In R.Focardi and R.Gorrieri (Eds.): *FOSAD 2000, LNCS 2171*, pp. 331-396, 2001.

Classification of Security Properties *
(Part I: Information Flow)

Riccardo Focardi[1] and Roberto Gorrieri[2]

[1] Dipartimento di Informatica, Università Ca'Foscari di Venezia
focardi@dsi.unive.it
[2] Dipartimento di Scienze dell'informazione, Università di Bologna
gorrieri@cs.unibo.it

Abstract. In the recent years, many formalizations of security properties have been proposed, most of which are based on different underlying models and are consequently difficult to compare. A classification of security properties is thus of interest for understanding the relationships among different definitions and for evaluating the relative merits. In this paper, many non-interference-like properties proposed for computer security are classified and compared in a unifying framework. The resulting taxonomy is evaluated through some case studies of access control in computer systems. The approach has been mechanized, resulting in the tool CoSeC. Various extensions (e.g., the application to cryptographic protocol analysis) and open problems are discussed.

This paper mainly follows [21] and covers the first part of the course "Classification of Security Properties" given by Roberto Gorrieri and Riccardo Focardi at FOSAD'00 school.

1 Introduction

The wide spread of distributed systems, where resources and data are shared among users located almost everywhere in the world, has enormously increased the interest in security issues. In this context, it is likely that a user gets some (possibly) malicious programs from an untrusted source on the net and executes them inside its own system with unpredictable results. Moreover, it could be the case that a system completely secure inside, results to be insecure when performing critical activities such as electronic commerce or home banking, due to a "weak" mechanism for remote connections. It is important to precisely define security properties in order to have formal statements of the correctness of a security mechanism. As a consequence, in the recent years there have been a number of proposals of formal definitions of security properties (see, for instance, [1, 2, 8, 11, 12, 17, 21, 30, 44, 45, 51, 53, 59, 60]).

* This work has been partially supported by MURST projects TOSCA, "Certificazione automatica di programmi mediante interpretazione astratta" and "Interpretazione astratta, type systems e analisi control-flow", and also partially supported by Microsoft Research Europe.

In this paper we deal with a particular class of security properties, called *information flow properties*, which aim at controlling the way information may flow among different entities. They have been first proposed as a means to ensure confidentiality, in particular to verify if access control policies are sufficient to guarantee the secrecy of (possibly classified) information. Indeed, although access control is a well studied technique for system security, it is not trivial to find an access control policy which guarantees that no information leak is possible. One of the main problems is to limit, and possibly to avoid, damages produced by malicious programs, called *Trojan Horses*, which try to leak secret information.

For example, consider a classic *Discretionary Access Control* security (DAC for short), where every *subject* (i.e., an active agent such as a user), decides the access properties of its *objects* (i.e., passive agents such as files). The file management in Unix is a classic example of DAC. Indeed it allows users to decide the access rights on their files. This flexibility may facilitate security leakages. As an example, if a user A executes a Trojan Horse program, this can directly modify the access properties of A's objects making, e.g., all A's data visible to every other user. In this respect, we can say that the DAC approach gives no guarantees against "internal" attacks.

A different approach to this problem is the *Mandatory Access Control* (MAC for short), where some access rules are imposed by the system. Even if we have executed a Trojan Horse program, its action will be limited since it is not allowed to change MAC rules. An example of MAC is *Multilevel Security* [5]: every object is bound to a security level, and so is every subject; information can flow from a certain object to a certain subject only if the level of the subject is greater than the level of the object. So a Trojan Horse, which operates at a certain level, has in principle no way to *downgrade* information, and its action is restricted to that level. This policy can be implemented through two access rules: *No Read Up* (a subject cannot read data from an upper level object) and *No Write Down* (a subject cannot write data in a lower level object).

However, even if we adopt the less flexible MAC approach, this could be insufficient to stop the action of a Trojan Horse. Indeed, it could be possible to transmit information indirectly using system side effects. For example, if two levels – 'high' and 'low' – share some finite storage resource (e.g., a hard disk), it may be possible to transmit data from level 'high' to level 'low' by exploiting the 'resource full' error message. For a high level transmitter, it is sufficient to alternatively fill or empty the resource in order to transmit a '1' or a '0' datum. Simultaneously, the low level receiver tries to write on the resource, decoding every error message as a '1' and every successful write as a '0'. It is clear that such indirect transmissions, called *covert channels*, do not violate the two multilevel access rules (see Figure 1). Therefore it is often necessary to integrate a MAC discipline with a covert channel analysis (see, e.g., [63]).

The existence of covert channels has led to the more general approach of information flow security, mentioned above. The idea is to try to directly control the whole flow of information, rather than the accesses of subjects to objects [36]. By imposing some information flow rules, it is possible to indifferently control

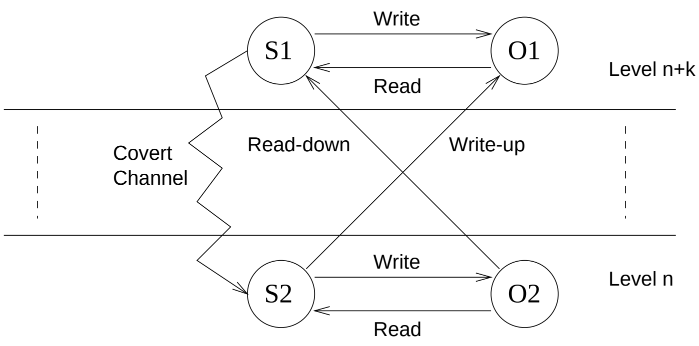


Fig. 1. Information flows in multilevel security.

direct and indirect leakages, as, in this perspective, they both become "unwanted information flows".

In the literature, there are many different security definitions reminiscent of the information flow idea, each based on some system model (see, e.g., [36, 64, 41, 48, 49, 62, 7]). In [24] we have compared and classified them, leading to our proposed notion of *Bisimulation Non Deducibility on Compositions* (*BNDC*, for short). We will present *BNDC* starting by the idea of Non Interference [36]. Through a running example and a comparison with other existing approaches, we will try to convince the reader that such a property can effectively detect unwanted information flows in systems, both direct and indirect.

We now describe the topics of the next sections. Section 2 presents the *Security Process Algebra* (SPA, for short) language. All the properties we will present and apply to the analysis of systems and protocols are based on such a language. SPA is an extension of CCS [50] – a language proposed to specify concurrent systems. The basic building blocks are the atomic activities, simply called *actions*; unlike CCS, in SPA actions belong to two different levels of confidentiality, thus allowing the specification of multilevel (actually, two-level) systems. As for CCS, the model used to describe the operational semantics of SPA is the *labelled transition system* model [43], where the states are the terms of the algebra. In order to express that certain states are indistinguishable for an external observer, semantic equivalences over terms/states are defined such that two terms are observationally indistinguishable iff they are equivalent. As explained below, the information flow security properties we introduce are all based on these notions of observable behaviours.

Section 3 is about such properties, that capture the existence of information flows among groups of users. We will see that these properties are all of the following algebraic form. Let E be an SPA process term, let X be a security property, let $\approx$ be a semantic equivalence among process terms and let $\mathcal{C}_X$ and $\mathcal{D}_X$ be two SPA contexts[1] for property X. Then, we can say:

$$E \text{ is } X\text{-secure if and only if } \mathcal{C}_X[E] \approx \mathcal{D}_X[E].$$

where the contexts $\mathcal{C}_X$ and $\mathcal{D}_X$ are such that only (part of) the low behaviour of E becomes observable; hence, the behavioural equivalence compares these, possibly different, low behaviours of E.

A first obvious consequence is that the security properties become parametric w.r.t. the chosen notion of equivalence: if an equivalence $\approx_1$ is finer than $\approx_2$ then each security property based on $\approx_1$ is satisfied by a subset of the processes satisfying the corresponding security property based on $\approx_2$. A second, less obvious consequence is that such information flow properties are not safety properties, i.e. properties that can be specified as sets of acceptable behaviours. Indeed, by defining a property of this form for E as an equivalence problem – $\mathcal{C}_X[E] \approx \mathcal{D}_X[E]$ – on suitable contexts, we are actually stating that if some behaviour can occur, then it must be the case that also some other related behaviour must be possible; such a property cannot be expressed simply as a set of acceptable behaviour.

We analyze which kinds of flows are detectable by the various properties through the running example of an access monitor. In particular, we try to show that certain properties are not appropriate to deal with some kinds of information flows and so it is necessary to strengthen them by choosing a finer equivalence notion or, if this is not enough, by following a different approach.

