# Preserving Information Flow Properties under Refinement

Heiko Mantel

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

## Abstract

*In a stepwise development process, it is essential that system properties that have been already investigated in some phase need not be re-investigated in later phases. In formal developments, this corresponds to the requirement that properties are preserved under refinement. While safety and liveness properties are indeed preserved under most standard forms of refinement, it is well known that this is, in general, not true for information flow properties, a large and useful class of security properties. In this article, we propose a collection of refinement operators as a solution to this problem. We prove that these operators preserve information flow as well as other system properties. Thus, information flow properties become compatible with stepwise development. Moreover, we show that our operators are an optimal solution.*

## 1 Introduction

In a stepwise development process, abstraction and decomposition are the main techniques that allow one to deal with the high complexity of large systems. In such a process, one usually starts with a very abstract specification of the desired system. This specification is then refined and decomposed until one arrives at a concrete specification that can directly be implemented. Naturally, one expects that a system which is developed formally in this way satisfies all properties that are satisfied by the abstract specification (plus possibly additional ones). While this holds for safety and liveness properties, it is not true for most information flow properties.

Information flow properties, however, provide a very elegant approach to specify security requirements. In this process, one first selects a set $\mathcal{D}$ of domains and then restricts the allowed flow of information between domains by a relation $\not\rightsquigarrow \subseteq \mathcal{D} \times \mathcal{D}$. For example, $H \not\rightsquigarrow L$ expresses that no information shall flow from domain $H$ to domain $L$. On the one hand side, this statement can be interpreted as a confidentiality requirement, i.e. that information in domain $H$ is hidden from domain $L$. On the other hand, it can be interpreted as an integrity requirement, i.e. that $L$ cannot be corrupted by $H$. Thus, information flow properties can be used to specify confidentiality as well as integrity requirements. Besides $\mathcal{D}$ and $\not\rightsquigarrow$, the only other ingredient that is required for information flow control is a formal definition of what information flow means. While for deterministic systems, non-interference [GM82] is widely accepted as the right definition of information flow, different definitions of information flow co-exist for non-deterministic systems.

Unfortunately, no satisfactory integration of information flow properties into a stepwise development process has been achieved so far. While many of these properties behave nicely under composition (cf. e.g. [McC87, JT88, WJ90, McL94]), the main problem is that they are, in general, not preserved under refinement. This problem has already been discussed in [Jac89] and some progress towards a solution has been made [GCS91, O'H92, RWW94]. However, to take information flow properties into account for design decisions during a stepwise development process is still infeasible because these properties would need to be re-investigated from scratch in every step. Since this would be too expensive, the only approach, that appears to be feasible today, is to consider information flow properties only at the very end of the development process when no further refinement of the specification is necessary. Clearly, this is a very poor approach because security is enforced after all design decisions have already been made. Enforcing security by preventing certain behaviours a posteriori may result in a so useless system that the complete development effort would be wasted.

In this article, we define conditions under that refinement preserves safety and liveness as well as information flow properties. Our main contribution is a collection of refinement operators that can be used not only for checking if a given refinement preserves information flow properties but also for constructing property preserving refinements. Clearly, the advantage of our approach is that information flow properties need only be proven once. They can be

In Proceedings of the 2001 IEEE Symposium on Security and Privacy, Oakland, CA, USA, pp.78–91, May 14-16, 2001. IEEE Computer Society.

taken into account during a stepwise development process without reproving them at every step. Thus, information flow properties become compatible with stepwise system development. Moreover, we demonstrate that our refinement operators provide an optimal solution to the problem.

This article is structured as follows: We introduce information flow control in Section 2 and refinement in Section 3. The original contributions of this article are presented in Section 4, 5, and 6. In Section 4, the problem of refining information flow properties is illustrated by a simple example before we present refinement operators for the perfect security property, an information flow property proposed in [ZL97]. We then prove that these operators preserve functional system properties as well as the perfect security property. The optimality of our refinement operators is discussed in Section 5. In Section 6, we generalize our results and propose refinement operators for other information flow properties. We conclude this article with a comparison to related work in Section 7 and a summary of our results together with a list of open tasks for future research in Section 8. All proofs are presented in the appendix.

## 2 Information Flow Properties

Information flow properties can be used to express confidentiality and integrity requirements. Each information flow property consists of two components: a *flow policy* and a *definition of information flow*. Such properties can be specified independently of any particular system specification. However, in order to refer to the underlying concepts it is necessary to choose a model of computation. In this article we use *event systems*.

An *event* is an atomic action with no duration, like e.g. sending or receiving a message on a communication channel. We distinguish *input events*, which cannot be enforced by the system, from *internal* and *output events*, which are controlled by the system. However, we do *not* make the restricting assumption that input events are always enabled. At the interface, input and output events can be observed while internal events cannot. The possible behaviours of a system are modeled as traces, i.e. sequences of events.

**Definition 1.** An *event system* $ES$ is a tuple $(E, I, O, Tr)$ where $E$ is a set of events, $I, O \subseteq E$ respectively are the input and output events, and $Tr \subseteq E^*$ is the set of traces, i.e. finite sequences over $E$. $Tr$ must be closed under prefixes.

*Example 1.* The event system $ES^a = (E, I, O, Tr^a)$ is the running example in this article. $E$ contains three events $l_1$, $l_2$, $h$, and $Tr^a$ contains all sequences in which each of these events occurs at most once, e.g. $l_2.h.l_1 \in Tr^a$. The distinction of input, internal, and output events will be unimportant in the sequel and, thus, we need not specify $I$ and $O$.

### 2.1 Flow Policies

*Flow policies* specify restrictions on the information flow within a system. They are defined with the help of a set $\mathcal{D}$ of *security domains*. Typical domains are e.g. groups of users, collections of files, or memory sections. We associate a security domain $dom(e) \in \mathcal{D}$ to each event $e \in E$.

**Definition 2.** A *flow policy FP* is a tuple $(\mathcal{D}, \leadsto_V, \leadsto_N, \not\leadsto)$ where $\leadsto_V, \leadsto_N, \not\leadsto \subseteq \mathcal{D} \times \mathcal{D}$ form a disjoint partition of $\mathcal{D} \times \mathcal{D}$ and $\leadsto_V$ is reflexive. *FP* is called *transitive* if $\leadsto_V$ is transitive and, otherwise, *intransitive*. In this article, we only consider transitive flow policies.

$\not\leadsto$ is the *non-interference relation* of *FP* and $D_1 \not\leadsto D_2$ expresses that there shall be no information flow from $D_1$ to $D_2$. Allowed information flow is specified by the two relations $\leadsto_V$ and $\leadsto_N$. $D_1 \leadsto_V D_2$ expresses that events in $D_1$ are visible for $D_2$. $D_1 \leadsto_N D_2$ expresses that occurrences of events from $D_1$ are invisible for $D_2$ but that we do not care if these occurrences can be deduced by $D_2$. Distinguishing between $\not\leadsto$, $\leadsto_V$, and $\leadsto_N$ results in more flexibility compared to having only two relations $\not\leadsto$ and $\leadsto$. However, for the moment, the reader can safely ignore the relation $\leadsto_N$ because, with the exception of Section 6, we will assume $\leadsto_N = \emptyset$.

*Example 2.* The *2-level flow policy* $FP_2$ has two domains, a low-level $L$ and a high-level $H$, i.e. $\mathcal{D} = \{L, H\}$. The relations are defined by $L \leadsto_V L$, $L \leadsto_V H$, $H \leadsto_V H$, and $H \not\leadsto L$, thus, there shall be no information flow from the high- to the low-level. For $ES^a$ we assign domains as follows: $dom(l_1) = dom(l_2) = L$ and $dom(h) = H$.

**Notational Convention.** We frequently use $D$ to denote the set of all events with domain $D$. E.g. for $ES^a$ we obtain $L = \{l_1, l_2\}$ and $H = \{h\}$. The *projection* $\alpha|_{E'}$ of a sequence $\alpha \in E^*$ to the events in $E' \subseteq E$ results from $\alpha$ by deleting all events *not* in $E'$, e.g. $l_2.h.l_1|_L = l_2.l_1$ and $l_2.h.l_1|_H = h$. The empty sequence is denoted by $\langle\rangle$.

### 2.2 Definitions of Information Flow

In order to state precisely under which conditions a system satisfies the restrictions described by a flow policy, it is necessary to define formally what information flow means. At least for non-deterministic systems, there is no agreement on a single definition of information flow but rather different definitions co-exist. Which of these definitions is best cannot be answered in general but depends on the particular application under consideration. In order to simplify their comparison, several frameworks have been proposed in which the various definitions of information flow can be uniformly represented [McL94, FG95, ZL97, Man00a].

