Refinement Operators and Information Flow Security*

Annalisa Bossi, Riccardo Focardi, Carla Piazza, Sabina Rossi
Dipartimento di Informatica - Università Ca' Foscari di Venezia
e-mail:{bossi,focardi,piazza,srossi}@dsi.unive.it

Abstract

The systematic development of complex systems usually relies on a stepwise refinement procedure from an abstract specification to a more concrete one, that can finally be implemented. The use of refinement operators preserving system properties is clearly essential since it avoids properties to be re-investigated at each development step.

In this paper we formalize the notion of refinement *for processes described as terms of the Security Process Algebra (SPA). We consider several information flow security properties and provide sufficient conditions under which our refinement operators preserve such security properties. Finally, we study how refinements can be composed still preserving the security of the system.*

1. Introduction

In the recent years, security has gained more and more importance. As a matter of fact, many critical applications like, e.g., e-commerce, require high degrees of reliability and protection against external attacks. In a stepwise development process, it is important to consider security related issues from the very beginning. Indeed, considering security only at the implementation phase could lead to a poor protection or, even worst, could make it necessary to restart the development process from scratch. On the other hand, taking into account security from the abstract specification level, better integrates it in the whole development process, possibly driving some of the development choices.

A security-aware stepwise development, of course, requires that the security properties of interest are preserved during the development steps, until a concrete (i.e., implementable) specification is obtained. Thus, this *refinement* task should be carefully designed so that (also) security properties are preserved from the very abstract initial specification to the final one.

*This work has been partially supported by the MURST project "Modelli formali per la sicurezza" and the EU project IST-2001-32617 "Models and Types for Security in Mobile Distributed Systems" (MyThS).

In this paper we focus on *information flow* properties (see, e.g., [8, 4, 5, 7, 13, 22]), i.e., security properties that allow to express constraints on how information should flow among different groups of entities. These properties are usually formalized by considering only two groups of entities labelled with two security levels: *high* (H) and *low* (L). The only constraint is that no information should flow from H to L. For example, this security model can be applied to guarantee confidentiality in a system: it is sufficient to label every confidential (i.e., secret) information with H and then partition each system user as H and L, depending on whether such a user is or is not authorized to access confidential information. The constraint of no information flow from H to L guarantees that no access to confidential information is possible by L-labelled users.

Unfortunately, as noticed in [14], most information flow properties are not preserved by usual refinement operators. This can be understood by noticing that the absence of information flow is usually formalized as a *non-interference* constraint [8]. The absence of information flow from H to L is achieved by requiring something intuitively stronger: what is done by entities operating at level H should never be observable by entities operating at level L. More succinctly, level H should never *interfere* with level L. This non-observability of high level events cannot be expressed as a property of single execution sequences. Instead, it is captured by requiring that each high level event moves the system into a state that behaves the same as before from a low level point of view. It is clear that if we refine the system by modifying the low level behaviour of some of such system states, we could invalidate this low level behavioural invariant, thus invalidating the non-interference constraint.

In this work, we propose a new notion of refinement for processes described through the *Security Process Algebra* (SPA). For this new notion we prove some interesting properties: (i) it is incremental, i.e., subsequent refinements are still a refinement; (ii) it is compositional with respect to SPA operators, i.e., it is possible to refine a process by refining its subcomponents.

Then, we consider a number of information flow properties proposed in literature and we express them through the

following *generalized unwinding condition*:

$$\forall \text{ state } S \text{ if } S \xrightarrow{h} S' \text{ then } S \dashrightarrow S'' \text{ and } S' \backsim^{l} S'' \quad (1)$$

Intuitively, when a high level action h is performed moving from S to S', then S should be able to simulate such a move by moving to a state S'' which is equivalent to S' from a low level point of view. Condition (1) is parametric with respect to the simulating transition relation $\dashrightarrow$ and the low level behavioural equivalence $\backsim^{l}$. We show that, by plugging different relations and equivalences, we capture many different existing information flow properties.

This general form for expressing non-interference allows us to state a general refinement theorem: every refinement that preserves both $\dashrightarrow$ and $\backsim^{l}$ preserves also condition (1). This theorem proves that every non-interference property expressed using (1) is invariant with respect to any refinement that preserves the two specific relations $\dashrightarrow$ and $\backsim^{l}$ used to express the property.

The paper is organized as follows: in Section 2 we present the SPA calculus; in Section 3 we introduce the new refinement notion and we prove some general results on it; in Section 4 we give the generalized unwinding condition, together with the main refinement result, and we instantiate it to many existing information flow security properties; finally, in Section 5 we compare our work with existing literature and we give some concluding remarks. Along the paper, a simple running example of a binary memory cell is used to show how the refinement and security notions work. All the proofs can be found in the Technical Report available at `http://www.dsi.unive.it/~bossi/sefm03TR.ps`.

2. The SPA Language

In this section we report the syntax and semantics of the *Security Process Algebra* (*SPA*, for short) [5].

The *Security Process Algebra* [5] is a variation of Milner's CCS [19], where the set of visible actions is partitioned into high level actions and low level ones in order to specify multilevel systems. SPA syntax is based on the same elements as CCS that is: a set $\mathcal{L}$ of *visible* actions such that $\mathcal{L} = I \cup O$ where $I = \{a, b, \ldots\}$ is a set of *input* actions and $O = \{\bar{a}, \bar{b}, \ldots\}$ is a set of *output* actions; a special action τ which models internal computations, i.e., not visible outside the system; a complementary function $\bar{\cdot} : \mathcal{L} \rightarrow \mathcal{L}$, such that $\bar{\bar{a}} = a$, for all $a \in \mathcal{L}$. $Act = \mathcal{L} \cup \{\tau\}$ is the set of all *actions*. The set of visible actions is partitioned into two sets, H and L, of high and low actions such that $\overline{H} = H$ and $\overline{L} = L$. We will use h to denote a generic high level action, and l to denote a generic low level action. The syntax of SPA *processes* (or *systems*) is defined as follows:

$$E ::= \mathbf{0} \mid a.E \mid E + E \mid E|E \mid E \setminus v \mid E[f] \mid Z$$

where $a \in Act$, $v \subseteq \mathcal{L}$, $f : Act \rightarrow Act$ is such that $f(\bar{\alpha}) = \overline{f(\alpha)}$, $f(\tau) = \tau$, $f(h) \in H \cup \{\tau\}$ for $h \in H$, and $f(l) \in L \cup \{\tau\}$ for $l \in L$, and Z is a constant that must be associated with a definition $Z \stackrel{\text{def}}{=} E$.