In Section 4 we present a tool called *Compositional Security Checker* (CoSeC, for short) which can be used to check automatically (finite state) SPA specifications against some information flow security properties. We exploit the algebraic definition style. Indeed, checking the X-security of E is reduced to the "standard" problem of checking semantic equivalence between two terms having E as a sub-term. The CoSeC tool has the same modular architecture as Concurrency Workbench (CW for short) [14], from which some modules have been imported, and others modified. The tool is equipped with a parser, which transforms an SPA specification into a parse-tree; then, for the parsed specification CoSeC builds the labelled transition system following the operational rules defined in Plotkin' SOS style [54]. When a user wants to check if an SPA process E is X-secure, CoSeC first provides operational semantic descriptions to the terms $\mathcal{C}_X[E]$ and $\mathcal{D}_X[E]$ in the form of two LTSs; then verifies the semantic equivalence of $\mathcal{C}_X[E]$ and $\mathcal{D}_X[E]$ using their LTS representations. An interesting feature of CoSeC is the exploitation of the compositionality of some security properties in order to avoid, in some cases, the exponential state explosion due to the parallel composition operator.

[1] An SPA context $\mathcal{G}$ is an SPA term "with a hole". E.g., $\mathcal{G}[-] = F + -$. The insertion of E in the context $\mathcal{G}$, written as $\mathcal{G}[E]$, has the effect of filling the hole with E. In the example, $\mathcal{G}[E] = F + E$. Subscript X simply means that $\mathcal{C}_X$ and $\mathcal{D}_X$ are two particular contexts for property X, i.e., it is used to give a name to contexts for property X.

Finally, in Section 5 we give some concluding remarks and discuss some open problems.

2 SPA and Value-Passing

In this Section we present the language that will be used to specify and analyze security properties over concurrent systems. We first present the "pure" version of the language. Then we show how to extend it with value-passing. Finally, we present an example of value-passing agent specification. It will be our running example for the next sections.

2.1 The Language

The *Security Process Algebra* (SPA for short) [24, 26] is a slight extension of Milner's CCS [50], where the set of visible actions is partitioned into high level actions and low level ones in order to specify multilevel systems. [2]

SPA syntax is based on the same elements as CCS. In order to obtain a partition of the visible actions into two levels, we consider two sets Act_H and Act_L of high and low level actions which are closed with respect to function $\bar{\cdot}$ (i.e., $\overline{Act_H} = Act_H$, $\overline{Act_L} = Act_L$); moreover they form a covering of $\mathcal{L}$ and they are disjoint (i.e., $Act_H \cup Act_L = \mathcal{L}$, $Act_H \cap Act_L = \emptyset$). Let Act be the set $Act_H \cup Act_L \cup \{\tau\}$, where τ is a special unobservable, internal action. The syntax of SPA *agents* (or *processes*) is defined as follows:

$$E ::= \underline{0} \;\big|\; \mu.E \;\big|\; E + E \;\big|\; E|E \;\big|\; E \setminus\!\setminus L \;\big|\; E[f] \;\big|\; Z$$

where μ ranges over Act, $L \subseteq \mathcal{L}$ and $f : Act \to Act$ is such that $f(\bar{\alpha}) = \overline{f(\alpha)}$, $f(\tau) = \tau$. Moreover, for every constant Z there must be the corresponding definition: $Z \stackrel{\text{def}}{=} E$, and E must be *guarded* on constants. This means that the recursive substitution of all the non prefixed (i.e., not appearing in a context $\mu.E'$) constants in E with their definitions terminates after a finite number of steps. In other words, there exists a term obtainable by constant substitutions from E where all the possible initial actions are explicitly represented (through the prefix operator $\mu.E$). For instance, agent $A \stackrel{\text{def}}{=} B$ with $B \stackrel{\text{def}}{=} A$ is not guarded on constants. On the contrary, if B is defined as $a.A$, then B is guarded on constants. This condition will be useful when we will do automatic checks over SPA terms. As a matter of fact, it basically avoids infinite constant substitution loops.

Intuitively, we have that $\underline{0}$ is the empty process, which cannot do any action; $\mu.E$ can do an action μ and then behaves like E; $E_1 + E_2$ can alternatively

[2] Actually, only two-level systems can be specified; note that this is not a real limitation because it is always possible to deal with the multilevel case by grouping – in several ways – the various levels in two clusters.

choose [3] to behave like E_1 or E_2 ; $E_1|E_2$ is the parallel composition of E_1 and E_2, where the executions of the two systems are interleaved, possibly synchronized on complementary input/output actions, producing an internal τ; $E \setminus\!\setminus L$ can execute all the actions E is able to do, provided that they do not belong to L; if E can execute action μ, then $E[f]$ performs $f(\mu)$.

The only other difference from CCS is the restriction operator $\setminus\!\setminus$; as we said above, $E \setminus\!\setminus L$ can execute all the actions E is able to do, provided that they do not belong to L. In CCS the corresponding operator $\setminus$ requires that the actions do not belong to $L \cup \bar{L}$. We will show in a moment that it is easy to define the standard restriction operator of CCS using this new restriction. The reason why we introduce this slight modification is that it is necessary to define an additional input restriction operator which will be useful in characterizing some security properties in an algebraic style. After that we will have no further use for the $\setminus\!\setminus$ operator. We define the CCS restriction and the input restriction operators as follows:

$$E \setminus L \stackrel{\text{def}}{=} E \setminus\!\setminus L \cup \bar{L}$$
$$E \setminus_I L \stackrel{\text{def}}{=} E \setminus\!\setminus L \cap I$$

$E \setminus L$ is the CCS restriction operator, while $E \setminus_I L$ requires that the actions of E do not belong to $L \cap I$

For the definition of security properties we also need the hiding operator of CSP [39] which can be defined as a relabelling as follows:

$$E/L \stackrel{\text{def}}{=} E[f_L] \text{ where } f_L(x) = \begin{cases} x \text{ if } x \notin L \\ \tau \text{ if } x \in L \end{cases}$$

E/L turns all the actions in L into internal τ's.

Let $\mathcal{E}$ be the set of SPA agents, ranged over by E and F. Let $\mathcal{L}(E)$ denote the *sort* of E, i.e., the set of the (possibly executable) actions occurring syntactically in E. The sets of high level agents and low level ones are defined as $\mathcal{E}_H \stackrel{\text{def}}{=} \{E \in \mathcal{E} \mid \mathcal{L}(E) \subseteq Act_H \cup \{\tau\}\}$ and $\mathcal{E}_L \stackrel{\text{def}}{=} \{E \in \mathcal{E} \mid \mathcal{L}(E) \subseteq Act_L \cup \{\tau\}\}$, respectively. ¿From a security point of view, processes in $\mathcal{E}_H$ and in $\mathcal{E}_L$ are secure by isolation, as all the actions they perform are bound to one particular level (high or low, respectively). More interesting is the case of processes in $\mathcal{E}_H \cup \mathcal{E}_L \subset \mathcal{E}$, i.e., of those processes that execute both high level and low level actions, allowing for communications between the two levels, hence possibly introducing unwanted information flows.

2.2 Operational Semantics and Equivalences

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

[3] For notational convenience, we use sometimes the $\sum$ operator (indexed on a set) to represent a general n-ary (or even infinitary) sum operator.

Prefix	$$\frac{-}{\mu.E \xrightarrow{\mu} E}$$
Sum	$$\frac{E_1 \xrightarrow{\mu} E_1'}{E_1 + E_2 \xrightarrow{\mu} E_1'} \qquad \frac{E_2 \xrightarrow{\mu} E_2'}{E_1 + E_2 \xrightarrow{\mu} E_2'}$$
Parallel	$$\frac{E_1 \xrightarrow{\mu} E_1'}{E_1 \vert E_2 \xrightarrow{\mu} E_1' \vert E_2} \qquad \frac{E_2 \xrightarrow{\mu} E_2'}{E_1 \vert E_2 \xrightarrow{\mu} E_1 \vert E_2'} \qquad \frac{E_1 \xrightarrow{\alpha} E_1' \quad E_2 \xrightarrow{\bar{\alpha}} E_2'}{E_1 \vert E_2 \xrightarrow{\tau} E_1' \vert E_2'}$$
Restriction	$$\frac{E \xrightarrow{\mu} E'}{E \setminus\setminus L \xrightarrow{\mu} E' \setminus\setminus L} \ \text{ if } \mu \notin L$$
Relabelling	$$\frac{E \xrightarrow{\mu} E'}{E[f] \xrightarrow{f(\mu)} E'[f]}$$
Constant	$$\frac{E \xrightarrow{\mu} E'}{A \xrightarrow{\mu} E'} \ \text{ if } A \stackrel{\text{def}}{=} E$$

Table 1. The operational rules for SPA.

as for CCS by structural induction as the least relation generated by the axioms and inference rules reported in Table 1. The operational semantics for an agent E is the subpart of the SPA LTS reachable from the initial state E. We denote with $\mathcal{E}_{FS}$ the set of all the SPA agents with a finite LTS as operational semantics. Table 2 shows some simple examples of SPA terms with their corresponding LTSs.