Usually, a definition of information flow is parametric in the flow policy. In order to simplify the presentation we use a fixed flow policy, the two-level flow policy $FP_2$ from Example 2. Moreover, we focus on a single definition of information flow, the so called *perfect security property* from [ZL97] (abbreviated by *PSP* in the sequel). How to relax these assumptions will be discussed in Section 6. *PSP* can be formally defined by

$$\begin{aligned} \mathit{PSP}(\mathit{Tr}) \equiv\ & \forall \tau \in \mathit{Tr}. \forall \alpha, \beta \in E^*. \forall h \in E. \\ & \tau|_L \in \mathit{Tr} \\ & \wedge [(\tau = \beta.\alpha \wedge h \in H \wedge \alpha|_H = \langle\rangle \wedge \beta.h \in \mathit{Tr}) \\ & \quad \Rightarrow \beta.h.\alpha \in \mathit{Tr}] \end{aligned}$$

Intuitively, $\mathit{PSP}(\mathit{Tr})$ ensures that an observer who knows the specification of a system, i.e. *Tr*, and observes events with domain $L$ cannot deduce any information about occurrences of events with domain $H$. This prevents a high-level Trojan horse from transmitting information to low-level users. When observing a behaviour $\tau$, a low-level user observes $\tau|_L$. From this observation and the knowledge of *Tr*, he cannot decide whether $\tau$ or $\tau|_L$ has occurred. This is ensured by the first conjunct in the definition of *PSP*, which demands that $\tau|_L$ is a possible behaviour. Consequently, the low-level user cannot deduce that any high-level events *have* occurred. The second conjunct prevents a low-level user to deduce that some admissible high-level event $h$ *has not* occurred. All low-level observations $\alpha$ that are enabled after a behaviour $\beta$ ($\beta.\alpha \in \mathit{Tr}$) must also be enabled after $\beta.h$ ($\beta.h.\alpha \in \mathit{Tr}$) if $h$ is enabled after $\beta$ ($\beta.h \in \mathit{Tr}$). Therefore, the low-level user cannot rule out the possibility that $h$ has occurred. By induction, this can be generalized to arbitrary sequences of high-level events.

## 2.3 Unwinding Conditions

Defining information flow properties in terms of whole traces (like in the definition of *PSP* in the preceeding subsection) improves the understandability of these properties. However, for proving them it is helpful to have more local conditions, so called *unwinding conditions*, which are formulated in terms of single events. Clearly, it is desirable to have two alternative representations, a global definition as well as unwinding conditions together with an *unwinding theorem*, which ensures that the unwinding conditions imply the global definition.

For the definition of unwinding conditions it is necessary to enrich event systems with states. With this enrichment the pre-condition of an event $e$ is the set of states in which $e$ can possibly occur. The post-condition is a function from states to the set of possible states after the event has occurred in the respective state. The notion of state is transparent, however, note that information flow is only caused by events and not by states.

$lr_H$:

$$\begin{tikzcd} \mathbf{s_1} \arrow[r, "\mathbf{h}"] \arrow[r, bend right, dash, "\approx_\mathbf{L}"'] & \mathbf{s_2} \end{tikzcd}$$

$osc_{L,\emptyset,H}$:

$$\begin{tikzcd} \mathbf{s_1'} \arrow[r, "\mathbf{l}"] \arrow[d, dash, "\approx_\mathbf{L}"] & s_2' \arrow[d, dash, "\approx_\mathbf{L}"] \\ \mathbf{s_1} \arrow[r, "\mathbf{l}"] & \mathbf{s_2} \end{tikzcd}$$

*where* $h \in H$ *and* $l \in L$

**Figure 1. Unwinding conditions for PSP**

**Definition 3.** A *state-event system* $SES$ is a quintuple $(S, S_I, E, I, O, T)$ where $S$ is a set of states, $S_I \subseteq S$ contains the initial states, $E$ is a set of events, $I, O \subseteq E$ respectively contain the input and output events, and $T \subseteq S \times E \times S$ is a transition relation.

In this article, we assume that $S_I$ is a singleton set and that the effect of events is deterministic, i.e. that $T$ is functional. Note that state-event systems are still non-deterministic because of the non-determinism in the choice of events. A *history* of a state-event system $SES$ is a sequence $s_1.e_1.s_2 \ldots s_n$ of states and events. The set of histories $\mathit{Hist}(SES) \subseteq S \times (E \times S)^*$ for $SES$ is defined inductively as follows: if $s \in S_I$ then $s \in \mathit{Hist}(SES)$; if $s_1.e_1.s_2 \ldots s_n \in \mathit{Hist}(SES)$ and $(s_n, e_n, s_{n+1}) \in T$ then $s_1.e_1.s_2 \ldots s_n.e_n.s_{n+1} \in \mathit{Hist}(SES)$. A state $s \in S$ is *reachable*, denoted by $\mathit{reachable}(s)$, if there is a history $s_1.e_1 \ldots s \in \mathit{Hist}(SES)$ that ends in $s$. Each state-event system $SES = (S, S_I, E, I, O, T)$ can be translated into a corresponding event system $ES_{SES} = (E, I, O, \mathit{Tr}_{SES})$ where the set of traces $\mathit{Tr}_{SES} \subseteq E^*$ results from $\mathit{Hist}(SES)$ by deleting states from the histories.

**Definition 4.** An event system $ES = (E, I, O, \mathit{Tr})$ *satisfies PSP* if and only if $\mathit{PSP}(\mathit{Tr})$ holds. A state-event system $SES$ *satisfies PSP* if and only if $\mathit{PSP}(\mathit{Tr}_{SES})$ holds.

We now define two unwinding conditions $lr_H$ and $osc_{L,\emptyset,H}$. In these conditions an equivalence relation $\approx_L \subseteq S \times S$, the *unwinding relation*, is used. $s_1 \approx_L s_2$ shall express that $s_1$ and $s_2$ cannot be distinguished by a low-level user, i.e. all sequences of low-level events that are enabled in $s_1$ are also enabled in $s_2$, and vice versa. This is ensured by $osc_{L,\emptyset,H}$. $lr_H$ demands that the states before and after the occurrence of a high-level event are equivalent wrt. $\approx_L$.

$$\begin{aligned} lr_H:\ & \forall s_1, s_2 \in S. \forall h \in H. \\ & ((\mathit{reachable}(s_1) \wedge (s_1, h, s_2) \in T) \Rightarrow s_1 \approx_L s_2) \end{aligned}$$

$$\begin{aligned} osc_{L,\emptyset,H}:\ & \forall s_1, s_1', s_2 \in S. \forall l \in E \setminus H. \\ & (s_1 \approx_L s_1' \wedge (s_1, l, s_2) \in T) \\ & \Rightarrow [\exists s_2' \in S.((s_1', l, s_2') \in T \wedge s_2 \approx_L s_2')] \end{aligned}$$

$lr_H$ and $osc_{L,\emptyset,H}$ are equivalent to the commutativity respectively of the diagrams depicted on the left and right

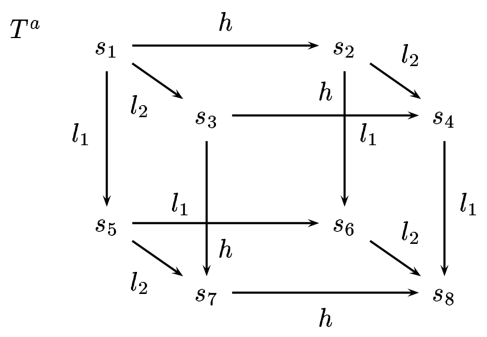

**Figure 2. Example transition relation $T^a$**

hand side in Figure 1. The parts of the diagrams which are given are indicated by boldface. All other parts may be chosen arbitrarily in order to make the diagrams commute.

**Theorem 1 (Unwinding Theorem).** *If there is an equivalence relation $\approx_L \subseteq S \times S$ for which $SES$ fulfills $lr_H$ and $osc_{L,\emptyset,H}$ then $SES$ also fulfills PSP.*

*Example 3.* In order to prove that $ES^a$ satisfies *PSP* we enrich $ES^a$ by states and obtain the state-event system $SES^a = (S, S_I, E, I, O, T^a)$ with $S = \{s_1, \ldots, s_8\}$, $S_I = \{s_1\}$, and the transition relation $T^a$ depicted in Figure 2. Let $\approx_L$ be the smallest equivalence relation with $s_1 \approx_L s_2$, $s_3 \approx_L s_4$, $s_5 \approx_L s_6$, and $s_7 \approx_L s_8$. With these definitions it is easy to check that $SES^a$ satisfies $lr_H$ and $osc_{L,\emptyset,H}$ for $\approx_L$. Thus, according to Theorem 1, $SES^a$ (and $ES^a$) fulfills *PSP*.

## 3 Specifications and Refinement

In this article we follow a semantic approach to the specification of systems. A system is specified by a specification *Spec* which defines its possible behaviours, i.e. a set $Tr_{Spec}$ of traces, plus some additional information. For event systems this additional information amounts to the set $E$ of events and the sets $I$ and $O$. For state-event systems this additionally includes a set $S$ of states, a subset $S_I$ of initial states, and a set of histories which results from an enrichment of the traces by states. However, the results in this article are not restricted to these two formalism but they can be used with any specification formalism for which a mapping into event systems or state-event systems exists. This includes process algebras like Hoare's CSP [Hoa85].

### 3.1 Satisfaction of Properties

Alpern and Schneider [AS85] define a property $P$ as a predicate on traces.[1] Alternatively, a property can be formalized as the set of all traces which satisfy $P$, i.e. $Tr_P = \{\tau \in E^* \mid P(\tau)\}$. A specification *Spec* satisfies $P$ if any behaviour that complies with *Spec* satisfies $P$, i.e. if $Tr_{Spec} \subseteq Tr_P$ holds.

[1]To be precise, properties of [AS85] correspond to predicates on infinite traces while in this paper they are predicates on traces of finite length.