Intuitively, $\mathbf{0}$ is the empty process that does nothing; $a.E$ is a process that can perform an action a and then behaves as E; $E_1 + E_2$ represents the nondeterministic choice between the two processes E_1 and E_2; $E_1|E_2$ is the parallel composition of E_1 and E_2, where executions are interleaved, possibly synchronized on complementary input/output actions, producing an internal action τ; $E \setminus v$ is a process E prevented from performing actions in v; $E[f]$ is the process E whose actions are renamed *via* the relabelling function f.

We denote by $\mathcal{E}$ the set of all SPA processes and by $\mathcal{E}_H$ the set of all high level processes, i.e., those constructed only using actions in $H \cup \{\tau\}$.

The operational semantics of SPA processes is given in terms of a *Labelled Transition System* (*LTS*, for short). A *LTS* is a triple $(S, A, \rightarrow)$ where S is a set of states, A is a set of labels (actions), $\rightarrow \subseteq S \times A \times S$ is a set of labelled transitions. The notation $(S_1, a, S_2) \in \rightarrow$ (or equivalently $S_1 \xrightarrow{a} S_2$) means that the process can move from the state S_1 to the state S_2 through the action a. The operational semantics of SPA is the *LTS* $(\mathcal{E}, Act, \rightarrow)$, where the states are the terms of the algebra and the transition relation $\rightarrow \subseteq \mathcal{E} \times Act \times \mathcal{E}$ is defined by structural induction as the least relation generated by the inference rules depicted in Figure 1. The operational semantics for a process E is the subpart of the SPA *LTS* reachable from the initial state.

The transition relation $\rightarrow$ is generalized to many steps as expected: If $t = a_1 \cdots a_n \in Act^*$ and $E \xrightarrow{a_1} \cdots \xrightarrow{a_n} E'$, then we say that E' is reachable from E and write $E \xrightarrow{t} E'$. $Reach(E)$ denotes the set of processes reachable from E. We say that E is a finite state process if $Reach(E)$ is finite.

Example 2.1 We consider an abstract specification M_x of a binary memory cell. M_x contains the binary value x and is accessible, by high and low users, through the four operations r_h, w_h, r_l, w_l representing a high read, a high write, a low read and a low write, respectively. Each operation is implemented through two different actions, one for each binary value. For example we write w_h_0 and w_h_1 to indicate a high level user writing value 0 and 1, respectively. [1]

$$\begin{aligned} M\_x \quad &\stackrel{\text{def}}{=} \quad \overline{r_h\_x} \,.\, M\_x + w_h\_0 \,.\, M\_0 + w_h\_1 \,.\, M\_1 \\ &+ \quad \overline{r_l\_x} \,.\, M\_x + w_l\_0 \,.\, M\_0 + w_l\_1 \,.\, M\_1 \end{aligned}$$

Notice that read (write) operations are modelled as outputs (inputs). In particular, process M_x can send the stored

[1] The following expression for M_x is indeed a definition scheme: the actual processes M_0 and M_1 are obtained by replacing x with 0 and 1, respectively.

$$\frac{-}{a.E \xrightarrow{a} E} \qquad \frac{E_1 \xrightarrow{a} E_1'}{E_1 + E_2 \xrightarrow{a} E_1'} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 + E_2 \xrightarrow{a} E_2'} \qquad \frac{E_1 \xrightarrow{a} E_1'}{E_1|E_2 \xrightarrow{a} E_1'|E_2} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1|E_2 \xrightarrow{a} E_1|E_2'}$$

$$\frac{E_1 \xrightarrow{a} E_1' \;\; E_2 \xrightarrow{\bar{a}} E_2'}{E_1|E_2 \xrightarrow{\tau} E_1'|E_2'}\, a \in \mathcal{L} \qquad \frac{E \xrightarrow{a} E'}{E \setminus v \xrightarrow{a} E' \setminus v} \text{ if } a \notin v \qquad \frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]} \qquad \frac{E \xrightarrow{a} E'}{Z \xrightarrow{a} E'} \text{ if } Z \stackrel{\text{def}}{=} E$$

Figure 1. The operational rules for SPA

value x through the two output actions $\overline{r_h\_x}$ and $\overline{r_l\_x}$. Moreover, write operations are performed by accepting an input w_h_y and w_l_y (with $y \in \{0,1\}$) and moving to M_y, i.e., storing y into the memory cell. The operational semantics of processes M_0 and M_1 is depicted in Figure 2.

Notice that M_0 and M_1 are totally insecure processes. As a matter of fact, no access control is implemented and a high level malicious entity (e.g., a Trojan Horse program) may write confidential information into the memory cell, through w_h_0, w_h_1. This information can be then read by any low level user through $\overline{r_l\_0}, \overline{r_l\_1}$. Information flow properties in Section 4 will aim at detecting this kind of flaws, even in more subtle and interesting situations.

The concept of *behavioural equivalence* is used to establish equalities among processes and it is based on the idea that two processes have the same semantics if and only if their behavior cannot be distinguished by an external observers. We report here a largely used equivalence, named (strong) *bisimulation* [19], which equates two processes when they are able to mutually simulate their behavior step by step.

Definition 2.2 (Strong Bisimulation) A binary relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ over processes is a *strong bisimulation* if $(E,F) \in \mathcal{R}$ implies, for all $a \in Act$,
- if $E \xrightarrow{a} E'$, then $\exists F'$ such that $F \xrightarrow{a} F'$ and $(E',F') \in \mathcal{R}$;
- if $F \xrightarrow{a} F'$, then $\exists E'$ such that $E \xrightarrow{a} E'$ and $(E',F') \in \mathcal{R}$.

Two processes E and F are *strongly bisimilar*, denoted by $E \sim_B F$, if there exists a strong bisimulation $\mathcal{R}$ containing the pair (E,F).

In Section 4 we will give other (weaker) notions of behavioural equivalence that will be useful for defining security properties.

3. Refinement Operators

In this section we give a new notion of refinement for SPA processes. Intuitively, an abstract specification (given here as a SPA system) defines the set of possible (allowed) behaviours of a system. Refining a specification corresponds to choosing among these allowed behaviours, the ones that will be actually implemented. The idea is that a refined specification should never show behaviours that were not foreseen in the initial specification. To formalize this idea, we require that (i) each state of the abstract specification is refined to, at most, one state of the more concrete (i.e., refined) specification; (ii) the behavior of the refined states is simulated by the abstract states, i.e., it should always be possible to simulate an action performed by a refined state by the corresponding abstract state, and the two reached states should be still one the refinement of the other.