Now we introduce the idea of observable behaviour: two systems should have the same semantics if and only if they cannot be distinguished by an external observer. To obtain this we define an equivalence relation over states/terms of the SPA LTS, equating two processes when they are indistinguishable. In this way the semantics of a term becomes an equivalence class of terms.

It is possible to define various equivalences of this kind, according to the different assumptions on the power of observers. We recall three of them. The first one is the classic definition of *trace equivalence*, according to which two agents are equivalent if they have the same execution traces. The second one discriminates agents also according to the nondeterministic structure of their LTSs. This equivalence is based on the concept of *bisimulation* [50]. The last one, introduced for the CSP language [39], is able to observe which actions are not executable after a certain trace (*failure* sets), thus detecting possible deadlocks.

Since we want to focus only on observable actions, we need a transition relation which does not take care of internal τ moves. This can be defined as follows:

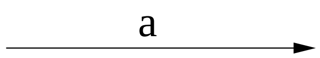
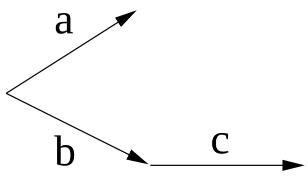
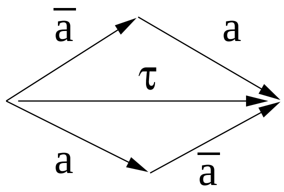
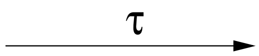
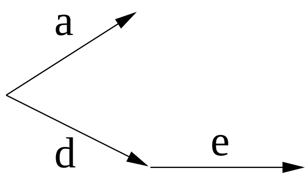
Prefix:	$E_1 \stackrel{\text{def}}{=} a.\underline{0}$	
Sum:	$E_2 \stackrel{\text{def}}{=} a.\underline{0} + b.c.\underline{0}$	
Parallel:	$E_3 \stackrel{\text{def}}{=} a.\underline{0} \mid \bar{a}.\underline{0}$	
Restriction:	$E_4 \stackrel{\text{def}}{=} E_3 \setminus \{a\}$	
Relabelling:	$E_5 \stackrel{\text{def}}{=} E_2[b/d, c/e]$	

Table 2. Some simple SPA terms.

Definition 1. *The expression* $E \stackrel{\alpha}{\Longrightarrow} E'$ *is a shorthand for* $E(\stackrel{\tau}{\rightarrow})^* E_1 \stackrel{\alpha}{\rightarrow} E_2(\stackrel{\tau}{\rightarrow})^* E'$, *where* $(\stackrel{\tau}{\rightarrow})^*$ *denotes a (possibly empty) sequence of* τ *labelled transitions. Let* $\gamma = \alpha_1 \dots \alpha_n \in \mathcal{L}^*$ *be a sequence of actions; then* $E \stackrel{\gamma}{\Longrightarrow} E'$ *if and only if there exist* $E_1, E_2, \dots, E_{n-1} \in \mathcal{E}$ *such that* $E \stackrel{\alpha_1}{\Longrightarrow} E_1 \stackrel{\alpha_2}{\Longrightarrow} \cdots \stackrel{\alpha_{n-1}}{\Longrightarrow} E_{n-1} \stackrel{\alpha_n}{\Longrightarrow} E'$. *For the empty sequence* $\langle\rangle$ *we have that* $E \stackrel{\langle\rangle}{\Longrightarrow} E'$ *stands for* $E(\stackrel{\tau}{\rightarrow})^* E'$. *We say that* E' *is* reachable *from* E *when* $\exists \gamma : E \stackrel{\gamma}{\Longrightarrow} E'$ *and we write* $E \Longrightarrow E'$. ■

Trace Equivalence We define trace equivalence as follows:

Definition 2. *For any* $E \in \mathcal{E}$ *the set* $T(E)$ *of* traces associated with E *is defined as follows:* $T(E) = \{\gamma \in \mathcal{L}^* \mid \exists E' : E \stackrel{\gamma}{\Longrightarrow} E'\}$. E *and* F *are* trace equivalent *(notation* $E \approx_T F$*) if and only if* $T(E) = T(F)$. ■

Observational Equivalence Bisimulation is based on an idea of mutual step-by-step simulation, i.e., when E executes a certain action moving to E', F must be able to simulate this single step by executing the same action and moving to an agent F' which is again bisimilar to E' (this is because it must be able to simulate every successive step of E'), and vice-versa.

A weak bisimulation is a bisimulation which does not care about internal τ actions. So, when F simulates an action of E, it can also execute some τ actions before or after that action.

In the following, $E \stackrel{\hat{\mu}}{\Longrightarrow} E'$ stands for $E \stackrel{\mu}{\Longrightarrow} E'$ if $\mu \in \mathcal{L}$, and for $E\ (\stackrel{\tau}{\rightarrow})^*\ E'$ if $\mu = \tau$ (note that $(\stackrel{\tau}{\rightarrow})^*$ means "zero or more τ labelled transitions" while $\stackrel{\tau}{\Longrightarrow}$ requires at least one τ labelled transition).

Definition 3. *A relation* $R \subseteq \mathcal{E} \times \mathcal{E}$ *is a* weak bisimulation *if* $(E, F) \in R$ *implies, for all* $\mu \in Act$,

- *whenever* $E \stackrel{\mu}{\longrightarrow} E'$, *then there exists* $F' \in \mathcal{E}$ *such that* $F \stackrel{\hat{\mu}}{\Longrightarrow} F'$ *and* $(E', F') \in R$;
- *conversely, whenever* $F \stackrel{\mu}{\longrightarrow} F'$, *then there exists* $E' \in \mathcal{E}$ *such that* $E \stackrel{\hat{\mu}}{\Longrightarrow} E'$ *and* $(E', F') \in R$.

Two SPA agents $E, F \in \mathcal{E}$ *are* observationally equivalent*, notation* $E \approx_B F$*, if there exists a weak bisimulation containing the pair* (E, F). ■

In [50] it is proved that $\approx_B$ is an equivalence relation. Moreover, it is easy to see that $E \approx_B F$ implies $E \approx_T F$; indeed, if $E \approx_B F$ then F must be able to simulate every sequence of visible actions executed by E, i.e., every trace of E; since the simulation corresponds to the execution of the actions interleaved with some τ's, then every trace of E is also a trace for F. Symmetrically, E must be able to simulate every sequence of F. So $E \approx_T F$.

In Figure 2 there is an example of two trace-equivalent systems which are not observationally equivalent. In fact both E and F can execute the three traces a, ab and ac. However, it is not possible for E to simulate step-by-step process F.

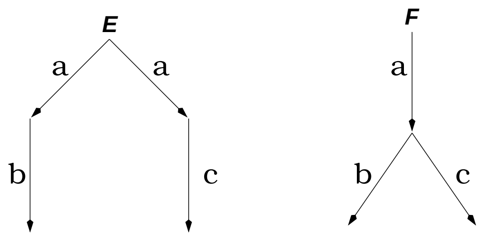


Fig. 2. Systems not observationally equivalent

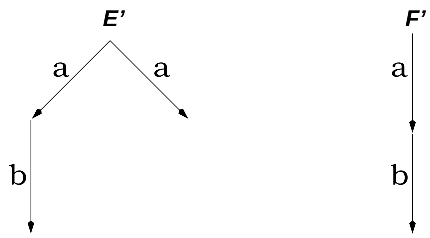


Fig. 3. Observational equivalence detects deadlocks.

In particular, F executes a and moves to a state where it can execute both b and c. Process E can simulate this first step of F but, after this, it cannot simulate F anymore since it is in a state where it can execute only b or c.

This ability of observing the branching structures of systems makes $\approx_B$ able to detect potential deadlocks. If we consider $E' \stackrel{\text{def}}{=} E \setminus \{c\}$ and $F' \stackrel{\text{def}}{=} F \setminus \{c\}$ (see Figure 3) we have that E' and F' are still trace equivalent but not observationally equivalent. Indeed, system E' can reach a deadlock state after the execution of action a. On the other hand, F' is not able to simulate this move since after a it can always execute b.

We conclude this section about bisimulation by introducing the notion of weak bisimulation up to $\approx_B$. It will be very useful when proving that two agents are observation equivalent. Indeed, we see that in order to ensure $P \approx_B Q$ it is sufficient that (P, Q) is in some weak bisimulation up to $\approx_B$.