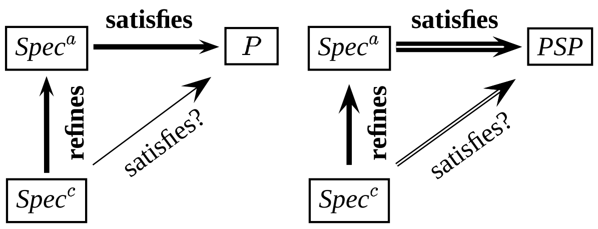

**Figure 3. Preservation under refinement**

**Definition 5.** An event system $ES = (E, I, O, Tr)$ *satisfies a property* $P$ if and only if $Tr \subseteq Tr_P$. A state-event system $SES$ *satisfies a property* $P$ if and only if $Tr_{SES} \subseteq Tr_P$.

Since traces are inductively defined in this article, they are always of finite length. Thus, no liveness but only safety properties will be considered. Note that information flow properties *cannot* be expressed as predicates of single traces. Rather they are closure conditions on sets of traces [McL94]. Thus, the notion of satisfaction in Definition 4 differs from that in Definition 5.

### 3.2 Refinement

An abstract specification $Spec^a$ is *refined* by a specification $Spec^c$ if all behaviours allowed by $Spec^c$ are also allowed by $Spec^a$, i.e. if $Tr_{Spec^c} \subseteq Tr_{Spec^a}$. Thus, refinement corresponds to the removal of non-determinism. The underlying idea is that an abstract specification focuses on *what* shall be achieved while the concrete specification is more specific on *how* to achieve this. The design decisions made in $Spec^c$ restrict the possible behaviours. This notion of refinement corresponds to the one investigated in [Jac89].

**Definition 6.** An event system $ES^c = (E, I, O, Tr^c)$ *refines* an event system $ES^a = (E, I, O, Tr^a)$ if and only if $Tr^c \subseteq Tr^a$. A state-event system $SES^c = (S, S_I, E, I, O, T^c)$ *refines* a state-event system $SES^a = (S, S_I, E, I, O, T^a)$ if and only if $T^c \subseteq T^a$.

The main topic of this article is the preservation of properties under refinement. The problem for a property $P$ in the sense of [AS85] is depicted on the left hand side of Figure 3. Assuming that $Spec^a$ satisfies $P$ and that $Spec^c$ refines $Spec^a$, the question is if $Spec^c$ also satisfies $P$. That this indeed holds is ensured by Theorem 2 below. The underlying argument is based on the transitivity of $\subseteq$ which is used in Definition 5 as well as in Definition 6. Unfortunately, this argument cannot be applied for information flow properties like *PSP* because satisfaction of these properties is not based on the subset relation (cf. Definition 4). This difference in the notion of satisfaction is indicated on the right hand side of Figure 3 by double-lined arrows.

**Theorem 2.** *If $ES^a$ satisfies a property $P$ and $ES^c$ refines $ES^a$ then $ES^c$ satisfies $P$. If $SES^a$ satisfies $P$ and $SES^c$ refines $SES^a$ then $SES^c$ satisfies $P$.*

## 4 Preserving Information Flow Properties

It is well known that information flow properties, like *PSP*, are not preserved under refinement. In Section 4.1, we first illustrate this problem by a simple example and then identify the source of this problem. In Section 4.2, we introduce refinement operators for *PSP* and prove that these operators preserve not only properties in the sense of [AS85] but also *PSP*. To which extent our operators are necessary for this preservation is not discussed in the current section but will be the topic of Section 5. Adaptations of the results from this section for other information flow properties will be presented in Section 6.

### 4.1 The Problem with Refinement

We now illustrate the problem of refining information flow properties by a simple example. For this purpose recall the state event system $SES^a$ from Example 3 .

*Example 4.* The transition relation $T^a$ of $SES^a$ (depicted in Figure 2) allows all sequences in which each of $l_1$, $l_2$, and $h$ occurs at most once. A very simple example for a design decision would be to require a certain ordering of these occurrences, e.g. that $l_2$ may only occur after $l_1$ or $h$ has already occurred. This results in the more concrete event system $SES^c = (S, S_I, E, I, O, T^c)$. $T^c$ is depicted in Figure 4. The main difference of $T^c$ to $T^a$ is that $l_2$ is disabled in state $s_1$. Consequently, state $s_3$ cannot be reached any more and, thus, the events $l_1$ and $h$ can also be disabled in $s_3$. Clearly, $SES^c$ is a refinement of $SES^a$ ($T^c \subseteq T^a$) and $SES^a$ satisfies *PSP* (cf. Example 3). However, $SES^c$ *does not fulfill PSP.* A low-level user, who makes the observation $l_2$ of $SES^c$ can deduce that $h$ has occurred. This is reflected by a violation of the first conjunct in the definition of *PSP* ($l_2 \notin Tr^c$). Thus, *PSP is, in general, not preserved under refinement.*

In the previous example, *PSP* is not preserved under refinement. Disabling $(s_1, l_2, s_3)$, $(s_3, h, s_4)$, and $(s_3, l_1, s_7)$ leads to a specification which is not closed under *PSP*. Clearly, such refinements must be ruled out in order to preserve *PSP*. The task therefore is to find restrictions to refinement which guarantee the preservation of *PSP*. According to the diagram on the right hand side of Figure 3, the information which could possibly provide a basis for such restrictions includes the two specifications *Spec*$^a$ and *Spec*$^c$,

$$\begin{tikzcd}
T^c & & & \\
s_1 \arrow[rr, "h"] \arrow[dd, "l_1"'] & & s_2 \arrow[dr, "l_2"] \arrow[dd, "l_1"] & \\
& & & s_4 \arrow[dd, "l_1"] \\
s_5 \arrow[rr, "h"'] \arrow[dr, "l_2"'] & & s_6 \arrow[dr, "l_2"] & \\
& s_7 \arrow[rr, "h"'] & & s_8
\end{tikzcd}$$

**Figure 4. Example for an insecure refinement**

the definition of *PSP*, and the proof that *Spec*$^a$ satisfies *PSP*. It will turn out that the proof of the satisfaction relation is, for this purpose, the most useful piece of information. The main idea is to reduce the problem of preserving *PSP* under refinement to the problem of preserving commutativity of the diagrams from Figure 1. This approach can be applied whenever *PSP* is proved using the unwinding theorem.

### 4.2 Refinement Operators for PSP

Rather than presenting conditions which could just be used to *check* if a given refinement preserves *PSP*, we present operators which can be used to *refine* specifications. When these refinement operators are applied, *PSP* is preserved by construction and no further check is necessary.

In particular, we present two refinement operators: $\underline{\textit{refine}}$ and $\overline{\textit{refine}}$. Each of these operators takes three arguments, an event system $SES$ which shall be refined, a set $DS \subseteq S \times E$ of state-event pairs which shall be disabled during the refinement, an unwinding relation $\approx_L \subseteq S \times S$, and yields a new, refined state-event system. Clearly, the goal is that if $SES$ satisfies *PSP* (or more precisely: if $SES$ satisfies *lr*$_H$ and *osc*$_{L,\emptyset,H}$ for $\approx_L$) then the resulting state-event system should also satisfy *PSP*. Simply disabling all pairs in $DS$ from $SES$, however, might yield a system which does *not* satisfy *PSP* (cf. Example 4). Therefore, an adaptation is necessary in which either *additional events are disabled* or *some state-event pairs in $DS$ remain enabled*. These two approaches give rise to a difference between $\underline{\textit{refine}}$ and $\overline{\textit{refine}}$. While $\underline{\textit{refine}}$ disables all pairs in $DS$ (plus possibly pairs not in $DS$), $\overline{\textit{refine}}$ only disables pairs which are in $DS$ (but possibly not all pairs in $DS$).

The formal definitions of $\underline{\textit{refine}}$ and $\overline{\textit{refine}}$ are depicted in Figure 5. These operators only have an impact on the transition relation. The new transition relation is constructed by the functions $\underline{\textit{disable}}$ and $\overline{\textit{disable}}$. These functions handle the high- and low-level events which shall be disabled separately using the functions *Hdisable*, $\underline{\textit{Ldisable}}$, and $\overline{\textit{Ldisable}}$. This separation is motivated by the unwinding conditions

$$\begin{array}{|l|}\hline \underline{\mathit{refine}}(SES, DS, \approx_L) \\ = (S, S_I, E, I, O, \underline{\mathit{disable}}(T, DS, \approx_L)) \\ \overline{\mathit{refine}}(SES, DS, \approx_L) \\ = (S, S_I, E, I, O, \overline{\mathit{disable}}(T, DS, \approx_L)) \\ \hline \underline{\mathit{disable}}(T, DS, \approx_L) \\ = \mathit{Hdisable}(T, DS_H) \cap \underline{\mathit{Ldisable}}(T, DS_L, \approx_L) \\ \overline{\mathit{disable}}(T, DS, \approx_L) \\ = \mathit{Hdisable}(T, DS_H) \cap \overline{\mathit{Ldisable}}(T, DS_L, \approx_L) \\ \hline \end{array}$$

*where* $SES = (S, S_I, E, I, O, T)$*,* $DS_H = \{(s,h) \in DS \mid h \in H\}$*,*
*and* $DS_L = \{(s,l) \in DS \mid l \in L\}$

**Figure 5. Refinement operators for PSP**

$lr_H$ and $osc_{L,\emptyset,H}$ which, respectively, are also only concerned with high- and low-level events (cf. Figure 1).