Refinement is formalized as a partial function from processes. A binary relation $\mathcal{R}$ is said to be a *partial function* when if both $(x,y) \in \mathcal{R}$ and $(x,y') \in \mathcal{R}$ then y and y' are equal. If $\mathcal{R}$ is a partial function we use the notation $\mathcal{R}(x)\downarrow$ to denote the fact that there exists y such that $(x,y) \in \mathcal{R}$. In this case we also use the notation $\mathcal{R}(x)$ to refer to y. Similarly we denote by $\mathcal{R}(x)\uparrow$ the fact that there is no y such that $(x,y) \in \mathcal{R}$. $\mathcal{R}^{-1}$ indicates *the converse relation* of $\mathcal{R}$, i.e., $\mathcal{R}^{-1} = \{(y,x) \mid (x,y) \in \mathcal{R}\}$.

To express the fact that the refined process should be simulated by the abstract one, we adopt the following notion.

Definition 3.1 (Simulation) A binary relation $\mathcal{S} \subseteq \mathcal{E} \times \mathcal{E}$ over processes is a *simulation* if $(E,F) \in \mathcal{S}$ implies, for all $a \in Act$, if $E \xrightarrow{a} E'$, then there exists F' such that $F \xrightarrow{a} F'$ and $(E',F') \in \mathcal{S}$.
We say that the process E *is simulated by* the process F, denoted by $E \leq F$, if there exists a simulation $\mathcal{S}$ containing the pair (E,F).

The relation $\leq$ is a preorder, i.e., it is reflexive and transitive. Notice that, if a relation $\mathcal{S}$ is such that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are simulations, then $\mathcal{S}$ is a strong bisimulation.

We now introduce our formal definition of refinement.

Definition 3.2 (Refinement) A binary relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ over processes is a *refinement* if

- $\mathcal{R}^{-1}$ is a simulation and
- $\mathcal{R}$ is a partial function from $\mathcal{E}$ to $\mathcal{E}$.

We say that E *is a refinement of* F, denoted by $E \preceq F$, if there exists a refinement $\mathcal{R}$ such that $\mathcal{R}(F) = E$.

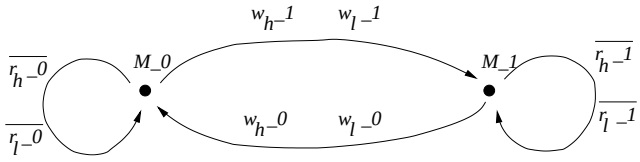

Figure 2. The operational semantics of the memory cell M_x.

The following example shows the difference between the notions of simulation and refinement. The condition that $\mathcal{R}$ is a (partial) function is useful to ensure that the refined process shows behaviors that are also shown by abstract one.

Example 3.3 Consider the processes $F_1 \stackrel{\text{def}}{=} a.b.\mathbf{0}$ and $F_2 \stackrel{\text{def}}{=} a.\mathbf{0} + a.b.\mathbf{0}$. Although both $F_1 \leq F_2$ and $F_2 \leq F_1$, we have that F_1 is a refinement of F_2 ($F_1 \preceq F_2$), but not viceversa ($F_2 \not\preceq F_1$). Indeed, it is easy to prove that $\mathcal{R} = \{(F_2, F_1), (b.\mathbf{0}, b.\mathbf{0}), (\mathbf{0}, \mathbf{0})\}$ is a refinement, thus $F_1 \preceq F_2$. On the other hand, in order to have $F_2 \preceq F_1$ we should consider a relation $\mathcal{R}'$ such that $\mathcal{R}'^{-1}$ is a simulation containing the pair (F_2, F_1). Since F_2 moves both to $b.\mathbf{0}$ and to $\mathbf{0}$ by performing a, and F_1 can only simulate this move by moving into $b.\mathbf{0}$, then $\mathcal{R}'^{-1}$ should contain (at least) the pairs $(b.\mathbf{0}, b.\mathbf{0})$ and $(\mathbf{0}, b.\mathbf{0})$. As a consequence $\mathcal{R}'$ cannot be a function since $b.\mathbf{0}$ is mapped into two different processes. This proves that $F_2 \not\preceq F_1$.

It is worth noticing that our notion of refinement is decidable for the class of processes we are considering. In fact, for finite states processes both the requirements in definition 3.2 are decidable. Moreover, the complexity of the decision procedure is bounded by the complexity of checking if a process F simulates another process E. This can be done in $\mathrm{O}(n \times m)$, where n is the total number of states of the two LTS's associated to E and F, and m is the total number of arcs (see [10]).
The following proposition follows immediately from the definition of refinement. It states that a sequence of actions performed by a refined state can always be simulated by the corresponding abstract state, and the two reached states are (still) one the refinement of the other. This is what we expected by our definition: all the behaviours of the refined process are simulated by the abstract process.

Proposition 3.4 Let $E, F \in \mathcal{E}$, $E \preceq F$ and $\mathcal{R}$ be a refinement such that $\mathcal{R}(F) = E$. If $E \stackrel{t}{\rightarrow} E'$ with $t \in Act^*$, then there exists F' such that $\mathcal{R}(F') = E'$ and $F \stackrel{t}{\rightarrow} F'$.

Example 3.5 Consider again the memory cell M_x of Example 2.1. We noticed that such a process is not secure as a direct information flow from high to low level is possible.

A standard way to protect confidential data is to apply the multilevel security model of [1]. First, we need to assign a security level to any information containers (called *objects*); then the following access control rules are imposed: (i) no low level user can read from high level objects; (ii) no high user can write into low level objects. Indeed, these are the only two (direct) ways for leaking confidential information.

M_x can be refined both into a high level cell M^h_x, by eliminating any low level read operation (rule (i)),

$$\begin{array}{rcl} M^h\_x & \stackrel{\text{def}}{=} & \overline{r_h\_x} \,.\, M^h\_x + w_h\_0 \,.\, M^h\_0 + w_h\_1 \,.\, M^h\_1 \\ & + & w_l\_0 \,.\, M^h\_0 + w_l\_1 \,.\, M^h\_1 \end{array}$$

and into a low level cell M^l_x, by eliminating any high level write operation (rule (ii)):

$$\begin{array}{rcl} M^l\_x & \stackrel{\text{def}}{=} & \overline{r_h\_x} \,.\, M^l\_x + \overline{r_l\_x} \,.\, M^l\_x \\ & + & w_l\_0 \,.\, M^l\_0 + w_l\_1 \,.\, M^l\_1 \end{array}$$

It is easy to see that $M^h\_x \preceq M\_x$ and $M^l\_x \preceq M\_x$, for each $x \in \{0, 1\}$, by considering the following two refinements: $\mathcal{R} = \{(M\_0, M^h\_0), (M\_0, M^h\_1)\}$ and $\mathcal{R}' = \{(M\_0, M^l\_0), (M\_0, M^l\_1)\}$.