Definition 4. *A relation* $S \subseteq \mathcal{E} \times \mathcal{E}$ *is a* weak bisimulation up to $\approx_B$ *if* $(E, F) \in S$ *implies, for all* $\mu \in Act$*,*

- *whenever* $E \stackrel{\mu}{\Longrightarrow} E'$*, then there exists* $F' \in \mathcal{E}$ *such that* $F \stackrel{\hat{\mu}}{\Longrightarrow} F'$ *and* $E' \approx_B S \approx_B F'$*;*
- *conversely, whenever* $F \stackrel{\mu}{\Longrightarrow} F'$*, then there exists* $E' \in \mathcal{E}$ *such that* $E \stackrel{\hat{\mu}}{\Longrightarrow} E'$ *and* $E' \approx_B S \approx_B F'$*.*

where $\approx_B S \approx_B$ is the composition of binary relations, so that $E' \approx_B S \approx_B F'$ means that for some E'' and F'' we have $E' \approx_B E''$, $(E'', F'') \in S$, $F' \approx_B F''$. ■

In [50] it is proven the following result:

Proposition 1. *If S is a bisimulation up to $\approx_B$ then $S \subseteq \approx_B$.*

Hence, to prove $P \approx_B Q$, we only have to find a bisimulation up to $\approx_B$ which contains (P, Q). This is one of the proof techniques we will often adopt in the following.

Failure/Testing Equivalence The failure semantics [9], introduced for the CSP language, is a refinement of the trace semantics where it is possible to observe which actions are not executable after a certain trace. In particular, a system is characterized by the so-called failures set, i.e., a set of pairs (s, X) where s is a trace and X is a set of actions. For each pair (s, X), the system must be able, by executing trace s, to reach a state where every action in X cannot be executed.[4] For instance, consider again agents E' and F' of Figure 3. As we said above, E' can stop after the execution of a and, consequently, E' can refuse to execute action b after the execution of a. So E' has the pair $(a, \{b\})$ in its failure set. System F' is always able to execute b after the execution of a. So F' does not have $(a, \{b\})$ in the failure set, hence it is not failure equivalent to E'. We deduce that also failure semantics is able to detect deadlocks. Moreover, also systems E and F of Figure 2 are not failure equivalent.

A different characterization of failure equivalence (called *testing equivalence*) has been given in [52]. It is based on the idea of tests. We can see a test T as any SPA process which can execute a particular *success* action $\omega \notin \mathcal{L}$. A test T is applied to a system E using the parallel composition operator $|$. A test T *may be satisfied* by system E if and only if system $(E|T) \setminus \mathcal{L}$ can execute ω. Note that in system $(E|T) \setminus \mathcal{L}$ we force the synchronization of E with test T.

Definition 5. *E may T if and only if $(E|T) \setminus \mathcal{L} \stackrel{\omega}{\Longrightarrow} (E'|T') \setminus \mathcal{L}$* ■

A maximal computation of $(E|T) \setminus \mathcal{L}$ is a sequence $(E|T) \setminus \mathcal{L} = ET_0 \stackrel{\tau}{\rightarrow} ET_1 \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} ET_n \stackrel{\tau}{\rightarrow} \ldots$ which can be finite or infinite; if it is finite the last term must have no outgoing transitions. A test T *must be satisfied* by E if and only if every maximal computation of $(E|T) \setminus \mathcal{L}$ contains a state ET_i which can execute ω.

Definition 6. *E must T if and only if for all maximal computations of $ET_0 = (E|T) \setminus \mathcal{L}, \exists i$ such that $ET_i \stackrel{\omega}{\rightarrow} ET_i'$* ■

Now we can define testing equivalence as follows:

Definition 7. *Two systems E and F are testing equivalent, $E \approx_{test} F$, if and only if*

[4] Indeed, there is a condition on the traces s in pairs (s, X). It is required that during any execution of s no infinite internal computation sequences are possible. We will analyze this aspect more in detail in the following.

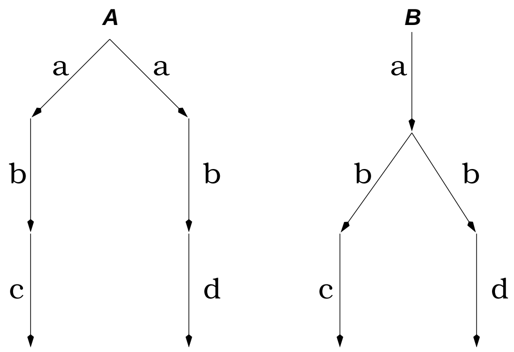


Fig. 4. Systems testing equivalent but not observationally equivalent.

i) E *may* $T \Leftrightarrow F$ *may* T
ii) E *must* $T \Leftrightarrow F$ *must* T

for every test T. ■

It is easy to see that the first condition holds if and only if $E \approx_T F$. In fact, if E may satisfy T, then E is able to execute the trace which moves T to the state where it can execute ω. [5] It is not difficult to see that the second condition corresponds to failure equivalence. The basic idea is that if E fails to execute a certain action a after a trace γ, we can detect this with a test T which executes the complementary actions of γ followed by $\bar{a}$, and then executes ω. In fact, for that T we have that $E\ \cancel{\mathit{must}}\ T$.

We have seen that both $\approx_B$ and $\approx_{test}$ are stronger than $\approx_T$, since they basically observe something of the branching structure of the LTSs. However we have said nothing about the relation between these two equivalences. The reason is that they are incomparable, i.e., no one of them implies the other. Indeed, we have that $\approx_B$ is in most cases more sensitive than $\approx_{test}$ to the branching structure of the agents. We can see this in the simple example of Figure 4. It shows that $\approx_B$ does not permit to "shift" the non-deterministic choice even if the first actions executed after the branch are exactly the same (b, in this case).

On the other hand $\approx_{test}$ is more discriminating with respect to divergent behaviour, i.e., infinite sequences of internal τ actions. Indeed, a divergence may "ruin" a test by generating unexpected maximal computations. As an example, see Figure 5 where the divergence after a in process B' determines that $B'\ \cancel{\mathit{must}}\ \bar{a}.\bar{b}.\omega.\underline{0}$ while A' *must* $\bar{a}.\bar{b}.\omega.\underline{0}$. This happens because, after the execution of a, the process could diverge never reaching the ω success action. This

[5] We could have more than one trace or more than one ω. However if a system may satisfy a test T, then it may satisfy also a test T' with only one ω and only one trace to it. T' corresponds to one of the traces executed by the system in order to satisfy the test T.

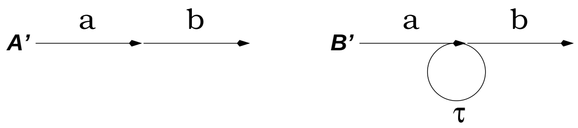


Fig. 5. Systems observationally equivalent but not testing equivalent.

aspect will be analyzed more in detail in the next chapters, when we will try to use $\approx_{test}$ in the specification of security properties.

We conclude this section presenting a class of finite state agents. This will be useful in the automatic verification of security properties. This class of agents consists of the so-called *nets of automata*:

$$
\begin{aligned}
p &::= \underline{0} \;\big|\; \mu.p \;\big|\; p+p \;\big|\; Z \\
E &::= p \;\big|\; E|E \;\big|\; E\setminus L \;\big|\; E\setminus_I L \;\big|\; E/L \;\big|\; E[f]
\end{aligned}
$$

where for every constant Z there must be the corresponding definition $Z \stackrel{\text{def}}{=} p$ (with p guarded on constants). It is easy to prove that every agent of this form is finite state. However, this condition is not necessary, in the sense that other agents not belonging to the class of nets of automata are finite state as well. For instance, consider $B \stackrel{\text{def}}{=} a.\underline{0} + D \setminus \{i\}$ with $D \stackrel{\text{def}}{=} i.(o.\underline{0}|D)$. It can execute only an action a and then it stops, so it is clearly finite state. However note that it does not conform to the syntax of nets of automata since there is a parallel operator underneath a sum.

2.3 Value-Passing SPA

In this section we briefly present a value-passing extension of "pure" SPA (VSPA, for short). All the examples contained in this paper will use this value passing calculus, because it leads to more readable specifications than those written in pure SPA. Here we present a very simple example of a value-passing agent showing how it can be translated into a pure SPA agent. Then we define the VSPA syntax and we sketch the semantics by translating a generic VSPA agent into its corresponding SPA agent.