*Hdisable*, $\underline{\mathit{Ldisable}}$, and $\overline{\mathit{Ldisable}}$ are formally defined in Figure 6. *Hdisable* is used to disable high-level events. The transition relation $\mathit{Hdisable}(T, DS_H)$ equals $T$ except for that pairs in $DS_H$ are disabled. Since such a change does not give rise to new proof obligations (according to $lr_H$ and $osc_{L,\emptyset,H}$) no adaptation is required. However, after disabling low-level events an adaptation becomes necessary. The transition relation $\{(s_1, l, s_2) \in T \mid (s_1, l) \notin DS_L\}$ which one receives from $T$ by simply disabling all pairs in $DS_L$ might violate $osc_{L,\emptyset,H}$ (depending on $DS_L$ and $\approx_L$). Recall that $osc_{L,\emptyset,H}$ demands that if a low-level event is disabled in some state then it must be disabled in all (wrt. $\approx_L$) equivalent states as well. There are two approaches to satisfy this requirement: Either one disables *at least* all pairs in $DS_L$ (plus possibly additional ones) or one disables *at most* all pairs in $DS_L$ (but possibly not all of them). This gives rise to the difference between $\underline{\mathit{Ldisable}}$ and $\overline{\mathit{Ldisable}}$. If an event $l$ shall be disabled in some state $s_1'$ ($(s_1', l) \in DS_L$) then $\underline{\mathit{Ldisable}}(T, DS_L, \approx_L)$ disables $l$ not only in $s_1'$ but also in all states $s_1$ which are equivalent ($s_1 \approx_L s_1'$). $\overline{\mathit{Ldisable}}(T, DS_L, \approx_L)$ disables an event $l$ in state $s_1$ only if $(s_1, l) \in DS_L$ and if $(s_1', l) \in DS_L$ for all states $s_1'$ which are equivalent to $s_1$ ($s_1' \approx_L s_1$). Thus, it can happen that $\overline{\mathit{Ldisable}}(T, DS_L, \approx_L) = T$ although $DS_L \neq \emptyset$, i.e. for inappropriate $DS_L$, $\overline{\mathit{Ldisable}}$ may refuse refinement.

The under- and overlining indicates that $\underline{\mathit{Ldisable}}$ may yield a smaller transition relation than a simple disabling of pairs in $DS_L$, and that $\overline{\mathit{Ldisable}}$ may yield a larger relation. The following subset relations hold in general.[2]

$$\begin{array}{l} \underline{\mathit{Ldisable}}(T, DS_L, \approx_L) \\ \subseteq \{(s_1, e, s_2) \in T \mid (s_1, e) \notin DS_L\} \\ \subseteq \overline{\mathit{Ldisable}}(T, DS_L, \approx_L) \quad \subseteq T \end{array}$$

$$\begin{array}{|l|}\hline \mathit{Hdisable}(T, DS_H) \\ = \{(s_1, e, s_2) \in T \mid (s_1, e) \notin DS_H\} \\ \hline \underline{\mathit{Ldisable}}(T, DS_L, \approx_L) \\ = \{(s_1, e, s_2) \in T \mid (s_1, e) \notin DS_L \wedge \\ \quad \neg \exists s_1', s_2' \in S. \\ \quad (s_1 \approx_L s_1' \wedge (s_1', e, s_2') \in T \wedge (s_1', e) \in DS_L)\} \\ \hline \overline{\mathit{Ldisable}}(T, DS_L, \approx_L) \\ = \{(s_1', e, s_2') \in T \mid (s_1', e) \in DS_L \Rightarrow \\ \quad \exists s_1, s_2 \in S. \\ \quad (s_1 \approx_L s_1' \wedge (s_1, e, s_2) \in T \wedge (s_1, e) \notin DS_L)\} \\ \hline \end{array}$$

*assuming* $DS_H \subseteq S \times H$ *and* $DS_L \subseteq S \times L$

**Figure 6. Hdisable, $\underline{\textbf{Ldisable}}$, and $\overline{\textbf{Ldisable}}$**

**Theorem 3 (Preservation of [AS85] properties).** *Let $P$ be a property, $\approx_L \subseteq S \times S$ be a relation, and $DS \subseteq S \times E$ be a set of state-event pairs. If $SES$ satisfies $P$ then $\underline{\mathit{refine}}(SES, DS, \approx_L)$ and $\overline{\mathit{refine}}(SES, DS, \approx_L)$ satisfy $P$.*

**Theorem 4 (Preservation of unwinding conditions).** *Let $DS \subseteq S \times E$ be a set of state-event pairs. If $SES$ satisfies $lr_H$ and $osc_{L,\emptyset,H}$ for an equivalence relation $\approx_L \subseteq S \times S$ then $\underline{\mathit{refine}}(SES, DS, \approx_L)$ and $\overline{\mathit{refine}}(SES, DS, \approx_L)$ both satisfy $lr_H$ and $osc_{L,\emptyset,H}$ for $\approx_L$.*

Theorem 4 (together with Theorem 1) gives rise to the following corollary.

**Corollary 1 (Preservation of *PSP*).** *Let $\approx_L \subseteq S \times S$ be an equivalence relation and $DS \subseteq S \times E$ be a set of state-event pairs. If $SES$ satisfies $lr_H$ and $osc_{L,\emptyset,H}$ for $\approx_L$ then $\underline{\mathit{refine}}(SES, DS, \approx_L)$ and $\overline{\mathit{refine}}(SES, DS, \approx_L)$ satisfy PSP.*

Theorem 3 shows that properties in the sense of [AS85] are preserved under our refinement operators. Note that this theorem does not require anything about the unwinding relation $\approx_L$. Thus, one is not forced to investigate information flow properties at the very beginning of a development process. However, according to Corollary 1 one may migrate to a secure specification, which satisfies *PSP*, at any step of the development by proving the unwinding conditions $lr_H$ and $osc_{L,\emptyset,H}$. After these unwinding conditions have been proved, *PSP* will be preserved under further refinements with our operators. Usually in such a process, *PSP* will be proved using unwinding conditions anyway. For this reason, we do not consider this assumption of our approach a real restriction.

We now illustrate the use of the refinement operators at the state-event system $SES^a$ from Example 3.

[2]The first three transition relations are equal if there are no $s_1, s_1'$ with $s_1 \approx_L s_1'$, $(s_1, e) \in DS_L$, and $(s_1', e) \notin DS_L$.

$\underline{T}^c = \underline{disable}(T^a, DS, \approx_L)$

$$\begin{tikzcd}
s_1 \arrow[rr, "h"] \arrow[d, "l_1"'] & & s_2 \arrow[d, "l_1"] & \\
s_5 \arrow[rr, "h"] \arrow[dr, "l_2"'] & & s_6 \arrow[dr, "l_2"] & \\
& s_7 \arrow[rr, "h"'] & & s_8
\end{tikzcd}$$

$\overline{T}^c = \overline{disable}(T^a, DS, \approx_L)$

$$\begin{tikzcd}
s_1 \arrow[rr, "h"] \arrow[dr, "l_2"'] \arrow[dd, "l_1"'] & & s_2 \arrow[dr, "l_2"] \arrow[dd, "l_1"] & \\
& s_3 \arrow[dd, "h"] & & s_4 \arrow[dd, "l_1"] \\
s_5 \arrow[rr, "l_1"] \arrow[dr, "l_2"'] & & s_6 \arrow[dr, "l_2"] & \\
& s_7 \arrow[rr, "h"'] & & s_8
\end{tikzcd}$$

**Figure 7. Example application of operators**

*Example 5.* $SES^c$ results from refining $SES^a$ by disabling the pairs in $DS = \{(s_1, l_2), (s_3, h), (s_3, l_1)\}$ in Example 4. Although $SES^a$ satisfies *PSP*, $SES^c$ does not satisfy *PSP*. We now illustrate how $SES^a$ can be refined using our refinement operators. Let $T^a$ and $\approx_L$ be defined as in Example 3. Let $\underline{SES}^c = \underline{refine}(SES^a, DS, \approx_L)$ and $\overline{SES}^c = \overline{refine}(SES^a, DS, \approx_L)$. The refined transition relations $\underline{T}^c$ (for $\underline{SES}^c$) and $\overline{T}^c$ (for $\overline{SES}^c$) are depicted in Figure 7. In $\underline{T}^c$ the transitions $(s_2, l_2, s_4)$ and $(s_4, l_1, s_8)$ have been disabled although $(s_2, l_2), (s_4, l_1) \notin DS$. In $\overline{T}^c$ only $(s_3, h, s_4)$ has been disabled but $(s_1, l_2, s_3)$ and $(s_3, l_1, s_7)$ from $DS$ remain enabled. According to Corollary 1, $\underline{SES}^c$ and $\overline{SES}^c$ satisfy *PSP*.

## 5 Optimality of Refinement Operators

The functions *Hdisable*, *$\underline{Ldisable}$*, and *$\overline{Ldisable}$* adapt the transition relation so that the unwinding conditions $lr_H$ and $osc_{L,\emptyset,H}$ are satisfied by the resulting transition relation (assuming that they are satisfied by the original transition relation). How good these adaptations are, is a natural question to ask. In particular, one would like to know if any additional disabling is unnecessary (in the case of *$\underline{Ldisable}$*) and if any of the state-event pairs in $DS_L$ which remain enabled could be safely disabled (in the case of *$\overline{Ldisable}$*). These questions are the topic of the current section.

One prerequisite for our subsequent results will be the use of a minimal unwinding relation $\approx_L$.

**Theorem 5.** *If there is an equivalence relation $\approx_L$ for which $SES$ satisfies $lr_H$ and $osc_{L,\emptyset,H}$ for $\approx_L$ then there is a unique minimal relation with this property.*

The following theorem shows that our refinement operators are optimal if one starts with a minimal unwinding relation and makes some assumptions about the set $DS$. Here, *optimality* means for $\underline{refine}$ that state-event pairs not in $DS$ are not disabled unnecessarily and for $\overline{refine}$ that as many state-event pairs in $DS$ are disabled as possible without endangering *PSP*.