In Section 4, we will show how to prove that M^h_x and M^l_x are secure, i.e., that they do not allow any information flow from high to low level.

The next theorem states that the composition of two refinements is still a refinement. We denote by $\circ$ the partial function composition, i.e., $(\mathcal{R}_1 \circ \mathcal{R}_2)(F) = \mathcal{R}_2(\mathcal{R}_1(F))$ if $\mathcal{R}_1(F)\downarrow$ and $(\mathcal{R}_1 \circ \mathcal{R}_2)(F)\uparrow$ if $\mathcal{R}_1(F)\uparrow$. This guarantees that our notion of refinement can be used in a stepwise refinement construction of a process.

Theorem 3.6 Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be two refinements. Then $\mathcal{R}_1 \circ \mathcal{R}_2$ is still a refinement.

Hence, $\preceq$ is a transitive relation. Since it is immediate to prove that $\preceq$ is reflexive, we get that $\preceq$ is a preorder.

Using Definition 3.2 we can introduce refinements which sound counter-intuitive, e.g., a refinement $\mathcal{R}$ with $\mathcal{R}(F)\downarrow$, $\mathcal{R}(F')\downarrow$, and such that F reaches F' but $\mathcal{R}(F)$ does not reach $\mathcal{R}(F')$.

Example 3.7 Let $F \equiv a.b.c.\mathbf{0} + a.\mathbf{0}$ and $E \equiv a.\mathbf{0}$. We have that E is a refinement of F. A relation $\mathcal{R}$ which proves that E refines F is the following $\mathcal{R} = \{(F,E),(c.\mathbf{0},c.\mathbf{0}),(\mathbf{0},\mathbf{0})\}$. This refinement is not intuitive, since it "does not preserve the reachability on its domain". A more natural refinement would be $\mathcal{R}' = \{(F,E),(\mathbf{0},\mathbf{0})\}$.

The following proposition states that if a process E refines a process F, there exists a refinement $\mathcal{R}_{(F,E)}$ which behaves "correctly" with respect to the notion of reachability.

Proposition 3.8 Let $E, F \in \mathcal{E}$ be two processes. If $E \preceq F$ and $\mathcal{R}$ is a refinement containing the pair (F,E), then the relation $\mathcal{R}_{(F,E)} = \mathcal{R} \cap (Reach(F) \times Reach(E))$ is a refinement such that: $R_{(F,E)}(F) = E$ and if $R_{(F,E)}(F')\downarrow$, then $F' \in Reach(F)$ and $E' \in Reach(E)$.

Let $\mathbb{R}_F$ be the set of processes E such that $E \preceq F$. Since $\preceq$ is a preorder on $\mathcal{E}$, $\preceq$ induces a preorder on $\mathbb{R}_F$.

Notice that $\preceq$ is not a partial order since it is not anti-symmetric, as shown in the following example.

Example 3.9 Let $F = a.b.\mathbf{0}$ and $F' = a.b.\mathbf{0} + a.b.\mathbf{0}$. In this case both $F' \preceq F$ and $F \preceq F'$.

If F is a finite state process and E is such that both F refines E and vice-versa, then F and E are strongly bisimilar. This result does not hold for infinite state processes.

Proposition 3.10 Let $F, E \in \mathcal{E}$, with F finite state. If $F \preceq E$ and $F' \preceq E$, then $F \sim_B E$.

Proposition 3.10 implies that if F is a finite state process, $\preceq$ induces on $\mathbb{R}_F$, up to strong bisimulation, a partial order with top element F and bottom element $\mathbf{0}$. This agrees with our intuition that F is the "largest" refinement for F.

Some natural refinements can be obtained by applying the basic CCS operators. In particular, the operation of restriction can be used to build refinements.

Example 3.11 For all process E and for all set of actions $v \subseteq \mathcal{L}$, the process $E\backslash v$ is a refinement of E, i.e., $E\backslash v \preceq E$. In fact, it is easy to prove that the relation $\mathcal{R}_{\backslash v} = \{(E, E \setminus v) \mid E \in \mathcal{E}\}$ is a refinement.

The following result explicates the relations between our notion of refinement and the basic CCS operators. It allows us to incrementally build refinements by combining refinements of process components. It shows also how to get the refinement of a process by refining just part of it.

Theorem 3.12 Let $F, G, E, I \in \mathcal{E}$. Let $\mathcal{R}$ be a refinement containing both (F,E) and (G,I).
- If $a.F \notin Reach(F)$, then $\mathcal{R}' = \mathcal{R}_{(F,E)} \cup \{(a.F, a.E)\}$ is a refinement;
- If $F + G \notin Reach(F) \cup Reach(G)$, then $\mathcal{R}' = \mathcal{R}_{(F,E)} \cup \mathcal{R}_{(G,I)} \cup \{(F+G, E+I)\}$ is a refinement;
- $\mathcal{R}' = \{(F'|G', E'|I') \mid (F',E'),(G',I') \in \mathcal{R}\}$ is a refinement;
- $\mathcal{R}' = \{(F' \setminus v, E' \setminus v) \mid (F',E') \in \mathcal{R}\}$ is a refinement;
- $\mathcal{R}' = \{(F'[f], E'[f]) \mid (F',E') \in \mathcal{R}\}$ is a refinement.

Corollary 3.13 Let $F, G, E, I \in \mathcal{E}$ be such that there exists a refinement containing both (F,E) and (G,I). Then:
- $a.E \preceq a.F$, if $a.F \notin Reach(F)$;
- $E + I \preceq F + G$, if $F + G \notin Reach(F) \cup Reach(G)$;
- $E|I \preceq F|G$;
- $E \setminus v \preceq F \setminus v$;
- $E[f] \preceq F[f]$.