As an example, consider the following buffer cell [50]:

$$
\begin{aligned}
C &\stackrel{\text{def}}{=} in(x).C'(x) \\
C'(x) &\stackrel{\text{def}}{=} \overline{out}(x).C
\end{aligned}
$$

where x is a variable that can assume values in $\mathbf{N}$ (we usually write $x \in \mathbf{N}$). C reads a natural number n through action in and stores it in variable x. Then this value is passed to agent C' which can give n as output through action out

moving again to C. So C represents a buffer which may hold a single data item. If we assume that in is a low level action and out a high level action, then C is a system exhibiting a legitimate direct information flow from low to high. On the contrary, if in is high and out is low, then C is an insecure system, downgrading information from high to low.

Now we show how C can be translated into an SPA agent. The parametrized constant C' becomes a family of constants C'_v one for each value $v \in \mathbf{N}$. Similarly $\overline{out}(x)$ becomes a family $\overline{out}_v$ of prefixes. So the single definition for C' becomes the following family of definitions:

$$C'_v \stackrel{\mathrm{def}}{=} \overline{out}_v.C \qquad (v \in \mathbf{N})$$

Now consider the prefix $in(x)$. To reflect the fact that it can accept any input value, binding variable x, we translate it into $\sum_{v \in \mathbf{N}} in_v$. So the definition becomes:

$$C \stackrel{\mathrm{def}}{=} \sum_{v \in \mathbf{N}} in_v.C'_v$$

VSPA is very similar to the value-passing CCS introduced in [50]. The main difference is that in VSPA we can have more than one parameter for actions and parameters are multi-sorted.

The syntax of VSPA agents is defined as follows:

$$\begin{aligned} E ::= \underline{0} \;\big|\; a(x_1, \ldots, x_n).E \;\big|\; \bar{a}(e_1, \ldots, e_n).E \;\big|\; \tau.E \;\big|\; E + E \;\big|\; E|E \;\big| \\ \big|\; E \setminus L \;\big|\; E \setminus_I L \;\big|\; E/L \;\big|\; E[f] \;\big|\; A(e'_1, \ldots, e'_n) \;\big| \\ \big|\; \mathbf{if}\ b\ \mathbf{then}\ E \;\big|\; \mathbf{if}\ b\ \mathbf{then}\ E\ \mathbf{else}\ E \end{aligned}$$

where the variables $x_1, \ldots, x_n$, the value expressions $e_1, \ldots, e_n$ and $e'_1, \ldots, e'_n$ must be consistent with the arity of the action a and constant A, respectively (the arity specifies the sorts of the parameters), and b is a boolean expression. The arity of actions and constants is given by function ari. This function returns a tuple of sets (called $Sorts$) that represent the ranges of the parameters for the specific action or constant considered. For example, $ari(a) = (S_1, \ldots, S_n)$ means that action a has n parameters with ranges $S_1, \ldots, S_n$, respectively.

It is also necessary to define constants as follows: $A(x_1, \ldots, x_m) \stackrel{\mathrm{def}}{=} E$ where E is a VSPA agent which may contain no free variables except $x_1, \ldots, x_m$, which must be distinct. As in [50] the semantics of the value-passing calculus is given as a translation into the pure calculus. The translations rests upon the idea that a single label a of VSPA, with n parameters with sorts $S_1 \ldots S_n$, becomes the set of labels $\{a_{v_1 \ldots v_n} : v_i \in S_i, \forall i \in [1, n]\}$ in SPA. We consider only agents without free variables because if an agent has a free variable then it becomes a family of agents, one for each value of the variable. The translation can be given recursively on the structure of agents. Note that, since we have no free variable all the value and boolean expressions can be calculated; we will write $\widehat{b}$ and $\widehat{e}$ for the value obtained by evaluating a boolean expression b and a value expression e respectively.

We will use the notation $E\{a/b\}$ to represent agent E with all the occurrences of b substituted by a. We will also use $\mathcal{E}^+$ to denote the set of VSPA agents. For each agent $E \in \mathcal{E}^+$ without free variables, its translation $[\![E]\!]$ is given in Table 3 where $ari(a) = S_1 \dots S_n$; $\widehat{L} = \{l_{v_1,\dots,v_n} : l \in L, ari(l) = S_1 \dots S_n, v_i \in S_i, \forall i \in [1,n]\}$ is the set of the translations of actions in L and $\widehat{f}(l_{v_1,\dots,v_n}) = f(l)_{v_1,\dots,v_n}$ is the translation of relabelling function f. Furthermore, the single definition $A(x_1,\dots,x_m) \stackrel{\text{def}}{=} E$ with $ari(A) = S_1 \dots S_m$, is translated to the set of equations:

$E \in \mathcal{E}^+$	$[\![E]\!] \in \mathcal{E}$
$\underline{0}$	$\underline{0}$
$a(x_1,\dots,x_n).E$	$\sum_{i\in[1,n],v_i\in S_i} a_{v_1,\dots,v_n}.[\![E\{v_1/x_1,\dots v_n/x_n\}]\!]$
$\bar{a}(e_1,\dots,e_n).E$	$\bar{a}_{\widehat{e_1},\dots,\widehat{e_n}}.[\![E]\!]$
$\tau.E$	$\tau.[\![E]\!]$
$E_1 + E_2$	$[\![E_1]\!] + [\![E_2]\!]$
$E_1 \vert E_2$	$[\![E_1]\!] \vert [\![E_2]\!]$
$E \setminus L$	$[\![E]\!] \setminus \widehat{L}$
$E \setminus_I L$	$[\![E]\!] \setminus_I \widehat{L}$
E/L	$[\![E]\!]/\widehat{L}$
$E[f]$	$[\![E]\!][\widehat{f}]$
$A(e_1,\dots,e_n)$	$A_{\widehat{e_1},\dots,\widehat{e_n}}$
if b **then** E	$\begin{cases} [\![E]\!] \text{ if } \widehat{b} = True \\ \underline{0} \quad \text{otherwise} \end{cases}$

Table 3. Translation of VSPA to SPA.

$$\{A_{v_1,\dots,v_m} = [\![E\{v_1/x_1,\dots,v_m/x_m\}]\!]; v_i \in S_i \forall i \in [1,m]\}$$

Note that we do not partition the set of actions into two levels; we directly refer to the partition in the pure calculus. In this way it is possible for a certain action in VSPA to correspond, in the translation, to actions at different levels in SPA. This can be useful if we want a parameter representing the level of a certain action. As an example consider an action $access\_r(l,x)$ with $l \in \{high, low\}$ and $x \in [1,n]$, representing a read request from a user at level l to an object x; we can assign the high level to the actions with $l = high$ and the low level to the others in this way: $access\_r(high,x) \in Act_H$ and $access\_r(low,x) \in Act_L$ for all $x \in [1,n]$. [6]

[6] Note that $access\_r(high,x)$ stands for $access\_r_{high,x}$, with $x \in [1,n]$. Indeed, for the sake of readability, we often write $c(v)$ instead of its translation c_v.

A VSPA agent is finite state if its corresponding SPA agent is so. Hence, in general, a necessary condition is that every variable can assume values over a finite set only.

2.4 The Access Monitor

Here we give a more complex example of a VSPA agent specification. It is an access monitor which handles read and write requests on two binary variables enforcing the multilevel security policy. We will analyse and modify this example in the next sections in order to assess the merits of the various information flow properties that we will propose.

Example 1. Consider the system in Table 4 where $x, y, z, l \in \{0, 1\}$, $L = \{r,$

$$
\begin{aligned}
Access\_Monitor\_1 &\stackrel{\text{def}}{=} (Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L \\
Monitor &\stackrel{\text{def}}{=} access\_r(l,x). \\
&\quad\; (\textbf{if } x \leq l \textbf{ then} \\
&\qquad\quad r(x,y).\overline{val}(l,y).Monitor \\
&\qquad \textbf{else} \\
&\qquad\quad \overline{val}(l,err).Monitor) \\
&\quad\; + \\
&\quad\; access\_w(l,x).write(l,z). \\
&\quad\; (\textbf{if } x \geq l \textbf{ then} \\
&\qquad\quad \overline{w}(x,z).Monitor \\
&\qquad \textbf{else} \\
&\qquad\quad Monitor) \\
Object(x,y) &\stackrel{\text{def}}{=} \overline{r}(x,y).Object(x,y) + w(x,z).Object(x,z)
\end{aligned}
$$

Table 4. The *Access_Monitor_1* System.

$w\}$ and $\forall i \in \{0,1\}$ we have $r(1,i)$, $w(1,i)$, $access\_r(1,i)$, $val(1,i)$, $val(1,err)$, $access\_w(1,i), write(1,i) \in Act_H$ and all the other actions are low level ones. Note that in *Access_Monitor_1* every variable can assume values over a finite set only. When we translate it into SPA, we obtain a net of automata, hence *Access_Monitor_1* is a finite state agent.