**Theorem 6 (One-step optimality).** *Let $T \subseteq S \times E \times S$ be a transition relation.*

1. *Let $DS \subseteq S \times E$ be a set of state-event pairs such that for all $(s_1, e), (s_1', e') \in DS$ holds $e, e' \in H$ or $s_1 \approx_L s_1'$, $e = e'$, and $e \in L$. Let $\approx_L$ be the minimal equivalence relation for which $T$ satisfies $lr_H$ and $osc_{L,\emptyset,H}$. Then $\underline{disable}(T, DS, \approx_L)$ is the* maximal *sub-relation of $T$, in which all pairs from $DS$ are disabled, and which fulfills $lr_H$ and $osc_{L,\emptyset,H}$.*

2. *Let $DS \subseteq S \times E$ be a set of state-event pairs such that for all $(s_1, e), (s_1', e') \in DS$ holds $e, e' \in H$ or $s_1 \approx_L s_1'$, $e = e'$, and $e \in L$. Let $\approx_L$ be the minimal equivalence relation for which $T$ satisfies $lr_H$ and $osc_{L,\emptyset,H}$. Then $\overline{disable}(T, DS, \approx_L)$ is the* minimal *sub-relation of $T$, in which only pairs from $DS$ are disabled, and which fulfills $lr_H$ and $osc_{L,\emptyset,H}$.*

Theorem 5 ensures the existence of a unique minimal unwinding relation. How to construct this minimal unwinding relation in practice, however, is a problem which is outside the scope of this article. Even if one starts with a minimal relation $\approx_L$ and applies our refinement operators then $\approx_L$ often will not be minimal for the resulting transition relation. A possible solution is to minimize the unwinding relation after each refinement step. This would require additional effort during refinement. However, that loosing minimality will turn out to be a significant problem in practice, is not certain. More experiences with applications of information flow control in formal system developments would be helpful to answer this. Although some practical experiments exist (e.g. [SRS$^+$00]), there still is a shortage of case studies with information flow control today.

Methods for constructing minimal unwinding relations and for minimizing unwinding relations after refinement might be desirable. However, such methods are outside the scope of this article.

## 6 Generalizations

In this section we generalize our results from Section 4. In particular, we present refinement operators for other information flow properties than *PSP* and demonstrate how to apply our results to other flow policies than the two-level flow policy $FP_2$. All refinement operators proposed in this section preserve properties in the sense of [AS85] as well as the respective unwinding conditions.

### 6.1 Beyond $FP_2$

In the previous sections, we have focused on a fixed flow policy, the two-level flow policy $FP_2$. However, one receives the full flexibility of information flow properties only by clearly separating flow policies and definitions of information flow. The use of $FP_2$ induces two assumptions: there is only one non-interference requirement, i.e. $|\not\rightsquigarrow| = 1$, and all invisible events must not be deducible, i.e. $\rightsquigarrow_N = \emptyset$. Note that $|\not\rightsquigarrow| = 1$ subsumes the simplification that there are only 2 domains (for transitive flow policies). In the following we demonstrate how to relax the assumption $|\not\rightsquigarrow| = 1$.

In order to show that a (transitive) flow policy with $|\not\rightsquigarrow| > 1$ is satisfied one needs to prove the unwinding conditions for every domain in the range of $\not\rightsquigarrow$. Intuitively, these unwinding conditions demand for a given domain $D \in \mathcal{D}$ that no information about events in $C = \bigcup\{D' \in \mathcal{D} \mid D' \not\rightsquigarrow D\}$ can be inferred from observations in $V = \bigcup\{D' \in \mathcal{D} \mid D' \rightsquigarrow_V D\}$. Thus, the unwinding conditions need to be parametric in $V$ and $C$. One can construct parametric unwinding conditions from $lr_H$ and $osc_{L,\emptyset,H}$ in Figure 1 by replacing $L$ by $V$ and $H$ by $C$. If $\rightsquigarrow_N \neq \emptyset$ then there would be a third parameter $N = \bigcup\{D' \in \mathcal{D} \mid D' \rightsquigarrow_N D\}$. We refer to the triple $V$, $N$, $C$ as the *view of domain $D$*. Unwinding conditions which are not only parametric in $V$ and $C$ but also in $N$ will be investigated in Section 6.2. In this subsection, we retain the assumption $\rightsquigarrow_N = \emptyset$.

For $|\not\rightsquigarrow| > 1$, it is essential that the unwinding conditions for the view of every domain is preserved during refinement. However, adapting a transition relation by investigating the different domains independently from each other is dangerous because adaptations for one domain might have side effects on other domains which have already been investigated. This kind of problem has been discussed in [Jac89] (cf. Theorem 2 in [Jac89]). We solve the problem by investigating all domains simultaneously rather than one after another during refinement. A refinement operator $\underline{\textit{refine}}$ which can handle flow policies with $|\not\rightsquigarrow| > 1$ is presented in Figure 8. Note that this version of $\underline{\textit{refine}}$ takes a family of unwinding relations as third argument rather than a single relation. The key idea is to construct the transitive closure of the union of the unwinding relations for all domains for which the event of a state-event pair in $DS$ is visible. An operator $\overline{\textit{refine}}$ could be constructed analogously.

$$\begin{aligned}
&\underline{\textit{refine}}(SES, DS, (\approx_D)_{D\in\mathcal{D}}) \\
&= (S, S_I, E, I, O, \underline{\textit{disable}}(T, DS, (\approx_D)_{D\in\mathcal{D}}))
\end{aligned}$$

$$\begin{aligned}
&\underline{\textit{disable}}(T, DS, (\approx_D)_{D\in\mathcal{D}}) \\
&= \textstyle\bigcap_{(s_1', e')\in DS} \\
&\quad \underline{\textit{Edisable}}(T, (s_1', e'), (\textstyle\bigcup\{\approx_{D'} \mid \textit{dom}(e') \rightsquigarrow_V D'\})^*)
\end{aligned}$$

$$\begin{aligned}
&\underline{\textit{Edisable}}(T, (s_1', e'), \approx) \\
&= \{(s_1, e, s_2) \in T \mid e = e' \Rightarrow \\
&\qquad [s_1 \neq s_1' \wedge \neg\exists s_2' \in S.(s_1' \approx s_1 \wedge (s_1', e', s_2') \in T)]\}
\end{aligned}$$

*where $SES = (S, S_I, E, I, O, T)$ and $(\bigcup\{\approx_{D'} \mid \textit{dom}(e') \rightsquigarrow_V D'\})^*$ is the transitive closure of $\bigcup\{\approx_{D'} \mid \textit{dom}(e') \rightsquigarrow_V D'\}$*

**Figure 8. Refinement operator for PSP and flow policies with $|\not\rightsquigarrow| > 1$ and $\rightsquigarrow_N = \emptyset$**

### 6.2 Other Information Flow Properties

The key idea in the construction of our refinement operators for *PSP* in Section 4 has been to make use of the unwinding relation $\approx_L$. This suggests that similar refinement operators can be developed for other information flow properties if appropriate unwinding results exist.

Below we develop refinement operators basing on the results in [Mil94] and [Man00b]. The unwinding conditions in these articles differ considerably from the ones in Section 4. The unwinding results in [Mil94] are concerned with *forward correctability* [JT88], an information flow property which behaves nicely under composition. Unwinding conditions for a class of very primitive information flow properties, so called *basic security predicates*, are presented in [Man00b]. Interestingly, these unwinding conditions are based on pre-orders, an approach, which is more flexible than the traditional use of equivalence relations. Basic security predicates can be used for a modular construction of more complicated information flow properties and the unwinding conditions from [Man00b] can be applied for proving various information flow properties, including non-inference [O'H90], generalized non-inference [McL94], generalized non-interference [McC87], separability [McL94], and the pretty good security predicate [Man00a].

In this subsection we drop the assumption $\rightsquigarrow_N = \emptyset$. Recall that $D_1 \rightsquigarrow_N D_2$ expresses that occurrences of events in $D_1$ are invisible for $D_2$ but that we do not care if they can be deduced by $D_2$. Note that replacing the statement $D_1 \rightsquigarrow_N D_2$ by either $D_1 \not\rightsquigarrow D_2$ or $D_1 \rightsquigarrow_V D_2$ would be more restrictive: $D_1 \not\rightsquigarrow D_2$ would demand that occurrences of events in $D_1$ cannot be deduced by $D_2$ and $D_1 \rightsquigarrow_V D_2$ would assume that $D_2$ can observe occurrences of events in $D_1$. Consequently, having $\rightsquigarrow_N$ as a third relation, in addition to $\not\rightsquigarrow$ and $\rightsquigarrow_V$, creates additional flexibility for specifying flow policies. In [Man00b] we have shown that a less restrictive version of the unwinding condition *osc* is appropriate for flow policies with $\rightsquigarrow_N \neq \emptyset$ which simplifies the proof of the unwinding conditions. However, for preserving this version of *osc* under refinement, the information contained in the unwinding relation does not suffice. Thus, we will introduce a relation $\hookleftarrow_V$ below which is used by the refinement operators. Like the unwinding relation, $\hookleftarrow_V$ can be derived from a proof of the unwinding conditions.