4. Unwinding Based Security Properties and Refinement

In this section we consider various information flow security properties which can be characterized in terms of unwinding conditions and we identify sufficient conditions to be satisfied by our refinement operators in order to preserve them. We consider a class of unwinding conditions. Informally, these conditions require that whenever a high level action can be performed from a state E reaching a state E', there exists a state E'' which for the low level user is indistinguishable from E' and which is locally connected to E, given a suitable notion of connectivity. Since an unwinding condition can be seen as a sufficient condition for many security properties, its preservation under refinement implies the preservation of all the security properties it guarantees.

4.1 A Generalized Unwinding Condition

We give a uniform presentation of the properties we consider by introducing a generalized unwinding condition which is parametric with respect to two binary relations on processes: an equivalence relation, $\backsim^l$, which represents the low level indistinguishability and a transition relation, $\dashrightarrow$, which characterizes the local connectivity required by the unwinding condition.

Definition 4.1 (Generalized Unwinding) Let $\backsim^l$ and $\dashrightarrow$ be two binary relations on $\mathcal{E} \times \mathcal{E}$. We define $\mathcal{W}(\backsim^l, \dashrightarrow)$ as

$$\begin{aligned}\mathcal{W}(\backsim^l, \dashrightarrow) \stackrel{\text{def}}{=} \{E \in \mathcal{E} \mid & \\ & \forall\, F, G \in Reach(E), \text{ if } F \xrightarrow{h} G \\ & \text{then } \exists G' \text{ such that } F \dashrightarrow G' \text{ and } G \backsim^l G'\}\end{aligned}$$

The generalized unwinding condition is based on two binary relations on processes. Thus in order to preserve it we can just check the preservation under refinement of the low level observation and the connectivity relation. For total functions this means to look for a congruence with respect to the two relations. Since refinements are partial functions we require that also undefined ($\uparrow$) is preserved.

Definition 4.2 (Refinement preserving $\odot$) Let $\odot$ be a binary relation on $\mathcal{E} \times \mathcal{E}$ and $\mathcal{R}$ be a refinement. We say that $\mathcal{R}$ is a *refinement preserving* $\odot$ if for all G, G' such that $G \odot G'$ then either both $\mathcal{R}(G)\uparrow$ and $\mathcal{R}(G')\uparrow$ or both $\mathcal{R}(G)\downarrow$ and $\mathcal{R}(G')\downarrow$ and $\mathcal{R}(G) \odot \mathcal{R}(G')$.

The preservation under refinement of the two relations $\backsim^l$ and $\dashrightarrow$ is sufficient for guaranteeing the preservation under refinement of the $\mathcal{W}(\backsim^l, \dashrightarrow)$ property.

Theorem 4.3 Let $\backsim^l$ and $\dashrightarrow$ be two binary relations on $\mathcal{E} \times \mathcal{E}$ and $F \in \mathcal{W}(\backsim^l, \dashrightarrow)$. If $\mathcal{R}$ is a refinement preserving both $\backsim^l$ and $\dashrightarrow$ and $\mathcal{R}(F)\downarrow$, then $\mathcal{R}(F) \in \mathcal{W}(\backsim^l, \dashrightarrow)$.

The next lemma shows that the composition of two refinements preserving a given binary relation still preserves the same relation. That offers us a condition to preserve $\mathcal{W}(\backsim^l, \dashrightarrow)$ under subsequent refinements.

Lemma 4.4 Let $\odot$ be a binary relation on $\mathcal{E}$ and $\mathcal{R}_1$ and $\mathcal{R}_2$ be two refinements preserving $\odot$. Then $\mathcal{R}_1 \circ \mathcal{R}_2$ is still a refinement preserving $\odot$.

4.2. Properties based on Unwinding Conditions

Many security properties can be characterized as suitable instances of our generalized unwinding condition as shown in [2]. Here we consider both properties based on bisimulation (named, P_BNDC, $SBNDC$, and CP_BNDC) and properties based on trace equivalence (named, $SNDC$). The first three properties implies the well-known *BNDC* property [4] while the last one implies *NDC* [4] (or, equivalently, *NF* [20]) and, in the deterministic case, *PSP* [23].

4.2.1 Bisimulation based Security Properties

Each of the three properties based on bisimulation we consider in this section imply *Bisimulation based Non Deducibility on Composition* (*BNDC*, for short), a security property introduced in [4] which aims at guaranteeing that no information flow from the high to the low level is possible, even in the presence of malicious processes. It is based on the idea of checking the system against all high level potential interactions, representing every possible high level malicious program. In particular, a system E is *BNDC* if for every high level process Π a low level user cannot distinguish E from $(E|\Pi)$.

The low level observation of a process is formalized in terms of *weak bisimulation on low actions*. Weak bisimulation is similar to strong bisimulation, but it does not care about internal τ actions. It is useful to introduce some notation: we write $E \stackrel{t}{\Longrightarrow} E'$ if $E(\stackrel{\tau}{\rightarrow})^* \stackrel{a_1}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* \cdots (\stackrel{\tau}{\rightarrow})^* \stackrel{a_n}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* E'$, where $(\stackrel{\tau}{\rightarrow})^*$ denotes a (possibly empty) sequence of τ labelled transitions. If $t \in Act^*$, then $\hat{t} \in \mathcal{L}^*$ is the sequence gained by deleting all occurrences of τ from t. As a consequence, $E \stackrel{\hat{a}}{\Longrightarrow} E'$ stands for $E \stackrel{a}{\Longrightarrow} E'$ if $a \in \mathcal{L}$, and for $E(\stackrel{\tau}{\rightarrow})^* E'$ if $a = \tau$ (note that $\stackrel{\tau}{\Longrightarrow}$ requires at least one τ transition while $\stackrel{\hat{\tau}}{\Longrightarrow}$ means zero or more τ transitions).

Given this notation, weak bisimulation is obtained from strong bisimulation by allowing a transition of the form $\stackrel{a}{\rightarrow}$ to be simulated by a transition $\stackrel{\hat{a}}{\Rightarrow}$. Two processes are weakly bisimilar on low actions when they are weakly bisimilar if we consider only low actions.

Definition 4.5 (Weak Bisimulation on Low Actions) A binary relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ over processes is a *weak bisimulation on low actions* if $(E, F) \in \mathcal{R}$ implies, for all $a \in L \cup \{\tau\}$,
- if $E \stackrel{a}{\rightarrow} E'$, then $\exists\, F'$ such that $F \stackrel{\hat{a}}{\Longrightarrow} F'$ and $(E', F') \in \mathcal{R}$;
- if $F \stackrel{a}{\rightarrow} F'$, then $\exists\, E'$ such that $E \stackrel{\hat{a}}{\Longrightarrow} E'$ and $(E', F') \in \mathcal{R}$.