Figure 6 represents process *Access_Monitor_1* that handles read and write requests from high and low level users on two binary objects: a high level variable and a low level one. It achieves *no read up* and *no write down* access control rules allowing a high level user to read from both objects and write only on the high one; conversely, a low level user is allowed to write on both objects and read

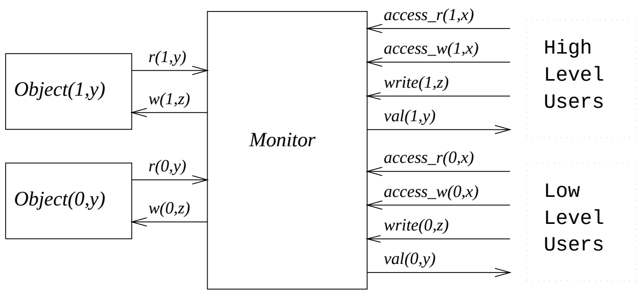


Fig. 6. The Access Monitor for Example 1.

only from the low one. Users interact with the monitor through the following access actions: $access\_r(l,x), access\_w(l,x), write(l,z)$ where l is the user level ($l = 0$ low, $l = 1$ high), x is the object ($x = 0$ low, $x = 1$ high) and z is the binary value to be written.

As an example, consider $access\_r(0,1)$ which represents a low level user ($l = 0$) read request from the high level object ($x = 1$), and $access\_w(1,0)$ followed by $write(1,0)$ which represents a high level user ($l = 1$) write request of value 0 ($z = 0$) on the low object ($x = 0$). Read results are returned to users through the output actions $val(l,y)$. This can be also an error in case of a read-up request. Note that if a high level user tries to write on the low object – through $access\_w(1,0)$ followed by $write(1,z)$ – such a request is not executed and no error message is returned.

In order to understand how the system works, let us consider the following transitions sequence representing the writing of value 1 in the low level object, performed by the low level user:

$$
\begin{array}{ll}
(Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L & \\
\xrightarrow{access\_w(0,0)} & (write(0,z).\overline{w}(0,z).Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L \\
\xrightarrow{write(0,1)} & (\overline{w}(0,1).Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L \\
\xrightarrow{\tau} & (Monitor \mid Object(1,0) \mid Object(0,1)) \setminus L
\end{array}
$$

The trace corresponding to this sequence of transitions is

$$access\_w(0,0).write(0,1)$$

and so we can write:

$$
\begin{array}{l}
(Monitor \mid Object(1,0) \mid Object(0,0)) \setminus L \\
\stackrel{access\_w(0,0).write(0,1)}{\Longrightarrow} (Monitor \mid Object(1,0) \mid Object(0,1)) \setminus L
\end{array}
$$

Note that, after the execution of the trace, the low level object contains value 1.

$Access\_Monitor\_1$ is a value passing specification of an access monitor. Its translation into CoSeC syntax for pure SPA is reported in Table 12 of Section 4.4. As an example, here we provide the translation of $Object(x, y)$ into the pure calculus by means of the following four constant definitions:

$$Object_{00} \stackrel{\text{def}}{=} \overline{r}_{00}.Object_{00} + w_{00}.Object_{00} + w_{01}.Object_{01}$$
$$Object_{01} \stackrel{\text{def}}{=} \overline{r}_{01}.Object_{01} + w_{00}.Object_{00} + w_{01}.Object_{01}$$
$$Object_{10} \stackrel{\text{def}}{=} \overline{r}_{10}.Object_{10} + w_{10}.Object_{10} + w_{11}.Object_{11}$$
$$Object_{11} \stackrel{\text{def}}{=} \overline{r}_{11}.Object_{11} + w_{10}.Object_{10} + w_{11}.Object_{11}$$

Note that we have, for every possible value of the pair (x, y), one different process $Object_{xy}$. ■

3 Information Flow Properties

In this Section we present some Information Flow properties. The common intuition behind all these properties is strictly related to the classic notion of *Non Interference* (NI, for short) [36], i.e., the low level users should not be able to deduce anything about high level users' activity.

As already mentioned earlier in this chapter, processes in $\mathcal{E}_H$ or $\mathcal{E}_L$ are secure by isolation, as they are confined to work in one single confidentiality level. The really interesting processes are those built with both high level and low level actions, as they may show unwanted information flows.

In the first three sections we present properties based on different equivalence notions. We start with the weakest and most intuitive one: trace equivalence. Then, we see that it is necessary to move to finer equivalence notions in order to detect possible high level deadlocks that can compromise the security of the system. For this reason, in the second section we base our security properties on failure and testing equivalences [9, 52], which have been designed specifically to detect deadlocks. However, we discover that the failure/testing setting is not ideal for our purposes because of the way it deals with (potential) divergences. This leads us, in the third section, to prefer to base our security properties on the notion of weak bisimulation. In the fourth section we show that deadlocks due to high level activity are indeed dangerous and cannot be ignored. In section 3.5 we compare our security properties with other proposals in the literature which are also based on Process Algebras.

Part of the material contained in this section has been published in [24, 26, 21, 23, 22, 18].

3.1 Properties Based on Trace Equivalence

We start with *Non-deterministic Non Interference* (*NNI*, for short) [24, 26], which is a natural generalization to non-deterministic systems of NI (assuming two user groups only). The basic idea of NI is that the high level does not interfere with the low level if the effects of high level inputs are not visible by a low level user. This idea can be rephrased on the LTS model as follows. We consider every trace γ of the system containing high level inputs. Then, we look if there exists another trace γ' with the same subsequence of low level actions and without high inputs. A low level user, which can only observe low level actions, is not able to distinguish γ and γ'. As both γ and γ' are legal traces, we can conclude that the possible execution of the high level inputs in γ has no effect on the low level view of the system.

As for NI, we can define this property by using some functions which manipulates sequences of actions. In particular, it is sufficient to consider the function $low : \mathcal{L}^* \longrightarrow Act_L^*$ which takes a trace γ and removes all the high level actions from it, i.e., returns the low level subsequence of γ. Moreover we use the function $highinput : \mathcal{L}^* \longrightarrow (Act_H \cap I)^*$ which extracts from a trace the subsequence composed of all the high level inputs.

Definition 8. *(NNI: Non-deterministic Non Interference)*
$E \in NNI$ if and only if $\forall \gamma \in T(E), \exists \delta \in T(E)$ such that

(i) $low(\gamma) = low(\delta)$
(ii) $highinput(\delta) = \langle\rangle$

where $\langle\rangle$ denotes the empty sequence. ■

This may be expressed algebraically as:

Proposition 2. $E \in NNI \Longleftrightarrow (E \setminus_I Act_H)/Act_H \approx_T E/Act_H$.

PROOF. ($\Rightarrow$) It is enough to show that if E is NNI then $T(E/Act_H) \subseteq T((E \setminus_I Act_H)/Act_H)$, because the opposite inclusion simply derives from $T(E \setminus_I Act_H) \subseteq T(E)$. Let $\gamma \in T(E/Act_H)$; then, by definition of / operator, $\exists \gamma' \in T(E)$ such that $low(\gamma') = \gamma$. Since $E \in NNI$ then $\exists \delta \in T(E)$ such that $low(\gamma') = low(\delta)$ and $highinput(\delta) = \langle\rangle$. Hence $\delta \in T(E \setminus_I Act_H)$ and $\delta' = low(\delta) \in T((E \setminus_I Act_H)/Act_H)$. Since $\gamma = low(\gamma') = low(\delta) = \delta'$, then $\gamma = \delta'$ and thus $\gamma \in T((E \setminus_I Act_H)/Act_H)$.
($\Leftarrow$) Let $\gamma \in T(E)$. Then $\exists \delta \in T(E \setminus_I Act_H)$ such that $low(\gamma) = low(\delta)$. Since $\delta \in T(E \setminus_I Act_H)$ then $highinput(\delta) = \langle\rangle$ and $\delta \in T(E)$. ■

As a matter of fact, in E/Act_H all the high level actions are hidden, hence giving the low level view of the system; $E \setminus_I Act_H$ instead prevents traces from containing high level inputs. So, if the two terms are equivalent, then for every trace with high level inputs we can find another trace without such actions but with the same subsequence of low level actions.

In the following we will consider this algebraic characterization as the definition of NNI. Indeed, all the other properties we present below in this section are defined using this compact algebraic style. An interesting advantage of this style is that it reduces the check of a security property to the "standard" and well studied problem of checking the semantic equivalence of two LTSs.

It is possible to prove that $Access\_Monitor\_1$ of Example 1 is NNI. In fact, the next example shows that NNI is able to detect whether the multilevel access control rules are implemented correctly in the monitor.

