\rangle) \\
&\quad \Rightarrow (\tau' = \beta.e.\alpha' \wedge \alpha'|_{L \cup HI} = \alpha|_{L \cup HI}) \\
SII(Tr) &\equiv (e \in HI \wedge \tau = \beta.\alpha \wedge \alpha|_{HI} = \langle\rangle) \\
&\quad \Rightarrow (\tau' = \beta.e.\alpha)
\end{aligned}
$$

BSPs based on insertion of hl-admissible events:

$$
\begin{aligned}
IHAE(Tr) &\equiv (R_{IE} \wedge HAdm_H(Tr, \beta, e)) \Rightarrow Q_{IE} \\
IHAI(Tr) &\equiv (R_{II} \wedge HAdm_{HI}(Tr, \beta, e)) \Rightarrow Q_{II} \\
BSIHAI(Tr) &\equiv (R_{BSII} \wedge HAdm_{HI}(Tr, \beta, e) \Rightarrow Q_{BSII} \\
SIHAI(Tr) &\equiv (R_{SII} \wedge HAdm_{HI}(Tr, \beta, e) \Rightarrow Q_{SII}
\end{aligned}
$$

BSPs based on insertion of admissible events:

$$
\begin{aligned}
IAE(Tr) &\equiv (R_{IE} \wedge Adm(Tr, \beta, e)) \Rightarrow Q_{IE} \\
IAI(Tr) &\equiv (R_{II} \wedge Adm(Tr, \beta, e)) \Rightarrow Q_{II} \\
BSIAI(Tr) &\equiv (R_{BSII} \wedge Adm(Tr, \beta, e) \Rightarrow Q_{BSII} \\
SIAI(Tr) &\equiv (R_{SII} \wedge Adm(Tr, \beta, e) \Rightarrow Q_{SII}
\end{aligned}
$$

Unprimed variables (τ, α, β, e) are universally and primed ones (τ', α', β') are existentially quantified. τ' must be in $LLES(Tr, \tau)$.

Figure 3. BSPs based on stepwise insertion of high-level events

The basic security predicates *IE* (Insertion of Events), *II* (Insertion of Inputs), *BSII* (Backwards Strict Insertion of Inputs), and *SII* (Strict Insertion of Inputs) require the stepwise insertion of events. These BSPs are, again, inductively defined. In Figure 3, the predicate *IE* demands that an arbitrary high-level event can be inserted into a trace. Given $\tau \in Tr$ which can be divided into $\beta.\alpha$ such that α contains no high-level events, it assures, the existence of a low-level equivalent trace $\tau' = \beta.e.\alpha$. τ' results from τ by inserting a high-level event e, e.g. if $l_1 \in Tr$ then $hi_1.l_1$ is also in *Tr* (among many other traces). For *II*, *BSII*, and *SII*, τ' results from τ by inserting a high-level input e. While in *II*, α' and β' may differ, respectively, from α and β in high-level internal and output events, in *BSII* only α' may differ, and in *SII* none of them may differ. Hence, *BSII* is called backwards strict and *SII* strict. Like for BSPs based on removal/deletion, the strict versions are motivated by their properties which help us to derive interesting theorems.

Although the definition of *IE* might appear appropriate

for the insertion of high-level events, it demands too much and makes any meaningful high-level behaviour impossible. A similar problem exists for *II*, *BSII*, and *SII*. The problem for *IE* is illustrated by the following example.

Example 5. We consider a system with two input events hi_1, hi_2, two output events ho_1, ho_2 on the high-level, and arbitrary low-level events. The purpose of the system is to record each high-level input by the corresponding output event, i.e. hi_1 by ho_1 and hi_2 by ho_2. $hi_1.ho_1$ is a possible trace of the system and therefore also the prefix hi_1 of this sequence. *IE* demands that $hi_1.ho_2$ must be a trace as well, however, this trace violates the intended system behaviour.

In general, *IE* rules out any meaningful system behaviour on the high-level. This problem can be solved by demanding that inserted events are admissible on the high-level, i.e. occur in some high-level behaviour. Note that this solution does not compromise security. In order to express high-level admissibility formally, we define $HAdm_H$ and $HAdm_{HI}$.

$$HAdm_H(Tr, \beta, e) \equiv \exists \gamma \in E^*.\gamma.e \in Tr \wedge \gamma|_H = \beta|_H$$
$$HAdm_{HI}(Tr, \beta, e) \equiv \exists \gamma \in E^*.\gamma.e \in Tr \wedge \gamma|_{HI} = \beta|_{HI}$$

$HAdm_H(Tr, \beta, e)$ ($HAdm_{HI}(Tr, \beta, e)$) holds if there is a trace $\gamma.e$ in *Tr* which has the same observations for high-level events (high-level input events) as $\beta.e$.

Adding high-level admissibility ($HAdm_H$ or $HAdm_{HI}$) to the restrictions of *IE*, *II*, *BSII*, and *SII* results in four new BSPs in Figure 3, *IHAE* (Insertion of High-level Admissible Events), *IHAI* (Insertion of Hl-Admissible Inputs), *BSIHAI* (Backwards Strict Insertion of Hl-Admissible Inputs), and *SIHAI* (Strict Insertion of Hl-Admissible Inputs). In the figure we use a shorthand notation and abbreviate the restriction and closure requirement of a previously defined BSP respectively by R_{BSP} and Q_{BSP}.

To require high-level admissibility solves the problem pointed out previously. However, the resulting BSPs are still too strong since they prevent information flow from the low- to the high-level which we demonstrate by example.

Example 6. Like in Example 5, a system is considered which records input events. However, this time low-level inputs are recorded in the high-level output. $li_1.ho_1$ and $li_2.ho_2$ are possible traces of the system and so is the prefix li_1 of the first trace. Since ho_2 is a possible high-level behaviour, *IHAE* demands that $li_1.ho_2$ is a trace of the system, however, this trace violates the intended system behaviour.

This problem can be solved by demanding the stronger admissibility condition $Adm(Tr, \beta, e) \equiv \beta.e \in Tr$.

Replacing high-level admissibility, $HAdm_H$ or $HAdm_{HI}$, by general admissibility, *Adm*, in the definition of *IHAE*, *IHAI*, *BSIHAI*, and *SIHAI* yields the BSPs, *IAE* (Insertion of Admissible Events), *IAI* (Insertion of Admissible Inputs), *BSIAI* (Backwards Strict Insertion of Admissible Inputs), and *SIAI* (Strict Insertion of Admissible Inputs) in Figure 3.

IAI (*BSIAI*, *SIAI*) and *IAE* demand that a low-level user cannot infer that an admissible high-level input or arbitrary high-level event has *not* occurred. Although *II*, *BSII*, and *SII* have similar problems like *IE*, they are reasonable predicates if one assumes *input totality* (as e.g. done in [20]), i.e. $ITOT(Tr) \equiv \forall \tau \in Tr.\forall i \in I.\tau.i \in Tr$. Under this assumption, input is always admissible at the end of a trace and the difference between e.g. *II*, *IHAI*, and *IAI* disappears. High-level admissibility appears to be inferior to *Adm*. However, for example generalized non-interference or separability are based on this concept as we will show in Section 5.