Two processes E and F are *weakly bisimilar on low actions*, denoted by $E \approx^l_B F$, if there exists a weak bisimulation on low actions $\mathcal{R}$ containing the pair (E, F).

Definition 4.6 (BNDC) Let $E \in \mathcal{E}$. $E \in BNDC$ if $\forall\, \Pi \in \mathcal{E}_H$, $E \approx^l_B (E|\Pi)$.

The properties *P_BNDC*, *SBNDC* and *CP_BNDC* are all sufficient conditions for *BNDC*. Moreover, differently from *BNDC*, they are *persistent* in the sense that if a process E satisfies one of them then all states rechable from E satisfy it. Here we define the three security properties as instances of our generalized unwinding condition. Clearly, since we are considering processes which are weakly bisimilar on low actions the relation $\backsim^l$ is instantiated by $\approx^l_B$. As regards $\dashrightarrow$, it varies according to the different requirements.

The security property called *Persistent_BNDC* [7] (*P_BNDC*, for short), which is suitable for analyzing systems in dynamic execution environments, requires that whenever a state F of a *P_BNDC* process may execute a high level action moving to a state G, then F should be also able to simulate such high move through a sequence of zero or more τ moving to a state G' which is equivalent to G for a low level user. Thus,

$$E \in P\_BNDC \iff E \in \mathcal{W}(\approx^l_B, \stackrel{\hat{\tau}}{\Longrightarrow})$$

As stated by Theorem 4.3, any refinement preserving both $\approx^l_B$ and $\stackrel{\hat{\tau}}{\Longrightarrow}$ preserves also *P_BNDC*.

Corollary 4.7 Let $F \in \mathcal{E}$ be a *P_BNDC* process. If $\mathcal{R}$ is a refinement preserving both $\approx^l_B$ and $\stackrel{\hat{\tau}}{\Longrightarrow}$ and $\mathcal{R}(F)\downarrow$, then $\mathcal{R}(F)$ is *P_BNDC*.

Example 4.8 We consider again the memory cell M_x defined in Example 2.1. We noticed that such a process is insecure because of a direct information flow. This fact is correctly revealed by *P_BNDC*.

In Example 3.5, we observed that this flaw can be repaired by refining the memory cell into a high and a low memory cell, defined as processes M^h_x and M^l_x, respectively. It is easy to see that $M^h\_0 \in P\_BNDC$. First, notice that $M^h\_0 \approx^l_B M^h\_1$, since there is no way for a low level user to distinguish between the two states. As a matter of fact, the only possible low level actions are the two write operations w_l_0, w_l_1 which, both in M^h_0 and in M^h_1, move the system into the same states. The fact that $M^l\_0 \in P\_BNDC$ is even easier to prove: the only high level actions r_h_0, r_h_1 do not change the system state.

We have proved that high and low level cells are secure. It is now interesting to study how this property is preserved by further refining the processes. To this aim we apply Theorem 4.7. Notice that neither M^h_0 nor M^l_0 perform any τ transitions, thus the only condition that we should care about is that the refinement preserves $\approx^l_B$. As a consequence, removing high level actions does not affect the security of the two systems. For example, if we allow the high level user to only reset the cell value to 0 (by removing the $w_h\_1 \,.\, M^h\_1$ branch), the resulting process is still secure.

On the other hand, modifications of low behavior should be performed coherently in all equivalent states. For example, the refinement

$$\begin{array}{lcl} N^h\_0 & \stackrel{\text{def}}{=} & \overline{r_h\_0} \,.\, N^h\_0 + w_h\_0 \,.\, N^h\_0 + w_h\_1 \,.\, N^h\_1 \\ N^h\_1 & \stackrel{\text{def}}{=} & \overline{r_h\_1} \,.\, N^h\_1 + w_h\_0 \,.\, N^h\_0 + w_h\_1 \,.\, N^h\_1 \\ & + & w_l\_0 \,.\, N^h\_0 \end{array}$$

in which the low level user can reset to 0 the high level cell, only when the cell contains value 1 (notice that in N^h_0 no low level write operations are allowed) is not preserving $\approx^l_B$. It is easy to see that $N^h\_0 \notin P\_BNDC$. The fact that N^h_0 is not *P_BNDC* reveals a slightly subtle information flow due to the fact that a low level user may track the content of the high level cell by trying to reset it: every time the reset succeeds the low level user can conclude that the cell contained value 1. A correct refinement achieving the same low level reset behavior described above, should include the branch $w_l\_0 \,.\, N^h\_0$ also in N^h_0.

The property *Strong BNDC* (*SBNDC*, for short) has been introduced in [4] as a sufficient condition for verifying *BNDC*. It just requires that before and after every high step, the system appears to be the same, from a low level perspective. It is stronger than *P_BNDC* and it can be defined through unwinding conditions as follows (see [2]).

$$E \in SBNDC \iff E \in \mathcal{W}(\approx^l_B, \equiv)$$

where $\equiv$ is the identity relation on processes.

If a refinement preserves the low level observation of a process, it preserves the *SBNDC* security property.

Corollary 4.9 Let $F \in \mathcal{E}$ be a *SBNDC* process. If $\mathcal{R}$ is a refinement preserving $\approx^l_B$ and $\mathcal{R}(F)\downarrow$, then $\mathcal{R}(F)$ is *SBNDC*.

Both *P_BNDC* and *SBNDC* are compositional with respect to almost all the SPA operators but they are not compositional with respect to the nondeterministic choice operator. This is not much surprising since security properties are, in general, not preserved under composition [16]. However, compositionality results are crucial for making the development of large and complex systems feasible [17, 18, 15]. The interest in the class of *Compositional P_BNDC* processes (*CP_BNDC*, for short) derives from the fact that it is fully compositional (i.e., it is compositional also with respect to the nondeterministic choice). *CP_BNDC* can be defined as follows (see [2]):

$$E \in CP\_BNDC \iff E \in \mathcal{W}(\approx^l_B, \stackrel{\tau}{\Longrightarrow})$$

Corollary 4.10 Let $F \in \mathcal{E}$ be a *CP_BNDC* process. If $\mathcal{R}$ is a refinement preserving both $\approx^l_B$ and $\stackrel{\tau}{\Longrightarrow}$ and $\mathcal{R}(F)\downarrow$, then $\mathcal{R}(F)$ is *CP_BNDC*.