### 6.2.1 Operators for Basic Security Predicates

In [Man00b] two classes of basic security predicates are investigated $BSD_{V,N,C}$ and $BSIA_{V,N,C}$. Due to space restrictions we refer to that article for formal definitions and motivations of the basic security predicates. In order to simplify the presentation, we assume a fixed flow policy $FP_3$ which has three domains $V$, $N$, $C$, and for which $C \not\rightsquigarrow V$, $N \rightsquigarrow_N V$, and $\rightsquigarrow_V = (\mathcal{D} \times \mathcal{D}) \setminus \{(C,V),(N,V)\}$ hold. Since $C \not\rightsquigarrow V$ is the only non-interference requirement, unwinding conditions need only be investigated for domain $V$. For this domain, all events in $V$ are visible and all events in $C$ are confidential. Events in $N$ have a special role because information about them may be deduced from observations in $V$, however, this must not reveal any information about events from $C$.[3]

In [Man00b], pre-orders are used as unwinding relations. A *pre-order* $\ltimes_V \subseteq S \times S$ is reflexive and transitive, but need neither be symmetric nor antisymmetric. The unwinding conditions are depicted in Figure 9. The use of pre-orders splits *lr* into the conditions *lrf* and *lrb* which differ in the direction of $\ltimes_V$. The diagram for $osc_{V,N,C}$ is similar to the one for $osc_{L,\emptyset,H}$ in Figure 1. The only difference is the dashed arrow from $s'_1$ to $s'_2$ which is labeled by $\gamma'$. $\gamma' \in (V \cup N)^*$ must be a sequence of events which induces the same observation as $e$, i.e. $\gamma'|_V = e|_V$. This difference to the diagram in Figure 1 originates from dropping the assumption $\rightsquigarrow_N = \emptyset$.

**Theorem 7 (Unwinding Theorem).** *If there is a pre-order $\ltimes_V \subseteq S \times S$ for which $SES$ fulfills $osc_{V,N,C}$ then the following implications are valid: $lrf_C \Rightarrow BSD_{V,N,C}(Tr)$ and $lrb_C \Rightarrow BSIA_{V,N,C}(Tr)$.*

Based on this theorem we derive refinement operators for $BSD_{V,N,C}$ and $BSIA_{V,N,C}$. The dashed arrow in the diagram of *osc* causes some difficulties in this construction.

[3]Using views as parameters of $BSD_{V,N,C}$ and $BSIA_{V,N,C}$ results in a slightly different notation. $H_c$ and $H_a$ in [Man00b] correspond respectively to our $C$ and $N$ while $L \cup H_o$ corresponds to our $V$.

*$lrf_C$*:

$$\begin{tikzcd}
\mathbf{s_1} \arrow[r, "\mathbf{c}"] \arrow[r, dash, bend right, "\rtimes_{\mathbf{V}}"'] & \mathbf{s_2}
\end{tikzcd}$$

*$lrb_C$*:

$$\begin{tikzcd}
\mathbf{s_1} \arrow[r, "\mathbf{c}"] \arrow[r, dash, bend right, "\ltimes_{\mathbf{V}}"'] & \mathbf{s_2}
\end{tikzcd}$$

*$osc_{V,N,C}$*:

$$\begin{tikzcd}
\mathbf{s'_1} \arrow[r, dashed, "\gamma'"] & s'_2 \\
\mathbf{s_1} \arrow[r, "\mathbf{e}"] \arrow[u, dash, "\ltimes_V"'] & \mathbf{s_2} \arrow[u, dash, "\ltimes_V"']
\end{tikzcd}$$

*where $c \in C$, $e \in V \cup N$, $\gamma' \in (V \cup N)^*$, and $\gamma'|_V = e|_V$*

**Figure 9. Unwinding conditions for the Basic Security Predicates** *BSD* **and** *BSIA*

Simply following the same approach as in Section 4 would result in refinement operators which are very complicated and tedious to use. In order to construct usable operators, we extract further information from the proof of the unwinding conditions. This information is encoded by a relation $\hookleftarrow_V \subseteq (S \times E) \times (S \times E)$. If $\gamma'$ has been used in the proof that $osc_{V,N,C}$ holds for $s_1$ and $e$ then $(s_1, e) \hookleftarrow_V (s'_1, e')$ shall express that $e'$ occurs in $\gamma'$ and that $s'$ is the starting state of this occurrence of $e'$.

To construct a fixed relation $\hookleftarrow_V$ from a given proof of the unwinding conditions would be possible. However, we prefer to specify this relation declaratively because this provides more flexibility. For this purpose, we impose an additional restriction on $osc_{V,N,C}$. We say that $osc_{V,N,C}$ is satisfied for $(\ltimes_V, \hookleftarrow_V)$ if for every $s_1, s_2, s'_1 \in S$ with $s_1 \ltimes_V s'_1$ and every $e \in V \cup N$ with $(s_1, e, s_2) \in T$ there are $s'_2 \in S$ and $\gamma' \in E^*$ such that the diagram for *osc* in Figure 9 commutes. Let $\delta = s^0.e^1.s^1.e^2.s^2.\ldots.s^n \in S \times (E \times S)^*$ be the unique sequence with $s^0 = s'_1$, $s^n = s'_2$, $\delta|_E = \gamma'$, and $\forall i < n : (s^i, e^{i+1}, s^{i+1}) \in T$. The additional restriction we impose on *osc* is that $(s_1, e) \hookleftarrow_V (s^i, e^{i+1})$ holds for all $i < n$. Note that the same relation $\hookleftarrow_V$ must be used in the proof of the commutativity for all instances of the diagram. Intuitively, $(s_1, e) \hookleftarrow_V (s^i, e^{i+1})$ expresses that $(s^i, e^{i+1})$ was used to rule out information leakage by $(s_1, e)$.

In Figure 10, a refinement operator <u>*refine*</u> for $BSD_{V,N,C}$ and $BSIA_{V,N,C}$ is defined. Interestingly, the use of pre-orders does not make any difference from the perspective of refinement. The same refinement operator can be used for *BSD* as well as for *BSIA*. The additional complexity in this subsection originates purely from dropping the assumption $N = \emptyset$. If $N \neq \emptyset$ then the use of $\hookleftarrow_V$ seems to be absolutely essential for the construction of usable refinement operators which preserve the unwinding conditions. Due to

$$\begin{array}{l}
\underline{refine}(SES, DS, \hookleftarrow_V^*) \\
= (S, S_I, E, I, O, \underline{Edisable}(T, DS, \hookleftarrow_V^*)) \\
\hline
\underline{Edisable}(T, DS, \hookleftarrow_V^*) \\
= \{(s_1, e, s_2) \in T \mid (s_1, e) \notin DS \wedge \\
\quad \neg\exists s_1' \in S.\exists e' \in E. \\
\quad ((s_1, e) \hookleftarrow_V^* (s_1', e') \wedge (s_1', e') \in DS)\}
\end{array}$$

*where $SES = (S, S_I, E, I, O, T)$ and $\hookleftarrow_V^*$ denotes the reflexive and transitive closure of $\hookleftarrow_V$.*

**Figure 10. Refinement operator for the Basic Security Predicates *BSD* and *BSIA***

space restrictions we only define $\underline{refine}$. An operator $\overline{refine}$ could be constructed analogously.

### 6.2.2 Operators for Forward Correctability

Unwinding conditions for forward correctability have been presented in [Mil94]. Forward correctability was proposed in [JT88] as an improvement of restrictiveness [McC87]. For a formal definition and motivation of forward correctability we refer the reader to [JT88, Mil94].

Forward correctability requires of a system that it is input total, i.e. all input events must be enabled in all states. This gives rise to another unwinding condition ($t_I$) in Figure 11. The flow policy assumed by forward correctability is an instantiation of $FP_3$ (from Section 6.2.1) in which $V = L$, $C = H \cap I$, and $N = H \setminus I$ hold. Further unwinding conditions are the corresponding instantiations of *lr* and *osc*. Moreover, two additional unwinding conditions, *fwc*$^1$ and *fwc*$^2$, are concerned with subsequent occurrences of high- and low-level inputs. *fwc*$^1$ and *fwc*$^2$ are necessary in order to capture the nice composability properties of forward correctability (cf. [JT88]). Altogether, there are five unwinding conditions, which are depicted in Figure 11.

The following unwinding theorem corresponds to that part of Theorem 2.1.11 in [Mil94] which ensures the correctness of the unwinding conditions. In [Mil94], the *maximal* equivalence relation is used in the unwinding conditions. However, for the purpose of refinement, it is best to choose the unwinding relation as small as possible (cf. Section 5). Therefore, we specify the unwinding relation by commutativity requirements (like in Section 4) rather than choosing a specific relation.