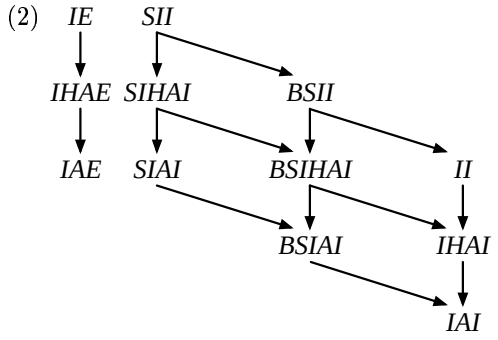

Figure 4. Ordering BSPs based on insertion

Theorem 7. *IE, II, BSII, SII, IHAE, IHAI, BSIHAI, SIHAI, IAE, IAI, BSIAI, and SIAI are ordered by implication as depicted in Figure 4.*

Note that there is no link between *IE*/*IHAE*/*IAE* and the BSPs which require the insertion of high-level input events in Figure 4. This is reflected by the following counterexamples. For $IE \not\Rightarrow IAI$ consider the trace set $Tr_1 = \{ho.l, ho, hi, \langle\rangle\}$ and the closure Tr_2 of Tr_1 under *IE*. $IE(Tr_2)$ holds by construction but $IAI(Tr_2)$ does not because $hi.ho.l \notin Tr_2$. To construct the other counterexamples is straightforward.

Remark 8. We have claimed at the beginning of this section that our collection of BSPs can be easily extended based on the two scenarios which we have considered here, i.e. all events in H or only events in HI are confidential. Technically, this can be achieved by replacing HI with some other set H' of high-level events in Figure 1 and 3. This yields BSPs for the case where events in H' are confidential rather than HI. Theorem 4 and 7 can be adapted accordingly. This allows for the easy extension of our collection of BSPs such that e.g. the case where all high-level outputs or only certain high-level events are confidential can be covered.

4.3. Induced Closure Operations

There is a correspondence between BSPs and closure operations. A closure operation constructs the closure of a given set by adding elements and a closure condition demands for a set that it is closed wrt. some criterion. Recall that this is also the underlying idea of possibilistic security. If a high-level behaviour is compatible with a given observation then certain other high-level behaviours must be compatible with it as well. For a basic introduction to closure operations and closure systems we refer to [3].

Definition 9. A *closure operation* $Cl : \mathcal{P}(S) \to \mathcal{P}(S)$ for a set S, briefly S*-closure*, is a function for which the following conditions hold.

Cl1 $S_1 \subseteq Cl(S_1)$ for any $S_1 \subseteq S$

Cl2 $S_1 \subseteq S_2 \Rightarrow Cl(S_1) \subseteq Cl(S_2)$ for any $S_1, S_2 \subseteq S$

Cl3 $Cl(Cl(S_1)) = Cl(S_1)$ for any $S_1 \subseteq S$

An ordering $\preceq$ on S-closures is defined by $Cl_1 \preceq Cl_2 \Leftrightarrow \forall S' \in \mathcal{P}(S).Cl_1(S') \subseteq Cl_2(S')$.

Now we are able to identify the correspondence between BSPs and closure operations formally. This correspondence is the basis for many of the subsequent considerations and results. Note that the satisfiability condition in Definition 2 is necessary in order to achieve this correspondence.

Definition 10. A *closure operation Cl for E^* ensures a basic security predicate BSP* if for any set *Tr* of traces *BSP*$(Cl(Tr))$ holds. The set of (wrt. $\preceq$) minimal closure operations which ensure *BSP*, the set of *induced closure operations* for *BSP*, is denoted by CL_{BSP}.

The following simple theorem states which BSPs have a unique corresponding closure operation. All other BSPs have, in general, more than one induced closure operation.

Theorem 11. *For BSP* $\in \{DE, RE, SRI, SDI, IE, IHAE, IAE, SII, SIHAI, SIAI\}$, CL_{BSP} *is a singleton. For these cases, we denote the unique closure operation by* Cl_{BSP}.

Given $Tr = \{hi.l_1.ho.l_2, hi.l_1.ho, hi.l_1, hi, \langle\rangle\}$, for example, $Cl_{SDI}(Tr) = Tr \cup \{l_1.ho.l_2, l_1.ho, l_1\}$ holds. This set also satisfies *BSDI*, however, *BSDI* does not have a unique closure operation. $Tr \cup \{ho.l_1.ho.l_2, ho.l_1.ho, ho.l_1, ho\}$, for example, would be another set which is closed wrt. *BSDI*. Note that the latter set is not a closed wrt. *SDI*.

5. Assembling Security Predicates

In this section, we illustrate how security predicates can be defined and demonstrate the expressiveness of our framework by describing several previously proposed definitions of security as security predicates, i.e. conjunctions of BSPs, from the preceding section. The defined security predicates are ordered with respect to their logical strength and will be used as examples in the remainder of this article. The modular construction within our framework helps us in achieving these results easily.

Generalized non-inference GNF [12] demands that a low-level user cannot infer that high-level inputs have occurred. It has been motivated by a limitation of *non-inference NF* [16] which prevents not only information flow from the high- to the low-level but also certain kinds of information flow in the other direction. We will discuss information flow from the low- to the high-level in greater detail in Section 6. *Generalized non-interference GNI* [10] demands that any interleaving of the high-level input of one trace with the low-level behaviour of another trace can be made a possible trace by adapting the outputs. We use the definition of generalized non-interference from [12] in which a function *interleave* is used to construct the set of all possible interleavings for two traces. All of these definitions allow some information flow from the high- to the low-level. McLeans *separability SEP* [12] prevents any such information flow. A similar notion of security has been introduced by Foley [5]. However, separability is stronger than necessary because the *perfect security property PSP* [20] also prevents any information flow from the high- to the low-level. Furthermore, there is no weaker definition which shares this property [20]. *PSP* resulted from the observation that high-level admissibility restricts information flow from the low- to the high-level and that therefore general admissibility should be used instead.

Definition 12.

$$
\begin{aligned}
GNF(Tr) &\equiv \forall t \in Tr.\exists t' \in LLES(Tr,t).t'|_{HI} = \langle\rangle \\
NF(Tr) &\equiv \forall t \in Tr.\exists t' \in LLES(Tr,t).t'|_{H} = \langle\rangle \\
GNI(Tr) &\equiv \forall t_l, t_h \in Tr.\forall t \in interleave(t_h|_{HI}, t_l|_{L}). \\
&\quad\ \exists t' \in Tr.t = t'|_{L\cup HI} \\
SEP(Tr) &\equiv \forall t_l, t_h \in Tr.\forall t \in interleave(t_h|_{H}, t_l|_{L}).t \in Tr \\
PSP(Tr) &\equiv \forall t \in Tr.\forall \alpha, \beta \in E^*.\forall e \in E.t|_{L} \in LLES(Tr,t) \\
&\quad \wedge[(e \in H \wedge \beta.\alpha \in LLES(Tr,t) \wedge \alpha|_{H} = \langle\rangle \wedge \\
&\quad\quad \beta.e \in Tr) \Rightarrow \beta.e.\alpha \in LLES(Tr,\tau)]
\end{aligned}
$$

All of these definitions of security can be expressed as security predicates in our framework.

Theorem 13. *The following equivalences hold:*

- $GNF(Tr) \Leftrightarrow RI(Tr)$
- $NF(Tr) \Leftrightarrow RE(Tr)$
- $GNI(Tr) \Leftrightarrow (RI(Tr) \wedge IHAI(Tr))$
- $SEP(Tr) \Leftrightarrow (RE(Tr) \wedge IHAE(Tr))$
- $PSP(Tr) \Leftrightarrow (RE(Tr) \wedge IAE(Tr))$

Proof. The first two equivalences and the last one follow directly from Figures 1, 3 and Definition 12. Recall, that *Tr* is closed under prefixes, therefore, e.g. $\langle\rangle \in Tr$ if $Tr \neq \emptyset$.