Example 4.11 As mentioned above, the *CP_BNDC* property has been proposed to obtain full compositionality with respect to SPA operators. We show this feature by considering a non-deterministic composition of the high and low memory cells of Example 3.5. In particular we consider memory cell M^{hl}_0 that, after the first computational step, behaves as either M^h_0 or M^l_0.

$$M^{hl}\_0 \stackrel{\text{def}}{=} M^h\_0 + M^l\_0$$

Intuitively, this process should be secure since we have proved that both M^h_0 and M^l_0 are secure but, quite surprisingly, this is not the case. Consider the execution of a high level write action w_h_0. This moves the whole M^{hl}_0 system to M^h_0 (notice that M^l_0 does not accept the high level input w_h_0). The problem is that a low level user can observe this move by trying to write some value into the memory cell. As a matter of fact, since M^h_0 does not accept low level inputs, the low level user can deduce that some high level action has been performed.

This indirect information flow can be exploited to build a so called *covert-channel* (see, e.g., [5] for more detail). Formally, the move $M^{hl}\_0 \stackrel{w_h\_0}{\rightarrow} M^h\_0$ cannot be simulated by M^{hl}_0 thus proving that M^{hl}_0 is neither *P_BNDC*, *SBNDC* nor *CP_BNDC*.

This problem can be corrected by making M^h_0 and M^l_0 *CP_BNDC*, since *CP_BNDC* is compositional with respect to $+$ operator. To correct the processes it is enough to add a τ-loop in the initial state, i.e., a $\tau \,.\, M^h\_0$ and a $\tau \,.\, M^l\_0$ branch in M^h_0 and M^l_0, respectively. It is easy to prove that these modified M^h_0 and M^l_0 are *CP_BNDC*. By compositionality results we thus obtain that

M^{hl}_0 is now *CP_BNDC*. Notice that the problem of simulating the move $M^{hl}\_0 \stackrel{w_h}{\rightarrow} M^h\_0$ is now solved by performing the τ of the added $\tau\ .\ M^h\_0$ branch. In particular we have that $M^{hl}\_0 \stackrel{\tau}{\rightarrow} M^h\_0$.

Finally, by Corollary 4.10, we obtain that every refinement of these new M^h_0 and M^l_0, that preserves $\approx^l_B$, as already discussed in Example 4.8, and that also preserves the τ-loops, always gives *CP_BNDC* processes.

4.2.2 Traces based Security Properties

Equivalence under bisimulation is too demanding when security properties related to protocol analysis are considered [6]. Actually, most of the security properties that have been proposed for the analysis of security protocols are based on the notion of trace equivalence: two processes are equivalent if they exactly show the same execution sequences (called traces). In this section we recall the definition of two information flow security properties, *NDC* ([4]) and *PSP* ([23]), defined in terms of the set of traces of processes. We introduce the security property *SNDC* by means of an unwinding condition. Similarly to *SBNDC* for *BNDC* it provides a sufficient condition for *NDC*. We show how to preserve *NDC* under refinement. Finally, we derive a sufficient condition for *PSP* and provide a condition to preserve it under refinement.

The *trace equivalence* relation, denoted by $\approx_T$, equates two processes if they have the same sets of traces without considering the τ actions, where the set of traces associated with a process E is $Tr(E) = \{t \in \mathcal{L}^* \mid \exists E' : E \stackrel{t}{\Longrightarrow} E'\}$. Trace equivalence is less demanding than weak bisimulation since it does not require a step by step mutual simulation. For example $a.0$ and $a.0 + \tau.0$ are trace equivalent (the only trace is a) but they are neither weakly nor strongly bisimilar. The first process cannot simulate the deadlock state reached through the internal τ action. It is instead easy to see that if two processes are weakly bisimilar, then they are also trace equivalent.

The *Non Deducibility on Composition* (*NDC*, for short) property has been introduced in [4] as the property corresponding to *BNDC* when trace equivalence is used instead of bisimulation. It has been proved to be equivalent to *noninterference (NF)* defined in [20]. In particular, the notion of low observation of a process with respect to trace equivalence is the following.

Definition 4.12 (Trace Equivalence on Low Actions) For any process $E \in \mathcal{E}$, we denote by $Tr^l(E)$ the set of low traces associated with E which is defined as follows: $Tr^l(E) = \{t \in L^* \mid \exists E' : E \stackrel{t}{\Longrightarrow} E'\}$. Two processes E and F are *trace equivalent on low actions*, denoted by $E \approx^l_T F$, if $Tr^l(E) = Tr^l(F)$.

Definition 4.13 (NDC) Let $E \in \mathcal{E}$. $E \in NDC$ if $\forall\ \Pi \in \mathcal{E}_H$, $E \approx^l_T (E|\Pi)$.

By instantiating our generalized unwinding condition with the trace equivalence on low actions and the identity, we obtain the following property: *Strong NDC* (*SNDC*, for short). It provides a sufficient condition for *NDC*.

$$SNDC = \mathcal{W}(\approx^l_T, \equiv)$$

Proposition 4.14 Let $E \in \mathcal{E}$. If E is *SNDC* then E is *NDC*.

As for the other properties it is easy to identify a sufficient condition to be satisfied by our refinement operators in order to preserve the persistent traces based security property *SNDC*.

Corollary 4.15 Let $F \in \mathcal{E}$ be a *SNDC* process. If $\mathcal{R}$ be a refinement preserving $\approx^l_T$, then $\mathcal{R}(F)$ is a *SNDC* process.

Example 4.16 Let $v \in \mathcal{L}$ and $\mathcal{R}_{\setminus v}$ be the refinement such that $\mathcal{R}_{\setminus v}(E) = E \setminus v$. Then $\mathcal{R}$ is a refinement preserving the low observations for trace equivalence, i.e., $\approx^l_T$.

Now we turn our attention to the *Perfect Security Property* (*PSP*, for short) which has been proved [23] to be the weakest property to ensure no information can flow from high level users to low level users. In [14] this property is defined as follows.

Definition 4.17 (PSP) Let $E \in \mathcal{E}$. $E \in PSP$ if
- for all $t \in Tr(E)$ $t_{|L} \in Tr(E)$; and
- for all $\beta\alpha \in Tr(E)$ with $\alpha \in L^*$, and for all $h \in H$, if $\beta h \in Tr(E)$, then $\beta h\alpha \in Tr(E)$;

where $t_{|L}$ denotes the sequence obtained from t by deleting all the occurrences of high level actions.