Example 2. Consider the modified monitor [7] in Table 5 which does not control write accesses: Now it is possible for a high level user to write in the low level

$$
\begin{aligned}
Access\_Monitor\_2 &\stackrel{\text{def}}{=} (Monitor\_2 \mid Object(1,0) \mid Object(0,0)) \setminus L \\
Monitor\_2 &\stackrel{\text{def}}{=} access\_r(l,x). \\
&\quad (\textbf{if } x \leq l \textbf{ then} \\
&\qquad r(x,y).\overline{val}(l,y).Monitor\_2 \\
&\quad\ \textbf{else} \\
&\qquad \overline{val}(l,err).Monitor\_2) \\
&\quad + \\
&\quad access\_w(l,x).write(l,z).\overline{w}(x,z).Monitor\_2
\end{aligned}
$$

Table 5. The *Access_Monitor_2* System.

object (action $access\_w(1,0)$ followed by $write(1,z)$) so the system is not secure. We have that *NNI* is able to detect this kind of direct flow. *Access_Monitor_2* can execute the following trace:

$$\gamma = access\_w(1,0).write(1,1).access\_r(0,0).\overline{val}(0,1)$$

In γ we have two accesses to the monitor: first a high level user modifies the value of the low object writing down value 1, and then the low user reads value 1 from the object. If we purge γ of high level actions, we obtain the sequence

$$\gamma' = access\_r(0,0).\overline{val}(0,1)$$

representing the reading by a low level user of value 1 from the low object. This trace is not a legal trace for *Access_Monitor_2* since the low object is initialized to value 0. Moreover, it is not possible to obtain a trace for *Access_Monitor_2* by adding to γ' only high level outputs, because all the high level outputs in *Access_Monitor_2* are prefixed by high level inputs. Hence γ' is not even a trace for $(Access\_Monitor\_2 \setminus_I Act_H)/Act_H$. In other words, it is not possible to find a trace γ'' of *Access_Monitor_2* with the same low level actions of γ and without high level inputs.

Since γ' is a trace for agent $Access\_Monitor\_2/Act_H$ but it is not a trace for agent $(Access\_Monitor\_2 \setminus_I Act_H)/Act_H$, we conclude that *Access_Monitor_2* is not *NNI*. ■

The example above shows that *NNI* reveals if something is wrong in the access policy we are implementing. Indeed, it is quite intuitive that if some access rule

[7] In the following, if an agent is not specified (e.g. $Object(x,y)$) we mean that it has not been modified with respect to the previous versions of the Access Monitor.

is missing then there will be a particular execution where some classified information is disclosed to low level users (otherwise such a rule would be useless). This will certainly modify the low level view of the system making it not *NNI* for sure.

However, the next example shows that *NNI* is not adequate to deal with synchronous communications and, consequently, it is too weak for SPA agents.

Example 3. Consider $Access\_Monitor\_1$, and suppose that we want to add a high level output signal which informs high level users about write operations of low level users in the high level object. This could be useful to know the integrity of high level information. We obtain the VSPA agent of Table 6 with the new *written_up* action and where $\forall i \in \{0,1\}$, $written\_up(i) \in Act_H$.

$$
\begin{array}{rl}
Access\_Monitor\_3 \stackrel{\mathrm{def}}{=} & (Monitor\_3 \mid Object(1,0) \mid Object(0,0)) \setminus L \\
Monitor\_3 \stackrel{\mathrm{def}}{=} & access\_r(l,x). \\
& (\ \mathbf{if}\ x \leq l\ \mathbf{then} \\
& \quad r(x,y).\overline{val}(l,y).Monitor\_3 \\
& \ \mathbf{else} \\
& \quad \overline{val}(l,err).Monitor\_3) \\
& \ + \\
& access\_w(l,x).write(l,z). \\
& (\ \mathbf{if}\ x = l\ \mathbf{then} \\
& \quad \overline{w}(x,z).Monitor\_3 \\
& \ \mathbf{else} \\
& \quad\quad \mathbf{if}\ x > l\ \mathbf{then} \\
& \quad\quad\quad \overline{w}(x,z).\overline{written\_up}(z).Monitor\_3 \\
& \quad\quad \mathbf{else} \\
& \quad\quad\quad Monitor\_3)
\end{array}
$$

Table 6. The *Access_Monitor_3* System.

It is possible to prove that $Access\_Monitor\_3$ is NNI. However, consider the following trace of $Access\_Monitor\_3$:

$$\gamma = access\_w(0,1).write(0,0).\overline{written\_up}(0).access\_w(0,1).write(0,0)$$

where a low level user writes two times value 0 into the high level object. If we purge γ of high level actions (i.e. of *written_up*) we obtain the following sequence:

$$\gamma' = access\_w(0,1).write(0,0).access\_w(0,1).write(0,0)$$

that cannot be a trace for $Access\_Monitor\_3$, because after every low level write operation there must be an action $written\_up$.

So, if a low level user succeeds in executing the two write requests, then (s)he will know that some high level user has "accepted" the high level output $\overline{written\_up}$ (because of synchronous communications). In other words, a high level user can interfere with a low level one accepting or not the high level output $\overline{written\_up}$. *NNI* is not able to detect this, because it verifies only the high level input interferences over low level actions. In fact, since γ is a trace of $Access\_Monitor\_3$, then γ' is a trace for both $Access\_Monitor\_3/Act_H$ and $(Access\_Monitor\_3 \setminus_I Act_H)/Act_H$. ■

The example above shows that synchronous communications induce a symmetry over inputs and outputs. For this reason, we define a symmetric form of *NNI*. It requires that, for every trace γ, the sequence γ', obtained from γ by deleting all the high level actions, is still a trace. This property is called *Strong NNI* (*SNNI* for short).

Definition 9. $E \in SNNI \Leftrightarrow E/Act_H \approx_T E \setminus Act_H$. ■

We have that $Access\_Monitor\_3$ is not *SNNI* since γ' is a trace for the agent $Access\_Monitor\_3/Act_H$ and γ' is not a trace for $Access\_Monitor\_3 \setminus Act_H$. The reason why the name "*Strong NNI*" has been chosen is a consequence of the following result:

Proposition 3. $SNNI \subset NNI$.

Proof. If $E \in SNNI$ then $E/Act_H \approx_T E \setminus Act_H$. Since $T(E \setminus Act_H) \subseteq T((E \setminus_I Act_H)/Act_H)$ and $T((E \setminus_I Act_H)/Act_H) \subseteq T(E/Act_H)$ then $E \in NNI$. The inclusion is strict because $E = \bar{h}.l.0$ is *NNI* but not *SNNI*. ■

It is thus possible to prove that $Access\_Monitor\_1$ of Example 1 is also *SNNI*.

SNNI seems to be quite satisfactory in a trace equivalence setting. Unfortunately, in the following example, we will see that trace equivalence – as the basic equivalence for security properties – is too weak; in particular, it is not able to detect deadlocks due to high level activities, that influence the security of a system.

Example 4. Suppose we have a high level action h_stop which explicitly stops the monitor. Obviously, in such a case there is a possible deadlock caused by a high level activity. In particular, consider the system in Table 7 where $h\_stop \in Act_H$. It is possible to prove that it is *SNNI*. This is because trace equivalence is not able to detect deadlocks and h_stop does not modify the low traces of the system. It could seem that a deadlock caused by a high level activity is not really interfering with any low level users, since a low level user, trying to make an access to the monitor, is not able to conclude that the monitor is blocked. However such a user can obviously deduce that the system is not blocked every time it accepts some access requests or gives some outputs. In the case of $Access\_Monitor\_4$, a low level user will never be able to conclude that h_stop has been executed; nonetheless, at every interaction with the system, the user

$$
\begin{array}{rl}
Access\_Monitor\_4 & \stackrel{\text{def}}{=} (Monitor\_4 \mid Object(1,0) \mid Object(0,0)) \setminus L \\
Monitor\_4 & \stackrel{\text{def}}{=} access\_r(l,x). \\
& (\ \textbf{if } x \leq l \textbf{ then} \\
& \quad\quad r(x,y).\overline{val}(l,y).Monitor\_4 \\
& \ \ \textbf{else} \\
& \quad\quad \overline{val}(l,err).Monitor\_4) \\
& \ \ + \\
& access\_w(l,x).write(l,z). \\
& (\ \textbf{if } x \geq l \textbf{ then} \\
& \quad\quad \overline{w}(x,z).Monitor\_4 \\
& \ \ \textbf{else} \\
& \quad\quad Monitor\_4) \\
& \ \ + \\
& h\_stop.\underline{0}
\end{array}
$$

Table 7. The *Access_Monitor_4* System.

will know that *Access_Monitor_4* is not blocked and so that *h_stop* has not been executed yet. In section 3.4 we will show how the subtle information flow caused by a potential deadlock can be exploited in order to construct an information channel from high level to low level. ∎

In order to detect this kind of flows, it is necessary to use some notion of equivalence which is able to detect deadlocks. Note that by simply changing the equivalence notion in the definition of *SNNI* we obtain a security property which inherits all the observation power of the new equivalence notion. So, for detecting deadlocks, one obvious possibility could be the failure/testing setting [9, 52], that has been designed for this purpose.