- For *GNI* $\Rightarrow$ *RI* choose $t_l = \tau$, $t_h = \langle\rangle$, $t = \tau|_L$, and $\tau' = t'$ in Figure 1 and Definition 12. For *GNI* $\Rightarrow$ *IHAI* choose $t_l = \tau = \beta.\alpha$, $t_h = \gamma.e$, $t = (\beta.e.\alpha)|_{(L \cup HI)}$, and $\tau' = t'$ in Figure 3 and Definition 12.
- (*RI* $\wedge$ *IHAI*) $\Rightarrow$ *GNI* is proved by induction on the length of $t_h|_{HI}$. The base case follows from *RI*. In the step case we assume the implication for all t_h^* with less than n and prove it for t_h with n high-level inputs. We choose $t_{h1}, t_{h2}, t_1, t_2 \in E^*$ and $hi \in HI$ such that $t_h = t_{h1}.hi.t_{h2}$, $t = t_1.hi.t_2$, and $t_{h2}|_{HI} = t_2|_{HI} = \langle\rangle$ hold. Since $t_{h1} \in Tr$, the induction assumption implies that there are t_1', t_2' with $t_1'.t_2' \in Tr$, $t_1'|_{L \cup HI} = t_1|_{L \cup HI}$, and $t_2'|_{L \cup HI} = t_2|_{L \cup HI}$. We choose (in *IHAI*) $\tau = t_1'.t_2'$, $\beta = t_1'$, $\alpha = t_2'$, $e = hi$, $\gamma = t_{h1}$, and construct $t' = \tau' = \beta'.e.\alpha'$.
- For *SEP* $\Rightarrow$ *RE* choose $t_l = \tau$, $t_h = \langle\rangle$, $t = \tau|_L$, and $\tau' = t$ in Figure 1 and Definition 12. For *SEP* $\Rightarrow$ *IHAE* choose $t_l = \tau = \beta.\alpha$, $t_h = \gamma.e$, $t = \beta.e.\alpha$, and $\tau' = t$ in Figure 3 and Definition 12.
- (*RE* $\wedge$ *IHAE*) $\Rightarrow$ *SEP* is proved by induction on the length of $t_h|_H$. The base case follows directly from *RE*. In the step case we assume the implication for all t_h^* with less than n and prove it for t_h with n high-level events. We choose $t_{h1}, t_{h2}, t_1, t_2 \in E^*$ and $h \in H$ such that $t_h = t_{h1}.h.t_{h2}$, $t = t_1.h.t_2$, and $t_{h2}|_H = t_2|_H = \langle\rangle$ hold. Since $t_{h1} \in Tr$, the induction assumption implies $t_1.t_2 \in Tr$. We choose (in *IHAE*) $\tau = t_1.t_2$, $\beta = t_1$, $\alpha = t_2$, $e = h$, $\gamma = t_{h1}$, and construct $t' = \tau' = \beta.e.\alpha$. □

Theorem 13 demonstrates that all these previously proposed definitions of possibilistic security can be expressed in our framework and the distinction of two dimensions for BSPs provides us with an intuitive understanding. E.g. *GNI* demands that the low-level user cannot infer that any high-level inputs have occurred (*RI*) or that a hl-admissible high-level input has not occurred (*IHAI*).

GNF, *NF*, *GNI*, and *SEP* can also be expressed by McLeans selective interleaving functions (*sifs*) [12, 14]. However, *sifs* are not expressive enough to define inductive definitions like *PSP*. *interleave* can only capture high-level admissibility. This was the main motivation for the definition of a new framework by Zakinthinos and Lee [20]. However, no correspondence exists between definitions of security and closure conditions in their framework. One benefit of our framework over [20] is that this correspondence exists as it does in the framework from [12, 14]. Each security predicate is a closure condition which induces a set of closure operations. CL_{SP} and Cl_{SP} are defined accordingly. The correspondence to closure conditions will be the basis for our investigations in Section 6 and 7. Another improvement is the modular structure which allows us to achieve results like the following theorem easily from properties of the building blocks. The additional security predicate *RI* $\wedge$ *IAE* in the theorem will be used in Section 6.

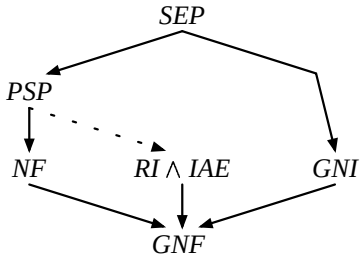

Figure 5. Security predicates

Theorem 14. *GNF, NF, GNI, GNI, SEP, PSP, and RI∧IAE are ordered by implication as depicted in Figure 5.*[2]

Proof. Due to the modular construction within our framework, most of the implications can be inferred from earlier results. *SEP* $\Rightarrow$ *PSP*, *PSP* $\Rightarrow$ *NF*, *NF* $\Rightarrow$ *GNF*, *PSP* $\Rightarrow$ (*RI* $\wedge$ *IAE*), (*RI* $\wedge$ *IAE*) $\Rightarrow$ *GNF*, and *GNI* $\Rightarrow$ *GNF* follow directly from Theorems 4, 7, and 13. We only need to prove *SEP* $\Rightarrow$ *IHAI*. With this implication, *SEP* $\Rightarrow$ *GNI* follows from Theorems 4 and 13. We assume *Tr* for which *RE*(*Tr*) and *IHAE*(*Tr*) hold, choose arbitrary $\tau \in Tr$, $\alpha, \beta \in E^*$, and $e \in HI$ with $\tau = \beta.\alpha$, $\alpha|_{HI} = \langle\rangle$, and $HAdm_{HI}(Tr, \beta, e)$. Because *RE* holds for *Tr* we conclude that $\tau|_L \in Tr$. $HAdm_{HI}$ ensures that there is a trace τ^* such that $\tau^*|_{HI} = (\beta.e)|_{HI}$. Choose $\alpha^*, \beta^* \in E^*$ such that $\tau^* = \beta^*.e.\alpha^*$ and $\beta^*|_{HI} = \beta|_{HI}$. *IHAE* allows us to insert all high-level events from $\beta^*.e$ into $\tau|_L$ ($HAdm_H$ is fulfilled) until we receive $\tau' \in LLES(Tr, \tau)$ for which $\alpha', \beta' \in E^*$ exist with $\tau' = \beta'.e.\alpha'$, $\beta'|_{L \cup HI} = \beta|_{L \cup HI}$, and $\alpha'|_{L \cup HI} = \alpha|_{L \cup HI}$. Since τ was arbitrary, *IHAI* holds for *Tr*. □

6. Compatibility with other Properties

The compatibility of security with other properties is a critical issue. The specification of a secure system consists not only of a security property but also of other system properties. Usually, such system properties are modeled as sets of traces. Thus, a secure system can be specified as a pair

[2] Note that there is a link from *PSP* to *NF* in Figure 5 which was forgotten in [20]. The link from *PSP* to *GNI* in that paper results from the assumption of input totality which we do not make.

(P, SP) where P is a set of traces and SP is a security predicate. For example, P could specify a bookkeeping system and SP the intended security property. If P is not closed for SP then P alone is an insecure specification. Simply constructing the closure $Cl_{SP}(P)$ of P by adding traces leads to a secure specification which allows behaviours which a bookkeeping system is not supposed to have. In order to ensure the bookkeeping functionality we, therefore, have to construct a subset $P' \subseteq P$ which is closed for SP. This leads to a specification of a secure bookkeeping system. Unfortunately, P' may allow no traces, i.e. be an inconsistent specification, require the system to stop after some steps, or just restrict the possible behaviours in a less severe but still undesirable way. Clearly, a specifier should be aware of any such restrictions. Therefore it is of interest with which kinds of properties the various security predicates are compatible or incompatible.