We can always associate to any process E a deterministic process $det(E)$ having the same set of traces. Requiring the condition *SNDC* on $det(E)$ we get a sufficient condition for E being in *PSP*.

Proposition 4.18 Let $E \in \mathcal{E}$. If $det(E)$ is $SNDC$, then E is *PSP*.

Thus, by applying Corollary 4.15 we obtain a sufficient condition for the preservation of *PSP* under refinement.

Corollary 4.19 Let $F \in \mathcal{E}$ be a process satisfying *PSP* and such that $det(F) \in$ *SNDC*. Moreover, let $\mathcal{R}$ be a refinement preserving $\approx^l_T$ and commuting with det (i.e. such that $\mathcal{R}(det(F)) = det(R(F))$). Then $\mathcal{R}(F)$ satisfies *PSP*.

5. Conclusion and Related Works

In this paper we presented a new notion of refinement for processes described as terms in the Security Process Algebra (SPA). We proved some basic important properties like incrementality of the refinement process and compositionality with respect to SPA operators. Moreover, we showed how to check preservation under refinement for a variety of information flow properties proposed in literature.

Paper [14] is undoubtedly very related to our work. In such a paper, Mantel gives some conditions under which refinement preserves information flow properties. There are, however, appreciable differences with our work: First, we consider systems expressed in a process calculus, while Mantel consider event systems; more importantly, we focus our attention to bisimulation-based properties while Mantel only considers trace-based models; more specifically, in [14] it is assumed that processes have a deterministic transition system while here processes may show any nondeterministic behavior. Due to these differences, the notion of refinement we propose is quite different with respect to the one of [14], where refinement is simply formalized as trace inclusion. Our generalized unwinding condition is indeed a generalization of the one proposed in [14] and, in Section 4.2.2, we have shown that some of Mantel's results can be given as instances of our general refinement theorem.

Another interesting related work is [12], where Lowe gives a new non-interference notion for quantifying the amount of information-flow in a system described as a CSP process. The proposed notion is based on the NDC property considered here. Quite interestingly, Lowe observes that NDC is not closed under CSP refinement, and he solves this problem by requiring, for a system to be secure, that all of its refinements are secure (i.e., by closing the property under refinement). Here, we take the dual approach of imposing constraints on refinements, instead of strengthening the security properties. It would be interesting to see if the notion of information-flow of [12] can be revisited by considering only "secure" refinements, as done in this work.

Other important works on refining non-interference are [11, 21, 22, 9]. As observed in [14], in [11] it is given a method for making a specification secure after it has been sufficiently refined. This differs by our approach where we intend to prove non-interference guarantees at the beginning of the refinement process. In [21], instead, confidentiality statements are proved from scratch after refinement, i.e., there is no result of preservation of confidentiality. In [22], it is given a notion of non-interference that is preserved under CSP refinement. Differently from our approach, such a notion requires that processes are deterministic from a low level point of view. A detailed comparison between NDC-like properties and the notions proposed in [22] can be found in [3]. Paper [9], similarly to our work and [14], requires that refinement preserves some relations between processes. However, differently from what we do, only traces are considered.

References

[1] D. E. Bell and L. J. L. Padula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, MITRE MTR-2997, 1976.

[2] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Bisimulation and Unwinding for Verifying Possibilistic Security Properties. In L. D. Zuck, P. C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *Proc. of Int. Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'03)*, volume 2575 of *LNCS*, pages 223–237. Springer-Verlag, 2003.

[3] R. Focardi. Comparing Two Information Flow Security Properties. In *Proceedings of CSFW'96*, pages 116–122. IEEE press, June 1996.

[4] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1994/1995.

[5] R. Focardi and R. Gorrieri. Classification of Security Properties (Part I: Information Flow). In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*. Springer-Verlag, 2001.

[6] R. Focardi and F. Martinelli. A Uniform Approach for the Definition of Security Properties. In J. Wing, J. Woodcook, and J. Davies, editors, *Proc. of World Congress on Formal Methods (FM'99)*, volume 1708 of *LNCS*, pages 794–813. Springer-Verlag, 1999.

[7] R. Focardi and S. Rossi. Information Flow Security in Dynamic Contexts. In *Proc. of the 15th IEEE Computer Security Foundations Workshop*, pages 307–319. IEEE Computer Society Press, 2002.

[8] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.

[9] J. Graham-Cumming and J. W. Sanders. On the Refinement of Non-Interference. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 35–42. IEEE Computer Society Press, 1991.

[10] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *IEEE Symposium on Foundations of Computer Science (FOCS'95)*, pages 453–462, 1995.

[11] J. Jacob. On the Derivation of Secure Components. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 242–247. IEEE Computer Society Press, 1989.

[12] G. Lowe. Quantifying Information Flow. In *Proc. of the 15th IEEE Computer Security Foundations Workshop*, pages 18–31. IEEE Computer Society Press, 2002.

[13] H. Mantel. Possibilistic Definitions of Security - An Assebly Kit -. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 185–199. IEEE Computer Society Press, 2000.

[14] H. Mantel. Preserving Information Flow Properties under Refinement. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 78–91. IEEE Computer Society Press, 2001.

[15] H. Mantel. On the Composition of Secure Systems. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 88–101. IEEE Computer Society Press, 2002.

[16] D. McCullough. Specifications for Multi-Level Security and a Hook-Up Property. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 161–166. IEEE Computer Society Press, 1987.

[17] J. McLean. A General Theory of Composition for Trace Sets Closed under Selective Interleaving Functions. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 79–93. IEEE Computer Society Press, 1994.

[18] J. McLean. A General Theory of Composition for a Class of "Possibilistic" Security Properties. *IEEE Trabsactions on Software Engineering*, 22(1):53–67, 1996.

[19] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[20] C. O'Halloran. A Calculus of Information Flow. In *Proc. of the European Symposium on Research in Security and Privacy*, pages 180–187. AFCET, 1990.

[21] C. O'Halloran. Refinement and Confidentiality. In *Proc. of the 5th Refinement Workshop*, pages 119–139, 1992.

[22] A. W. Roscoe, J. C. P. Woodcock, and L. Wulf. Non-Interference through Determinism. In *Proc. of the European Symposium on Research in Computer Security*, volume 875 of *LNCS*, pages 33–53. Springer-Verlag, 1994.

[23] A. Zakinthinos and E. S. Lee. A General Theory of Security Properties. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 74–102. IEEE Computer Society Press, 1997.