**Theorem 8.** *If $SES$ fulfills $t_I$, $lr_{HI}$, $fwc^1_{LI,HI}$, $fwc^2_{LI,HI}$, and $osc_{L,H\setminus HI,HI}$ for some equivalence relation $\approx_L \subseteq S \times S$ then $SES$ fulfills forward correctability.*

During refinement, the commutativity of the diagrams for $lr_{HI}$ is trivially preserved. Assuming that $t_I$ is preserved,

$t_I$ :
$$\begin{tikzcd}
\mathbf{s_1} \arrow[r, "\mathbf{i}"] & s_2
\end{tikzcd}$$

$lr_{HI}$ :
$$\begin{tikzcd}
\mathbf{s_1} \arrow[r, "\mathbf{hi}"] \arrow[r, leftrightarrow, bend right, "\approx_L"'] & s_2
\end{tikzcd}$$

$fwc^1_{LI,HI}$ :
$$\begin{tikzcd}
\mathbf{s_1'} \arrow[r, "\mathbf{li}"] & s_2' \arrow[d, dash, "\approx_L"] \\
\mathbf{s_1} \arrow[u, "\mathbf{hi}"] \arrow[r, "\mathbf{li}"] & \mathbf{s_2}
\end{tikzcd}$$

$fwc^2_{LI,HI}$ :
$$\begin{tikzcd}
\mathbf{s_1'} \arrow[r, "\mathbf{li}"] & \mathbf{s_2'} \arrow[d, dash, "\approx_L"] \\
\mathbf{s_1} \arrow[u, "\mathbf{hi}"] \arrow[r, "\mathbf{li}"] & s_2
\end{tikzcd}$$

$osc_{L,H\setminus HI,HI}$ :
$$\begin{tikzcd}
\mathbf{s_1'} \arrow[r, dashed, "\gamma'"] \arrow[d, dash, "\approx_L"] & s_2' \arrow[d, dash, "\approx_L"] \\
\mathbf{s_1} \arrow[r, "\mathbf{e}"] & \mathbf{s_2}
\end{tikzcd}$$

*where $HI = H \cap I$, $LI = L \cap I$, $i \in I$, $hi \in HI$, $li \in LI$, $e \in E \setminus HI$, $\gamma' \in (E \setminus HI)^*$, and $\gamma'|_L = e|_L$*

**Figure 11. Unwinding conditions for forward correctability**

the preservation of $fwc^1_{LI,HI}$ and $fwc^2_{LI,HI}$ also becomes trivial. Thus, the diagrams for $t_I$ and $osc_{L,H\setminus HI,HI}$ are the only ones which impose restrictions on refinement. A technique to preserve $osc_{L,H\setminus HI,HI}$ when $\rightsquigarrow_N \neq \emptyset$ holds, has already been developed in Section 6.2.1. The additional problem of preserving $t_I$ can be handled by ignoring state-event pairs in $DS$ which involve an input event. The resulting refinement operator $\underline{refine}$ is depicted in Figure 12. An operator $\overline{refine}$ could be constructed analogously.

## 7 Comparison to Related Work

The common reference for stating that information flow control is incompatible with refinement is [Jac89]. Two major difficulties for a stepwise development of secure systems are identified at the example of the information flow property "ignorance of progress". Firstly, security orderings are, in general, neither monotonic nor anti-monotonic with respect to the safety ordering. This implies that information flow properties are, in general, not preserved under refinement. Secondly, two security orderings need not be monotonic with respect to each other. Thus, the security requirements of different domains cannot be established independently. Furthermore, in [Jac89] a method for stepwise development is proposed. Roughly, this method corresponds to first refining the specification until it could directly be implemented and then making this specification secure by

$$\begin{aligned}&\underline{\mathit{refine}}(SES, DS, \hookleftarrow_L^*)\\&= (S, S_I, E, I, O, \underline{\mathit{Edisable}}(T, DS, \hookleftarrow_L^*))\end{aligned}$$

$$\begin{aligned}&\underline{\mathit{Edisable}}(T, DS, \hookleftarrow_L^*)\\&= \{(s_1, e, s_2) \in T \mid e \notin I \Rightarrow [(s_1, e) \notin DS \wedge\\&\quad \neg\exists s_1' \in S. \exists e' \in E.\\&\quad\quad ((s_1, e) \hookleftarrow_L^* (s_1', e') \wedge (s_1', e') \in DS \wedge e' \notin I)]\}\end{aligned}$$

*where $SES = (S, S_I, E, I, O, T)$ and $\hookleftarrow_L^*$ denotes the reflexive and transitive closure of $\hookleftarrow_L$.*

**Figure 12. Refinement operator for forward correctability**

deleting traces from the specification using so called weakest trusted users. By iterating the later step for different security orderings, the security requirements of different domains are established independently from each other. However, it is neither ensured that this process terminates nor that it results in a useful live system. In comparison to our approach, [Jac89] does not propose a method for preserving information flow properties under refinement but rather one for making a specification secure after it has been sufficiently refined. To apply this method during refinement, appears to be infeasible because the complete specification must be re-investigated at every development step without taking advantage of investigations performed in previous steps. Another difference is that the method is based on the global deletion of complete traces while our approach is based on the more local disabling of single events. Finally, we consider the security requirements for all domains simultaneously during refinement rather than one after another, thereby overcoming the second difficulty for stepwise development described in [Jac89].

In [GCS91], data refinement is discussed for a security property which is quite similar to *PSP*. Also the unwinding conditions correspond to our unwinding conditions in Section 2.3. However, one difference is the use of *maximal* unwinding relations. Data refinement allows one to refine the notion of state but cannot be used to reduce nondeterminism of an abstract specification by disabling events. Thus, data refinement differs from the notion of refinement we have considered in this article. In [GCS91], a condition is proposed under which the inspected information flow property is preserved. This condition requires that equivalence classes of the abstract specification are mapped to equivalence classes of the concrete specification. To prove this condition appears to be non-trivial because the maximal equivalence relations for the concrete and the abstract specification may differ considerably. However, no efficient technique for proving this condition is proposed.

In [O'H92], information flow is investigated using the framework of category theory. As a basis for concrete investigations, a specific category is selected which is based on sets of traces. Necessary conditions are presented for proving that a given specification satisfies a confidentiality statement. Confidentiality statements are distinguished from functional requirements. In [O'H92], the functional requirements are regarded as an upper bound and the confidentiality requirements as a lower bound. The task of a system designer is to develop a system which is between these two bounds. However, no efficient method for staying within the bounds is provided. Rather, it appears to be necessary to prove confidentiality statements for the refined specification from scratch after refinement. All results on preservation of information flow properties in [O'H92] are concerned with composition (parallel composition as well as choice) rather than with refinement.

In [RWW94] two definitions of information flow are presented which are based on determinism in the framework of the process algebra CSP [Hoa85]. Although both definitions differ, the common underlying idea is that there can be no information flow from $H$ to $L$ if the low-level behaviour is completely deterministic. An advantage is that these information flow properties are preserved under refinement. The refinements which are possible under the assumptions of this approach are similar to our refinement operator *Hdisable* (although the models of computation differ). However, a major disadvantage of this approach is that it imposes severe limitations on specifications. Requiring a deterministic low-level behaviour does not only rule out information flow but also forbids common forms of parallelism for the low-level and limits the possible abstractions in abstract specifications.

For recent results on the preservation of security properties which are no information flow properties, we refer to [Jür01]. The security property investigated in that article follows the approach of [DY83].

## 8 Conclusion

It has been well known that information flow properties are, in general, not preserved under refinement. However, we have demonstrated how refinement can be restricted such that these properties are preserved. The key idea has been to exploit knowledge about the proof of the information flow property that has already been performed on a more abstract level.

Basing on this idea, we have developed refinement operators for several information flow properties. These operators can be used to derive more concrete specifications from abstract specifications. We have proved that certain information flow properties are preserved under the respective refinement operators. Thus, the use of these information flow properties in a stepwise development process has become feasible. In particular, we have investigated the perfect security property (*PSP*) [ZL97], forward correctability [JT88], and basic security predicates [Man00a]. Various other information flow properties can be assembled from basic security predicates, including non-inference [O'H90], generalized non-inference [McL94], generalized non-interference [McC87], separability [McL94], and the pretty good security predicate [Man00a]. Since the unwinding conditions from [Man00b] can be used to prove these properties, it would be straightforward to develop corresponding refinement operators by adapting the ones for the basic security predicates.

In our approach, we consider the security requirements of all domains simultaneously during refinement when there is more than one non-interference requirement. This technique overcomes a problem described in [Jac89] which occurs if the security requirements of different domains are established one after another.

The approach which we have proposed in this article is, to the best of our knowledge, the first feasible approach to refining information flow properties (cf. our comparison in Section 7). Nevertheless, a few questions are left unanswered which could be valuable topics for future research. In particular, a method would be desirable which is feasible for constructing minimal unwinding relations in practice. Especially, because minimality is a prerequisite of our optimality result (cf. Theorem 6). Since minimality is not preserved during refinement without spending additional effort at each development step, an efficient method for minimizing the unwinding relation after each refinement step would be beneficial. However, the significance of these two open issues can, in our opinion, only be justified by experiments. Unfortunately, information about only very few case studies which involve information flow properties is publicly available (e.g. [SRS$^+$00]). Consequently, the development of further case studies seems to be a very important task for future research in this area.

**Acknowledgments.** The author would like to thank the anonymous referees for their useful suggestions. Thanks also to Alexandra Heidger and Jan Jürjens for many valuable comments on the presentation.

## References

[AS85] Bowen Alpern and Fred B. Schneider. Defining Liveness. *Information Processing Letters*, 21:181–185, 1985.

[DY83] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

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

[GCS91] John Graham-Cumming and J.W. Sanders. On the Refinement of Non-interference. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 35–42, Franconia, NH, 1991.

[GM82] 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.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, London, 1985.

[Jac89] Jeremy Jacob. On the Derivation of Secure Components. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 242–247, Oakland, CA, 1989.

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

[Jür01] Jan Jürjens. Secrecy-preserving Refinement. In *Proceedings of the Formal Methods Europe (FME)*, LNCS, Berlin, Germany, 2001. to appear

[Man00a] Heiko Mantel. Possibilistic Definitions of Security – An Assembly Kit –. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 185–199, Cambridge, UK, 2000.

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

[McC87] Daryl McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 161–166, Oakland, CA, 1987.

[McL94] John 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.