The correspondence between security predicates and closure conditions gives rise to a formal definition of compatibility which we use as the basis for our investigations. We illustrate the approach at several examples which require information flow from the low- to the high-level for the security predicates from the previous section. The results are also interesting in their own right. It turns out that a complete prevention of information flow from the high- to the low-level implies that certain kinds of information flow in the other direction are also restricted. If such information flow is required then complete security must be sacrificed. We propose a new security predicate *PGSP* which can be used to gradually allow for information flow from low to high while minimizing the resulting loss of security.

6.1. Closures and Compatibility

Definition 15. A *property* P is a set of traces over E. A closure operation Cl is *compatible* with P if $Cl(P) = P$. A security predicate *SP* is *compatible* with P if one of its induced closure operations is. Otherwise, it is *incompatible*.

The compatibility of *SP* with P ensures that no stronger property than P is enforced by SP which might restrict the possible behaviours in an undesirable way. The following two lemmas are very useful in proving the compatibility or incompatibility of a security predicate with a given property in the subsequent subsections.

Lemma 16. *Let SP_1 and SP_2 be security predicates with $SP_1 \Rightarrow SP_2$. If P is compatible with SP_1 then it is compatible with SP_2. If P is incompatible with SP_2 then it is incompatible with SP_1.*

Proof. We prove that if there is a closure operation Cl_1 induced by SP_1 then there is a closure operation Cl_2 induced by SP_2 such that $Cl_2 \preceq Cl_1$ holds. The lemma follows from this and **Cl1** from Definition 9. Cl_1 is a closure operation for SP_2 because $SP_1(Cl_1(Tr))$ implies $SP_2(Cl_1(Tr))$. If Cl_1 is minimal wrt. $\preceq$ then choose $Cl_2 = Cl_1$, otherwise, choose some $Cl_2 \preceq Cl_1$ which is minimal. □

Lemma 17. *Let $Cl_1 \in CL_{SP_1}$, $Cl_2 \in CL_{SP_2}$ be closure operations for security predicates SP_1, SP_2 and let $\circ$ denote function composition. If $Cl_2 \circ (Cl_1 \circ Cl_2) = Cl_1 \circ Cl_2$ then $Cl_1 \circ Cl_2$ is a closure operation for $SP_1 \wedge SP_2$. If CL_{SP_1} and CL_{SP_2} are singleton sets then $CL_{SP_1 \wedge SP_2}$ is a singleton set with $Cl_{SP_1} \circ Cl_{SP_2}$ as the only minimal closure operation.*

Proof. We first show that $Cl_1 \circ Cl_2$ is a closure operation. **Cl1** and **Cl2** in Definition 9 follow from **Cl1** and **Cl2** for Cl_1 and Cl_2. The following equations prove **Cl3**.

$$
\begin{array}{ll}
(Cl_1 \circ Cl_2) \circ (Cl_1 \circ Cl_2) & \\
= Cl_1 \circ (Cl_2 \circ (Cl_1 \circ Cl_2)) & \text{(associativity)} \\
= Cl_1 \circ (Cl_1 \circ Cl_2) & \text{(by assumption)} \\
= (Cl_1 \circ Cl_1) \circ Cl_2 & \text{(associativity)} \\
= Cl_1 \circ Cl_2 & \textbf{(Cl3)}
\end{array}
$$

$SP_1(Cl_1(Tr_1))$ and $SP_2(Cl_2(Tr_2))$ hold for any Tr_1 and Tr_2. To show that $Cl_1 \circ Cl_2$ is a closure operation for $SP_1 \wedge SP_2$ we choose $Tr_1 = Cl_2(Tr)$ and $Tr_2 = (Cl_1 \circ Cl_2)(Tr)$ for arbitrary Tr.

To show that $Cl_{SP_1} \circ Cl_{SP_2}$ is the unique minimal closure operation for $SP_1 \wedge SP_2$ (if CL_{SP_1} and CL_{SP_2} are singleton sets), we assume that Cl is a minimal closure operation for $SP_1 \wedge SP_2$. Thus, Cl is a closure operation for SP_1 as well as for SP_2. Because Cl_{SP_1} and Cl_{SP_2} are both minimal and unique, we receive $Cl_{SP_1} \preceq Cl$ and $Cl_{SP_2} \preceq Cl$. This implies that $(Cl_{SP_1} \circ Cl_{SP_2}) \preceq (Cl \circ Cl) = Cl$. □

6.2. Information Flow from Low- to High

The two-level security policy Pol_{HL} requires that there is no information flow from the high- to the low-level, i.e. $H \not\rightsquigarrow L$. It does not restrict information flow in the other direction $L \rightsquigarrow H$. In fact, such information flow is often required, however, many well-known definitions of possibilistic security restrict or prevent it completely. We demonstrate, using concrete examples, that it depends on the specific kind of information flow if a security predicate is compatible or not.

In particular, we investigate the auditing of low-level events on the high-level. Here, each low-level event l is recorded by a corresponding high-level output $ho(l)$ from a set of recording events $HO_{rec} \subseteq HO$. We distinguish three variations of such properties, depending on when the recording event must occur. In *IRec* (immediate recording), the recording event $ho(l)$ must occur immediately after the low-level event l has occurred. In *SRec* (stepwise recording) $ho(l)$ must occur before the next low-level event l' occurs.

Finally, *Rec* (general recording) demands only that some part of the low-level history $t|_L$ is recorded. In the formal definition, this is expressed using a relation $\sqsubseteq \subseteq E^* \times E^*$. $\gamma \sqsubseteq \gamma'$ holds if γ is an initial substring of γ'. Clearly, *IRec* is more restrictive than *SRec* and *SRec* is more restrictive than *Rec*. These properties are formalized as follows:

$$\begin{aligned} IRec(t) &\equiv \forall \alpha, \beta \in E^*. \forall l \in L. \forall e \in E. \\ &\quad t = \beta.l.e.\alpha \Rightarrow e = ho(l) \\ SRec(t) &\equiv \forall \alpha, \beta \in E^*. \gamma \in H^*. l, l' \in L. \\ &\quad [t = \alpha.l.\gamma \Rightarrow \gamma|_{HO_{Rec}} \sqsubseteq ho(l)] \wedge \\ &\quad [t = \alpha.l.\gamma.l'.\beta \Rightarrow \gamma|_{HO_{Rec}} = ho(l)] \\ Rec(t) &\equiv t|_{HO_{rec}} \sqsubseteq ho(t|_L) \end{aligned}$$

For auditing, a low-level event enforces a high-level event. The prevention of a high-level event by the occurrence of a low-level event is an alternative type of information flow. A system where a guard must pass a checkpoint within a specific time period is an example which requires this type of information flow. If the guard passes the checkpoint in time, then no alarm should be invoked, otherwise, the alarm may be invoked. We distinguish two variants. In *IPI* (immediate prevention of input) the next event after l must not be a high-level input from the set $HI(l)$ of prevented inputs and in *IPO* (immediate prevention of output) the next event after l must not be a high-level output from the set $HO(l)$. These properties are formalized as follows:

$$\begin{aligned} IPI(t) &\equiv \forall \alpha, \beta \in E^*. \forall l \in L. \forall e \in E. \\ &\quad t = \beta.l.e.\alpha \Rightarrow e \notin HI(l) \\ IPO(t) &\equiv \forall \alpha, \beta \in E^*. \forall l \in L. \forall e \in E. \\ &\quad t = \beta.l.e.\alpha \Rightarrow e \notin HO(l) \end{aligned}$$

Lemma 18.

1. *SDI, IAE, and SIAI are compatible with IRec. SDI, IAE, and SII are compatible with SRec. SDI, DE, IAE, and SII are compatible with Rec. SDI, DE, IAE, and SIAI are compatible with IPI. DE, IAE, and SII are compatible with IPO.*
2. *RE, IHAE, and IHAI are incompatible with IRec. RE and IHAE are incompatible with SRec. IHAE is incompatible with Rec. IHAE and IHAI are incompatible with IPI. SRI and IHAE are incompatible with IPO.*

A proof of Lemma 18 is contained in the appendix. The next theorem follows directly from Lemma 16 and 18. The modular structure of our framework has simplified this proof considerably. The only cases that we had to prove (in Lemma 18) are the ones which are underlined in Figure 6, i.e. only 27 out of 95 cases had to be investigated.

Theorem 19. *The basic security predicates are compatible/incompatible with IRec, SRec, Rec, IPI, and IPO as depicted in Figure 6.*

	compatible	incompatible
IRec	*SDI, SRI, RI, BSDI, DI, IAE, SIAI, BSIAI, IAI*	*RE, DE, IHAE, IE, IHAI, SII, BSII, II, SIHAI, BSIHAI*
SRec	*SDI, SRI, RI, BSDI, DI, IAE, SII, BSII, II, SIHAI, BSIHAI, IHAI, SIAI, BSIAI, IAI*	*RE, DE, IHAE, IE*
Rec	*SDI, SRI, RI, BSDI, DI, DE, RE, IAE, SII, BSII, II, SIHAI, BSIHAI, IHAI, SIAI, BSIAI, IAI*	*IHAE, IE*
IPI	*SDI, SRI, RI, BSDI, DI, DE, RE, IAE, SIAI, BSIAI, IAI*	*IHAE, IE, IHAI, SII, BSII, II, SIHAI, BSIHAI*
IPO	*DE, BSDI, DI, RI, RE, IAE, SII, BSII, II, SIHAI, BSIHAI, IHAI, SIAI, BSIAI, IAI*	*SRI, SDI, IHAE, IE*

Figure 6. Compatibility of BSPs

The following theorem follows from Theorem 13, Lemma 16, and Theorem 19.

Theorem 20.

- *NF and PSP are incompatible with IRec and SRec.*
- *GNI is incompatible with IRec and IPI.*
- *SEP is incompatible with IRec, SRec, Rec, IPI, and IPO.*

Theorem 20 might suggest that all established security properties, except for *GNF*, are incompatible with information flow $L \rightsquigarrow H$. *SEP* is not even compatible with a single one of the investigated kinds of information flow. However, depending on the application, some of these incompatibilities might be acceptable. Our investigation identifies that the problem for *PSP* is the use of *RE*. To use any of *BSDI*, *DI*, or *RI* would solve the problem. This is the motivation for the definition of *PGSP* in Subsection 6.3.

Lemma 21. $Cl_{BSP_1} \circ (Cl_{BSP_2} \circ Cl_{BSP_1}) = Cl_{BSP_2} \circ Cl_{BSP_1}$ *holds for* $BSP_1 \in \{SDI, DE\}$ *and* $BSP_2 \in \{SII, SIAI, IAE\}$.

The proof of Lemma 21 is contained in the appendix.

Theorem 22.

1. *GNF and RI* $\wedge$ *IAE are compatible with IRec, SRec, Rec, IPI, and IPO.*
2. *NF and PSP are compatible with Rec, IPI, and IPO.*
3. *GNI is compatible with Rec, SRec, and IPO.*

Proof. Recall Theorem 13 which demonstrates how the security predicates can be assembled from BSPs. The compatibilities of *GNF* and *NF* follow directly from Theorem 19.

The compatibilities of *RI* ∧ *IAE* follow from Lemma 16 and the compatibility of *SDI* ∧ *IAE* with *IRec*, *SRec*, *Rec*, and *IPI* and of *DE* ∧ *IAE* with *IPO*. The compatibilities of *PSP* follow from Lemma 16 and the compatibility of *DE* ∧ *IAE* with *Rec*, *IPI*, and *IPO*. The compatibilities of *GNI* follow from Lemma 16, the compatibility of *DE* ∧ *SII* with *Rec* and *IPO*, and the compatibility of *SDI* ∧ *SII* with *SRec*. Compatibilities of the conjunctions used in this proof follow from Lemma 17, 21, and Theorem 11, 19. □

We have demonstrated at several examples how to prove the compatibility or incompatibility of a security predicate with a property and how to exploit the modularity of our framework during this process. The results are also interesting in their own right. We have shown that the compatibility of a security predicate with information flow from the low- to the high-level depends on the particular kind of information flow because the results for well-known definitions differ for the various examples (Theorem 20 and 22). *PSP* is incompatible with certain kinds of information flow from L to H. According to Theorem 2 in [20] there is no security property which is weaker than *PSP* and ensures complete security, i.e. prevents all information flow from H to L. Thus, one cannot have both, complete security and information flow like *IRec* or *SRec*.

6.3. The Pretty Good Security Predicate

We have demonstrated in the previous subsection that certain kinds of information flow become impossible when complete security is required. Thus, for information flow like in *IRec* or *SRec* one must sacrifice complete security. Clearly, it is desirable not to give up more security than necessary. This is the motivation for the definition of a new security property, the pretty good security predicate *PGSP*.

Theorem 22 (1) demonstrates that *RI* ∧ *IAE* shares the compatibilities of *GNF*, i.e. it is compatible with all the properties which we have considered. On the one hand, it is strictly stronger than *GNF* and, thus, more secure. On the other hand, it provides only slightly less security than *PSP* does (which ensures complete security). Therfore, *RI* ∧ *IAE* is an attractive candidate for a security predicate. We define the *pretty good security predicate* by

$$PGSP \equiv RI \wedge IAE\ .$$

Since *PGSP* is weaker than *PSP*, it must allow for some (undesired) flow of information from the high- to the low-level. It is of interest to identify to which extent and by which means high-level behaviour can be deduced for a system which satisfies *PGSP*. By looking at the basic building blocks we see that the use of *RI* instead of *RE* can be the only reason for such information flow. Thus, *PGSP* allows the low-level user to infer that a high-level output or internal event has occurred.

This provokes the question if it is possible to strengthen *PGSP* while preserving the compatibilities. In fact, it is possible. Let $H' \subseteq H \setminus HI$ contain all high level events which are concerned with the processing of low-level information. Let $R(H \setminus H')$ be the BSP which requires the removal of all high-level events which are not in H'. Then $PGSP_{H'} \equiv R(H \setminus H') \wedge IAE$ has the same compatibilities as *PGSP* does. Depending on the choice of H' one receives a collection of predicates which is indicated by the dotted arrow in Figure 5. In order to receive a security predicate, as strong as possible, one must choose H' as small as possible. However, it is essential that H' really contains all high level events which are concerned with the processing of low-level information because, otherwise, the compatibilities may be lost. Note that $PGSP = PGSP_{H \setminus HI}$ and $PSP = PGSP_{\emptyset}$.

A look at the BSPs from which $PGSP_{H'}$ is composed yields an intuitive understanding: By observing a system which is $PGSP_{H'}$-secure, a low-level user cannot infer that an admissible high-level event has *not* occurred or that a high-level event in $H \setminus H'$ has occurred. However, he might be able to infer that a high-level event from H' has occurred, i.e. events which are concerned with the processing of low-level events must not be confidential.