[Mil94] Jonathan K. Millen. Unwinding Forward Correctability. In *Proceedings of the Computer Security Foundations Workshop*, pages 2–10, Franconia, NH, 1994.

[O'H90] Colin O'Halloran. A Calculus of Information Flow. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 1990.

[O'H92] Colin O'Halloran. Refinement and Confidentiality. In *Proceedings of the 5th Refinement Workshop*, Workshops in Computing, pages 119–139, London, UK, 1992.

[RWW94] A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Non-interference through Determinism. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, LNCS 875, pages 33–53, Brighton, UK, 1994.

[SRS+00] Gerhard Schellhorn, Wolfgang Reif, Axel Schairer, Paul Karger, Vernon Austel, and David Toll. Verification of a Formal Security Model for Multiapplicative Smart Cards. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, LNCS 1895, pages 17–36, Toulouse, France, 2000.

[WJ90] J. Todd Wittbold and Dale M. Johnson. Information Flow in Nondeterministic Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 144–161, Oakland, CA, 1990.

[ZL97] Aris Zakinthinos and E.S. Lee. A General Theory of Security Properties. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 94–102, Oakland, CA, 1997.

## Appendix

This appendix contains the proofs for all theorems in the article.

*Proof (Theorem 1).* Follows from the entry for *PSP* in Table 1 of [Man00b].

*Proof (Theorem 2).* Follows from Definition 5, Definition 6, and the transitivity of $\subseteq$.

*Proof (Theorem 3).* Follows from $\underline{\mathit{disable}}(T, DS, \approx_L) \subseteq \overline{\mathit{disable}}(T, DS, \approx_L) \subseteq T$ and Definition 5.

*Proof (Theorem 4).* Let $DS_H = \{(s,h) \in DS \mid h \in H\}$ and $DS_L = \{(s,l) \in DS \mid l \in L\}$. According to the definitions of $\underline{\mathit{refine}}$ and $\overline{\mathit{refine}}$ we need to prove the following three statements.

1. $T_1^c = \mathit{Hdisable}(T^a, DS_H)$ satisfies $\mathit{lr}_H$ as well as $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$.
2. $T_2^c = \underline{\mathit{Ldisable}}(T^a, DS_L, \approx_L)$ satisfies $\mathit{lr}_H$ as well as $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$.
3. $T_3^c = \overline{\mathit{Ldisable}}(T^a, DS_L, \approx_L)$ satisfies $\mathit{lr}_H$ as well as $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$.

(1) If $h \in H$ and $(s_1, h, s_2) \in T_1^c$ then $(s_1, h, s_2) \in T^a$ by definition of *Hdisable*. Since $T^a$ satisfies $\mathit{lr}_H$ for $\approx_L$ so does $T_1^c$. For $l \in L$ holds $(s_1, l, s_2) \in T_1^c$ if and only if $(s_1, l, s_2) \in T^a$. Since $T^a$ satisfies $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$ so does $T_1^c$.
(2) For $h \in H$ holds $(s_1, h, s_2) \in T_2^c$ if and only if $(s_1, h, s_2) \in T^a$. Since $T^a$ satisfies $\mathit{lr}_H$ for $\approx_L$ so does $T_2^c$. $T^a$ satisfies $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$. We prove by contradiction that $T_2^c$ also satisfies $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$. Assume $s_1, s_1', s_2 \in S$ and $l \in L$ with $s_1 \approx_L s_1'$ and $(s_1, l, s_2) \in T_2^c$ such that there is no $s_2^*$ with $(s_1', l, s_2^*) \in T_2^c$ and $s_2 \approx_L s_2^*$. $(s_1, l, s_2) \in T^a$ because $T_2^c \subseteq T^a$. Since $T^a$ satisfies $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$ there is a state $s_2' \in S$ such that $(s_1', l, s_2') \in T^a$ and $s_2 \approx_L s_2'$. According to our initial assumption holds $(s_1', l, s_2') \notin T_2^c$. From the definition of $\underline{\mathit{Ldisable}}$ we conclude that $(s_1', l) \in DS_L$ or, otherwise, there are states $s_1'', s_2'' \in S$ with $s_1' \approx_L s_1''$, $(s_1'', l' s_2'') \in T^a$, and $(s_1'', l) \in DS_L$. In the first case the definition of $\underline{\mathit{Ldisable}}$ yields $(s_1, l, s_2) \notin T_2^c$ directly and in the second case by transitivity of $\approx_L$. This contradicts our initial assumption.
(3) For $h \in H$ holds $(s_1, h, s_2) \in T_3^c$ if and only if $(s_1, h, s_2) \in T^a$. Since $T^a$ satisfies $\mathit{lr}_H$ for $\approx_L$ so does $T_3^c$. $T^a$ satisfies $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$. We prove by contradiction that $T_3^c$ also satisfies $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$. Assume $s_1, s_1', s_2 \in S$ and $l \in L$ with $s_1 \approx_L s_1'$ and $(s_1, l, s_2) \in T_3^c$ such that there is no $s_2^*$ with $(s_1', l, s_2^*) \in T_3^c$ and $s_2 \approx_L s_2^*$. $(s_1, l, s_2) \in T^a$ because $T_3^c \subseteq T^a$. Since $T^a$ satisfies $\mathit{osc}_{L,\emptyset,H}$ for $\approx_L$ there is a state $s_2' \in S$ such that $(s_1', l, s_2') \in T^a$ and $s_2 \approx_L s_2'$. According to our initial assumption holds $(s_1', l, s_2') \notin T_3^c$. Thus, $(s_1', l) \in DS_L$. We now distinguish two cases: $(s_1, l) \notin DS_L$ and $(s_1, l) \in DS_L$. In the first case, we obtain $(s_1', l, s_2') \in T_3^c$ from $(s_1', l) \in DS_L$, $s_1 \approx_L s_1'$, $(s_1, l, s_2) \in T^a$, $(s_1, l) \notin DS_L$, and the definition of $\overline{\mathit{Ldisable}}$. In the second case, we conclude from $(s_1, l, s_2) \in T_3^c$ and the definition of $\overline{\mathit{Ldisable}}$ that there are states $s_1'', s_2'' \in S$ with $s_1'' \approx_L s_1$, $(s_1'', l, s_2'') \in T^a$, and $(s_1'', l) \notin DS_L$. We obtain $(s_1', l, s_2') \in T_3^c$ from the definition of $\overline{\mathit{Ldisable}}$ using the transitivity of $\approx_L$. Thus in both cases, we have a contradiction to our initial assumption.

*Proof (Theorem 5).* Assume $\sim = \{(s_1, s_2) \in S \times S \mid \mathit{reachable}(s_1) \wedge \exists h \in H.(s_1, h, s_2) \in T\}$. Closing $\sim$ under reflexivity, symmetry, transitivity, and $\mathit{osc}_{L,\emptyset,H}$ yields a minimal unwinding relation $\approx_L$. There are no choices in this construction (Recall from Section 2.3 that $T$ is functional. The choice of $s'_2$ is completely determined by $s'_1$ and $l$.). Thus, $\approx_L$ is the *unique* minimal unwinding relation.

*Proof (Theorem 6).* We only prove the first proposition in the theorem by contradiction. The second proposition can be proved analogously.

(1) Assume there is a *nonempty* set $T' \subseteq T$ with $T' \cap \underline{disable}(T, DS, \approx_L) = \emptyset$ and $\forall (s_1, e, s_2) \in T'.(s_1, e) \notin DS$ such that $T' \cup \underline{disable}(T, DS, \approx_L)$ satisfies *lr*$_H$ and *osc*$_{L,\emptyset,H}$ for $\approx_L$. Select $(s_1, e, s_2) \in T'$ arbitrarily. $e \in L$ must hold according to the assumptions about $T'$ and the definition of *Hdisable*. According to the definition of *<u>Ldisable</u>* there are states $s'_1, s'_2 \in S$ with $s_1 \approx_L s'_1$, $(s'_1, e, s'_2) \in T$, and $(s'_1, e) \in DS_L$. Summarizing, we have $e \in L$, $s_1 \approx_L s'_1$, and $(s_1, e, s_2) \in T' \cup \underline{disable}(T, DS, \approx_L)$ but there is no state $s'_2 \in S$ such that $(s'_1, e, s'_2) \in T' \cup \underline{disable}(T, DS, \approx_L)$. Thus $T' \cup \underline{disable}(T, DS, \approx_L)$ does *not* satisfy *osc*$_{L,\emptyset,H}$, a contradiction to our initial assumption.

*Proof (Theorem 7).* Theorem 7 corresponds to Theorem 3 in [Man00b].

*Proof (Theorem 8).* Theorem 8 is a specialization of the correctness part of Theorem 2.1.11 in [Mil94]. If there is an equivalence relation $\approx_L$ such that the diagrams for $t_I$, *lr*$_{HI}$, *fwc*$^1_{LI,HI}$, *fwc*$^2_{LI,HI}$, and *osc*$_{L,H\setminus HI,HI}$ commute then the conditions $(q/x) \sim q$ and $(q/x)/a \sim q/a$ from Theorem 2.1.11 are satisfied. $t_I$ implies the input totality requirement, *lr*$_{HI}$ ensures $(q/x) \approx_L q$, and *fwc*$^1_{LI,HI}$, *fwc*$^2_{LI,HI}$ together guarantee $(q/x)/a \approx_L q/a$. That $\approx_L \subseteq \sim$ holds, is ensured by *osc*$_{L,H\setminus HI,HI}$. Thus, according to Theorem 2.1.11 in [Mil94] $SES$ fulfills forward correctability.