7. Stepwise Development of Secure Systems

A stepwise development process for secure systems is desirable. In stepwise development, one starts with an abstract specification and refines it in several steps to a concrete specification of a system which then can be implemented. However, unlike other system properties, security properties need to be expressed as sets of trace sets and cannot be expressed simply by sets of traces, thus, they are outside the Alpern-Schneider framework of safety and liveness properties [12]. Moreover, Jacob [9] showed that, in general, a security ordering is neither monotonic nor antimonotonic with respect to the subset relation. Since the subset relation is the basis for refinement, this gives rise to the *refinement paradox* which says that the refinement of a secure system need not be secure. Thus, in general, security is not preserved under refinement.

The refinement paradox is a major obstacle for a stepwise development of secure systems. However, our general correspondence between security predicates and closure conditions can be used as the basis for such a stepwise development. Starting with an abstract specification $ES_a = (E, I, O, Tr_a)$, which has been proved to be secure, one constructs more concrete specifications. In each step one constructs a specification $ES_c = (E, I, O, Tr_c)$ which must be proved to be closed wrt. the corresponding closure condition or, alternatively, the closure of ES_c can be constructed using an appropriate closure operation. While this idea might sound simple, in theory, it can be difficult to ap-

ply in practice. Therefore, it is important to identify specialized refinement operators which preserve security. In the sequel we demonstrate that the intersection of specifications is such a refinement operator which preserves certain security predicates. The correspondence to closure conditions is very helpful in achieving this result.

The following theorem shows that the intersection of two specifications which are closed under a closure operation Cl is again closed under Cl.

Theorem 23. *Given two properties P_1 and P_2 with which a closure operation Cl is compatible then Cl is also compatible with $P_1 \cap P_2$.*

Proof. According to Theorem 1.1 in [3] a closure operation induces a closure system and closure systems are closed under intersection. □

Corollary 24. *Given two properties P_1 and P_2 which are secure with respect to a security predicate SP which induces a unique closure operation Cl then $P_1 \cap P_2$ is secure with respect to SP.*

When constructing a refinement ES_c of a specification ES_a, which is secure with respect to *SP*, then one must again prove that ES_c is secure. Corollary 24 states that we do not need to re-prove security if we use intersection because *intersection preserves security*.

If *SP* does not have a unique closure operation (cf. Theorem 11) then Corollary 24 is not applicable. In this case one should select one particular closure operation Cl in CL_{SP} and use Theorem 23. This requires that the security of specifications which are composed has been proved wrt. the same closure operation.

While the stepwise development of a secure system from its specification, i.e. the top-down approach, has been somewhat handicapped by the refinement paradox, the bottom-up approach, i.e. the modular construction of a secure system from components has received considerable attention, e.g. [10, 12, 14, 17, 19]. If the composition of secure components yields again a system which is secure then the security of composed systems can be proved in a modular fashion. This reduces the complexity when reasoning about security. Another motivation is that it allows for the use of off-the-shelf components which are certified to fulfill a certain security property but for which the complete system documentation is not disclosed. Composed systems are a natural area of application for possibilistic security because they are often non-deterministic, like distributed systems in general. However, an investigation of system composition within our framework is outside the scope of this article and we point to the existing work mentioned above.

8. Discussion

We discuss the relation to previously proposed frameworks for possibilistic security by McLean [12, 14] and Zakinthinos/Lee [20] as well as to a comparison of possibilistic security properties by Focardi and Gorrieri [4]. Finally, we summarize criteria for selecting a security predicate.

McLean uses traces that are sequences of states (instead of events) as basis for his framework of selective interleaving functions. A selective interleaving function (*sif*) takes two traces as arguments and returns a trace. The type of a *sif* decides from which trace the values of a state object are taken for the resulting trace or if they may be chosen arbitrarily. In McLeans framework each definition corresponds to a closure condition, i.e. the set of traces must be closed under a *sif* of a specific type. The expressiveness of his framework is limited since *sifs* can only express high-level admissibility and cannot capture inductive definitions (cf. Section 5), which are required in *PGSP* as well as in *PSP*.

The framework of Zakinthinos and Lee [20] uses a schema for the representation of security properties which is based on the notion of a low-level equivalence set (like ours). Their framework is more expressive than the one by McLean and can capture inductive definitions as well as admissibility. However, the correspondence between security properties and closure conditions is lost. Their schema provides less structure than ours for the representation of security properties and does not incorporate a satisfiability condition (cf. Definition 2).

In our framework, we re-establish the correspondence between security properties and closure conditions by the satisfiability condition in the schema for BSPs (cf. Definition 2). The quantification of τ, α, β, e, and τ' allowed us to express such a satisfiability condition. The distinction of a restriction R and a closure requirement Q provides more structure for the definition of a BSP and for proving satisfiability conditions. Apparently, this additional structure could restrict the expressiveness of our framework in comparison to the one by Zakinthinos and Lee. The reader might be curious since we have not formalized McCulloughs original definition of generalized noninterference or non-deducibility output security [8] which can be expressed in the framework by Zakinthinos and Lee. However, both of these definitions of security could also be expressed as security predicates in our framework, although this requires the definition of further BSPs.[3] In general, the schema can capture closure conditions where the requirement of τ' is caused by a finite set of traces rather than only by a single trace τ. First, one should recall that e.g. in *IHAI* or

[3] Non-deducibility output security can be expressed by choosing $\tau =$ 't', $\alpha =$ 'τ', and $\tau' =$'s' in Definition 2 where the symbols in apostrophes are taken from [20]. The concept underlying McCulloughs definition of generalized non-interference is similar to our backwards strict BSPs.

IAI the existence of τ' depends on two traces τ and $\gamma.e$ or $\beta.e$ and all definitions of security which were expressed in the selective interleaving framework could be expressed in our framework as well. If definitions of security are defined which cannot be expressed easily by our schema, one might be tempted to use α or β for artificial encodings. Since this is not intuitive, we propose that under such circumstances one should move to a more general schema in which τ is replaced by a finite subset T of *Tr* which results in:

$$\forall T \in \mathcal{P}_{fin}(Tr), (\alpha_\tau, \beta_\tau)_{\tau \in T} \in (E^* \times E^*)^T, (e_\tau)_\tau \in E^T.\ [R(\ldots) \Rightarrow \exists \tau'.Q(\ldots)]$$

Focardi and Gorrieri [4] investigate various possibilistic security properties in a process algebra based on CCS [15]. Therefore, internal events are not distinguished but rather identified with a single hidden event τ. In particular, nondeterministic non-interference (which is similar to *GNF* discussed in this paper), strong non-deterministic noninterference (similar to *NF*), two-level non-deducibility on inputs (similar to *GNI*), lts-restrictiveness, lts-correctability, and non-deducibility on compositions are compared. Interestingly, the comparison is not limited to trace equivalence but also covers other notions of behavioural equivalence, in particular weak bisimulation. However, unlike in [12, 14, 20] or this article, no general schema for the definition of possibilistic security (like McLeans selective interleaving functions, the security properties from [20], or our security predicates) is provided which we consider to be a necessary component of a general framework for security.

We now return to the issue of selecting an appropriate definition of security which has been mentioned at various places in this article. We have argued that the choice of an appropriate definition depends on the application and that there is no definition which can be recommended as the optimal one for all purposes. As usual in the development of secure systems one has to analyze which information flows are critical and what threads exist. This might result e.g. in the decision that only high-level inputs or outputs must be hidden from low-level users. The compatibility of security with other system properties is critical because if the two are incompatible, a stronger system property must be implemented. If one wants to use off-the-shelf components which are certified to satisfy a given security property then the chosen definition of security and the one ensured by the component must fit together. The composability of specifications for a stepwise development process and the composability of components is important for modular system construction. Another issue which – to our knowledge – so far has not been addressed in the literature (including this article) is, how difficult it is to actually prove that a system fulfills a possibilistic security property. This certainly is an important area for future work.

9. Conclusion

We have proposed a new framework for investigating and comparing definitions of possibilistic security, which can be assembled within our framework in a modular way and provided a variety of BSPs as building blocks for such definitions. New building blocks can be added using a general schema for the definition of BSPs. All definitions expressed in earlier frameworks [12, 14, 20] could also be expressed in our framework. Our framework is more expressive than the one by McLean [12, 14] and there is a general correspondence between definitions of security and closure conditions which is not present in the framework by Zakinthinos and Lee [20]. We have investigated the compatibility of the various definitions of security with system properties apart from security at several examples which require information flow from the low- to the high-level. In order to overcome limitations of the perfect security property *PSP*, we have derived the pretty good security predicate *PGSP*. We also indicated how a stepwise and modular development of secure systems is possible in our framework and demonstrated the benefits of using intersection. Finally, we have provided criteria for the selection of an appropriate security predicate.

In future work we intend to investigate how systems can be proved to fulfill a possibilistic security property. Another area of interest are intransitive security policies for non-deterministic systems.

Acknowledgments

This work benefited from discussions with Fred B. Schneider and Christoph Kreitz. Thanks to Christoph and Dieter Hutter for many valuable comments on the presentation and to Cornell University for providing an inspiring working environment.

References

[1] D. E. Bell and L. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. Technical Report MTR-2997, MITRE, March 1976.

[2] K. Biba. Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, MITRE, 1977.

[3] P. M. Cohn. *Universal Algebra*. Harper & Row, 1965.

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

[5] S. N. Foley. A Universal Theory of Information Flow. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 116–122, Oakland, CA, 1987.

[6] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, CA, 1982.

[7] J. A. Goguen and J. Meseguer. Inference Control and Unwinding. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 75–86, Oakland, CA, 1984.
[8] J. Guttman and M. Nadal. "What Needs Securing?". In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 34–57, 1988.
[9] J. Jacob. On the Derivation of Secure Components. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 242–247, Oakland, CA, 1989.
[10] D. McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 161–166, Oakland, CA, 1987.
[11] J. McLean. Security Models and Information Flow. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 180–187, Oakland, CA, 1990.
[12] J. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 79–93, Oakland, CA, 1994.
[13] J. McLean. Security Models. *Encyclopedia of Software Engineering*, 1994.
[14] J. McLean. A General Theory of Composition for a Class of "Possibilistic" Security Properties. *IEEE Transaction on Software Engineering*, 22(1):53–67, January 1996.
[15] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
[16] C. O'Halloran. A Calculus of Information Flow. In *Proceedings of the European Symposium on Research in Computer Security, ESORICS 90*, pages 147–159, Toulouse, France, 1990.
[17] A. Roscoe and L. Wulf. Composing and Decomposing Systems under Security Properties. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 9–15, Kenmare, Ireland, 1995.
[18] D. Sutherland. A Model of Information. In *9th National Computer Security Conference*, 1986.
[19] A. Zakinthinos and E. Lee. The Composability of Non-Interference. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 2–8, Kenmare, Ireland, 1995.
[20] A. Zakinthinos and E. Lee. A General Theory of Security Properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 94–102, Oakland, CA, 1997.

Appendix

Proof. (of Lemma 18) Recall that Theorem 11 ensures that all BSPs in the lemma induce a unique closure operation. We denote the respective closure operation by Cl and the property by *Tr* in each of the cases. Below, we show that $Cl(Tr) \subseteq Tr$. With $Tr \subseteq Cl(Tr)$ (**Cl1**) this implies $Tr = Cl(Tr)$, i.e. compatibility.

First, the proofs of the compatibilities. If $\tau \in Cl(Tr)$ for *SDI* then there are $\alpha, \beta \in E^*$ such that $\beta.\alpha \in Tr$ and $\tau = \beta.(\alpha|_{E \setminus HI})$ because Cl is minimal. *IRec* holds in $\beta.\alpha$ by assumption. The strict deletion of high-level inputs does not violate this. Thus, *SDI* is compatible with *IRec*. If $\beta.l.e.\alpha \in Cl(Tr)$ then $\beta.l.e \in Tr$ (*Adm*) for *IAE* and *SIAI* because Cl is minimal. Thus, they are compatible with *IRec*. The strict deletion of high-level inputs does not violate *SRec*. Thus, *SDI* is compatible with *SRec*. If $\tau \in Cl(Tr)$ then there is a $\tau' \in Tr$ such that τ results from τ' by the strict insertion of high-level inputs according to *SII*. *SRec* holds for τ' by assumption. The strict insertion does not violate *SRec*. Thus, *SII* is compatible with *SRec*. If $\tau \in Cl(Tr)$ then there is a $\tau' \in Tr$ such that τ results from τ' by the insertion of admissible high-level events according to *IAE*. *SRec* holds for τ' by assumption. The insertion does not violate *SRec* because Cl is minimal and, therefore, each of the inserted events must be admissible. Thus, *IAE* is compatible with *SRec*. If $\tau \in Cl(Tr)$ then there is a $\tau' \in Tr$ such that $\tau|_{HO_{Rec}} \sqsubseteq \tau'|_{HO_{Rec}}$ and τ results from τ' by the deletion of arbitrary high-level events for *DE*. $\tau|_L = \tau'|_L$ holds. Thus, *DE* is compatible with *Rec*. If $\tau \in Cl(Tr)$ then there is a $\tau' \in Tr$ such that $\tau|_{HO_{Rec} \cup L} = \tau'|_{HO_{Rec} \cup L}$ for *SDI* and *SII*. Thus, they are compatible with *Rec*. If $\tau \in Cl(Tr)$ then there is a $\tau' \in Tr$ such that $\tau|_{HO_{Rec}} = \tau'|_{HO_{Rec}}$ and $\tau|_L = \tau'|_L$ for *IAE*. This holds because Cl is minimal and because of the admissibility condition in *IAE*. Thus, *IAE* is compatible with *Rec*. The stepwise deletion of high-level events or high-level inputs does not violate *IPI*. Thus, *SDI* and *DE* are compatible with *IPI*. If $\beta.l.hi.\alpha \in Cl(Tr)$ then $\beta.l.hi \in Tr$ for *IAE* and *SIAI*. Thus, they are compatible with *IPI*. If $\tau \in Cl(Tr)$ for *DE* then there are $\alpha, \beta \in E^*$ such that $\beta.\alpha \in Tr$ and $\tau = \beta.(\alpha|_L)$ because Cl is minimal. *IPO* holds in $\beta.\alpha$ by assumption. The stepwise deletion of high-level events does not violate this. Thus, *DE* is compatible with *IPO*. If $\beta.l.ho.\alpha \in Cl(Tr)$ then $\beta.l.ho \in Tr$ for *IAE*. Thus, *IAE* is compatible with *IPO*. If $\tau \in Cl(Tr)$ for *SII* then there are $\alpha, \beta \in E^*$ such that $\beta.(\alpha|_{E \setminus HI}) \in Tr$ and $\tau = \beta.\alpha$ because Cl is minimal. *IPO* holds in $\beta.(\alpha|_{E \setminus HI})$ by assumption. Strict stepwise insertion of high-level inputs does not violate this. Thus, *SII* is compatible with *IPO*.

For the proofs of the incompatibilities, we give counterexamples. For the incompatibility of *RE* with *IRec* take the closure of $\{l.ho(l).l_2, l.ho(l), l, \langle\rangle\}$ which contains $l.l_2$. For the incompatibility of *IHAE* and *IHAI* with *IRec* take the closure of $\{hi, l.ho(l), l, \langle\rangle\}$ which contains $l.hi$. For the incompatibility of *RE* with *SRec* take the closure of the set $\{l_1.ho(l_1).l_2, l_1.ho(l_1), l, \langle\rangle\}$ which contains $l_1.l_2$. For the incompatibility of *IHAE* with *SRec* and *Rec* take the closure of $\{l_1.ho(l_1).l_2.ho(l_2), l_1.ho(l_1).l_2, l_1.ho(l_1), l_1, \langle\rangle\}$ which contains $ho(l_1).l_1.ho(l_2)$. For the incompatibility of *IHAE* and *IHAI* with *IPI* take the closure of the set $\{hi, l, \langle\rangle\}$ which contains $l.hi$ ($hi \in HI(l)$). For the incompatibility of *SRI* with *IPO* consider the closure of the set $\{l.hi.ho, l.hi, l, \langle\rangle\}$ which contains $l.ho$ ($ho \in HO(l)$). For the incompatibility of *IHAE* with *IPO* take the closure of the set $\{ho, l, \langle\rangle\}$ which contains $l.ho$ ($ho \in HO(l)$). □

Proof. (of Lemma 21) We prove each of the cases by contradiction. Assume that there is a set of traces *Tr* such that

the equation does not hold. Thus, there is a $\tau \in (Cl_{BSP_1} \circ (Cl_{BSP_2} \circ Cl_{BSP_1}))(Tr)$ with $\tau \notin (Cl_{BSP_2} \circ Cl_{BSP_1})(Tr)$.

For $BSP_1 = SDI$ and $BSP_2 \in \{SII, SIAI\}$: Since Cl_{SDI} is minimal we can choose τ such that $\beta, \alpha \in E^*$ and $e \in HI$ exist with $\tau = \beta.\alpha$, $\beta.e.\alpha \in (Cl_{BSP_2} \circ Cl_{SDI})(Tr)$, and $\alpha|_{HI} = \langle\rangle$. $\beta.e.\alpha \notin Cl_{SDI}(Tr)$ because, otherwise, we have a contradiction. $\beta.\alpha \in (Cl_{BSP_2} \circ Cl_{SDI})(Tr)$ since Cl_{BSP_2} is minimal. This contradicts our initial assumption.

For $BSP_1 = SDI$ and $BSP_2 = IAE$: Since Cl_{SDI} is minimal we can choose τ such that there are $\beta, \alpha \in E^*$ and $e \in HI$ with $\tau = \beta.\alpha$, $\tau' = \beta.e.\alpha \in (Cl_{IAE} \circ Cl_{SDI})(Tr)$, and $\alpha|_{HI} = \langle\rangle$. $\beta.e.\alpha \notin Cl_{SDI}(Tr)$ because, otherwise, we would have a contradiction. We distinguish two cases (1) there are $\beta_1, \beta_2 \in E^*$ with $\beta = \beta_1.\beta_2$ such that $\tau'' \in Cl_{SDI}(Tr)$ where $\tau'' = \beta_1.(\beta_2|_L).(\alpha|_L)$, (2) there are $\alpha_1, \alpha_2 \in E^*$ with $\alpha = \alpha_1.\alpha_2$ such that $\beta.e.\alpha_1.(\alpha_2|_L) \in Cl_{SDI}(Tr)$. In case (1) there is a sequence of traces $\sigma'' = \langle\tau_0'', \ldots \tau_n''\rangle$ in $(Cl_{IAE} \circ Cl_{SDI})(Tr)$ for which $\tau_0'' = \tau''$, $\tau_n'' = \tau'$, and τ_{i+1}'' results from τ_i'' by insertion of a single high-level event according to *IAE*. Since Cl_{IAE} is minimal, there is a sequence $s = \langle t_1, \ldots, t_n\rangle$ of traces in $Cl_{SDI}(Tr)$ such that t_i ensures the admissibility condition as required by *IAE* in the construction of τ_i'' from τ_{i-1}''. There is a $m \in \{1, \ldots, n\}$ such that $t_m = \beta.e$. For any $j \geq m$ there is a α_j such that $\tau_j'' = \beta.e.\alpha_j$ with $\alpha_j|_L = \alpha|_L$. We construct a sequence $\overline{s}$ of traces by $\overline{s} = \langle t_1, \ldots, t_{m-1}, \overline{t_{m+1}}, \ldots, \overline{t_n}\rangle$ where each of the $\overline{t_j}$ is constructed by deleting e in t_j according to *SDI*. Now, τ can be constructed from τ'' by the insertion of high-level events as in the construction of τ' from τ'' except for e which is not inserted. The admissibility of the events is ensured by $\overline{s}$. Thus, $\beta.\alpha \in (Cl_{IAE} \circ Cl_{SDI})(Tr)$, a contradiction. Case (2) is proved similarly by choosing $\tau'' = \beta.\alpha_1.(\alpha_2|_L)$.

For $BSP_1 = DE$ and $BSP_2 \in \{SII, SIAI\}$: Since Cl_{DE} is minimal we can choose τ such that there are $\beta, \alpha \in E^*$ and $e \in H$ with $\tau = \beta.\alpha$, $\beta.e.\alpha \in (Cl_{BSP_2} \circ Cl_{DE})(Tr)$, and $\alpha|_H = \langle\rangle$. $\beta.e.\alpha \notin Cl_{DE}(Tr)$ because, otherwise, we would have a contradiction. We distinguish two cases (1) $e \in HI$, (2) $e \in (H \setminus HI)$. In case (1), we infer $\beta.\alpha \in (Cl_{BSP_2} \circ Cl_{DE})(Tr)$ or $\beta.e.\alpha \in Cl_{DE}(Tr)$ from the minimality of Cl_{BSP_2}. In case (2), there exist $\beta_1, \beta_2 \in E^*$ such that $\beta = \beta_1.\beta_2$ and $\beta_1.(\beta_2|_{E \setminus HI}).e.\alpha \in Cl_{DE}(Tr)$ because Cl_{BSP_2} is minimal. Thus, $\beta_1.(\beta_2|_{E \setminus HI}).\alpha \in Cl_{DE}(Tr)$. According to *SII* and *SIAI* we can re-insert the HI-events into $\beta_1.(\beta_2|_{E \setminus HI}).\alpha$ and receive $\beta_1.\beta_2.\alpha \in (Cl_{BSP_2} \circ Cl_{DE})(Tr)$. For both cases we have a contradiction with our initial assumption.

For $BSP_1 = DE$ and $BSP_2 = IAE$: Since Cl_{DE} is minimal we can choose τ such that $\beta, \alpha \in E^*$ and $e \in H$ exist with $\tau = \beta.\alpha$, $\beta.e.\alpha \in (Cl_{IAE} \circ Cl_{DE})(Tr)$, and $\alpha|_H = \langle\rangle$. $\beta.e.\alpha \notin Cl_{DE}(Tr)$ because, otherwise, we would have a contradiction. $\beta.\alpha \in (Cl_{IAE} \circ Cl_{DE})(Tr)$ because Cl_{IAE} is minimal. This contradicts our initial assumption. □