3.2 Detecting High Level Deadlocks Through Failure/Testing Equivalences

Consider the version of *SNNI* based on testing equivalence:

Definition 10. *(testing SNNI)* $E \in TSNNI \iff E/Act_H \approx_{test} E \setminus Act_H$ ∎

We have that $Access\_Monitor\_4 \notin TSNNI$. In fact it is sufficient to consider the test $T \stackrel{\text{def}}{=} \overline{access\_r}(0,0).\omega.\underline{0}$ which is able to detect the deadlock introduced by action *h_stop*. In particular, we have that:

$$Access\_Monitor\_4 \setminus Act_H \ \ must\ T$$

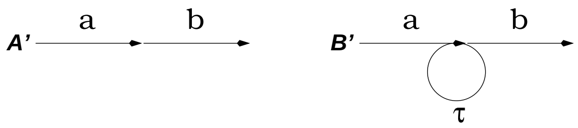


Fig. 7. Divergence makes A and B non testing equivalent.

$$Access\_Monitor\_4/Act_H \not{m}ust\ T$$

Intuitively, when we hide h_stop in $Access\_Monitor\_4/Act_H$ we obtain an internal transition to a deadlock state. If the system moves to that state before the execution of $access\_r(0,0)$ the test will not be satisfied.

So, it seems that *TSNNI* is exactly the deadlock-sensitive extension of *SNNI* we were looking for. However, in the following we want to show that for systems with some high level loops or with τ loops, *TSNNI* is not interesting and is not able to detect security flaws. This is caused by the way testing equivalence deals with divergences (infinite sequences of τ actions). In fact, it is possible to prove that if E is divergent, then the (failure) condition (ii) of $\approx_{test}$ is verified if and only if also F is divergent. In the failure setting, when we have a divergent state we can observe the behaviour of a process only before reaching that state. All the actions after a divergence cannot be observed. In section 2.2, we have seen that $A' \stackrel{\text{def}}{=} a.b.0$ is not testing equivalent to $B' \stackrel{\text{def}}{=} a.B''$ with $B'' \stackrel{\text{def}}{=} \tau.B'' + b.0$. The only difference between A' and B' is the divergence after the execution of a in B' (see Figure 7).

Now, consider a system E with some high level loops and without divergences. When we hide high level actions in *TSNNI* with operation E/Act_H, we obtain some τ loops and so E/Act_H is divergent. As we said above, this will be failure equivalent to $E \setminus Act_H$ only if $E \setminus Act_H$ is divergent as well. However since E is without divergences, $E \setminus Act_H$ cannot have divergences too. We conclude that this kind of systems cannot be *TSNNI* no matter what the interactions between high and low level actions are. Note that every recursive high level system (with no low level actions and so secure by definition) is of this kind. All the access monitors we have seen so far have this feature as well.

Moreover we have an even worse situation if we consider a divergent process D, as also $D \setminus Act_H$ and D/Act_H are divergent. So for these two processes (ii) is verified. This means that for divergent processes $\approx_{test}$ becomes equal to $\approx_T$. Hence *TSNNI* becomes equal to *SNNI* and, in general, all the properties based on testing equivalence become equal to the corresponding ones based on trace equivalence. Now, we present an example of a divergent *TSNNI* process which, nonetheless, contains some high level deadlocks.

Example 5. Consider again agent $Access\_Monitor\_4$. We want to add a backup feature which is able to make a copy of the values stored in objects periodically. Obviously, there should be also a recovery procedure, but we do not model this in order to simplify as much as possible the example. We have a first process which represents the backup timer and sends periodically a signal in order to obtain a backup. It is an abstraction of a clock, since in SPA it is not possible to handle time directly.

$$Backup\_timer \stackrel{\text{def}}{=} \overline{backup}.Backup\_timer$$

Then we slightly modify the $Monitor$ process by inserting two actions which suspend its execution until the backup is finished.

$$\begin{aligned} Monitor\_B \stackrel{\text{def}}{=} & \dots \\ & \text{the same as in } Access\_Monitor\_4 \\ & \dots \\ & + start\_backup.end\_backup.Monitor \end{aligned}$$

The backup process is enabled by the timer, then it stops the monitor, reads the values of variables, stores them into two additional objects ($Object(2, y)$ and $Object(3, y)$) and resumes the monitor:

$$\begin{aligned} Backup \stackrel{\text{def}}{=} & backup. \\ & \overline{start\_backup}. \\ & r(0, y).r(1, z). \\ & \overline{w}(2, y).\overline{w}(3, z). \\ & \overline{end\_backup}. \\ & Backup \end{aligned}$$

The access monitor with backup is given by the following system:

$$\begin{aligned} Access\_Monitor\_B \stackrel{\text{def}}{=} & (Monitor\_B \mid Backup\_timer \mid Backup \mid Object(0, 0) \mid \\ & \mid Object(1, 0) \mid Object(2, 0) \mid Object(3, 0)) \setminus L \end{aligned}$$

where $L = \{r, w, start\_backup, end\_backup, backup\}$. As a result, the backup procedure of the system is something internal, i.e., an external user can see nothing of the backup task. This makes the system divergent. In fact, if the variable values are unchanged, then the backup procedure is a τ loop that moves the system to the same state where it started the backup. For weak bisimulation this is not a problem and we can analyze this new system as well. In particular, we can check with the CoSeC tool (presented in Section 4) that $Access\_Monitor\_B$ is observationally equivalent to $Access\_Monitor\_4$. This is enough to prove (Theorem 5) that every security analysis made on $Access\_Monitor\_4$ is valid also for $Access\_Monitor\_B$. In particular, $Access\_Monitor\_B$ is not secure because of the potential high level deadlock we have explicitly added in $Access\_Monitor\_4$.

On the other hand, if we try to analyze this system with some testing equivalence based property, we have an inaccurate result. Indeed $Access\_Monitor\_B$, differently from $Access\_Monitor\_4$, is *TSNNI*. This happens because process

$Access\_Monitor\_B$ is divergent and so processes $Access\_Monitor\_B/Act_H$ and $(Access\_Monitor\_B|\Pi) \setminus Act_H$ $(\forall \Pi \in \mathcal{E}_H)$ are divergent as well. Thus they are failure equivalent and, since $Access\_Monitor\_B$ is *SNNI* they are also trace (and so testing) equivalent. ∎

There is also an interesting, practical difference between bisimulation and failure/testing. On the one hand, it is possible to check bisimulation (observational equivalence) in polynomial time [42]. On the other hand, we have that the problem of establishing language equivalence of nondeterministic finite automata is reducible in polynomial time to the problem of checking any of the testing, failure, and trace equivalences. Such a problem has been proved to be PSPACE-complete [61]. The consequence is that all these problems are PSPACE-complete as well.

Moreover it is interesting to observe that failure/testing equivalences are not known to be decidable on infinite state systems. On the other hand, there are some interesting results on the decidability of weak bisimulation over some classes of infinite state systems, e.g. totally normed Basic Process Algebras (BPA) [38].[8] For instance it is possible to define a BPA agent representing an unbounded queue.

All these arguments have convinced us to adopt bisimulation and observational equivalence as the default semantic equivalence for our properties.

In the next section we move to weak bisimulation and observational equivalence. As we have seen, these notions are able to detect deadlocks as well. Moreover they give a *fair* interpretation of divergence, i.e., they assume that a τ loop will be executed an arbitrary, yet finite, number of times. In this way they can observe system behaviour also after divergences.

3.3 Properties Based on Observational Equivalence

We introduce the bisimulation-based security properties *BNNI* and *BSNNI*, by substituting $\approx_B$ for $\approx_T$ in their SPA-based definitions.

Definition 11. *(Bisimulation NNI, SNNI)*

(i) $E \in BNNI \iff E/Act_H \approx_B (E \setminus_I Act_H)/Act_H$
(ii) $E \in BSNNI \iff E/Act_H \approx_B E \setminus Act_H$ ∎

As expected, it can be proved that each of these new properties is properly finer than its corresponding trace-based one.

Proposition 4. *The following hold:*

(i) $BNNI \subset NNI$,
(ii) $BSNNI \subset SNNI$,

[8] BPA [6] are basically the transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are permitted. In order to obtain an LTS an action is associated with every rule.

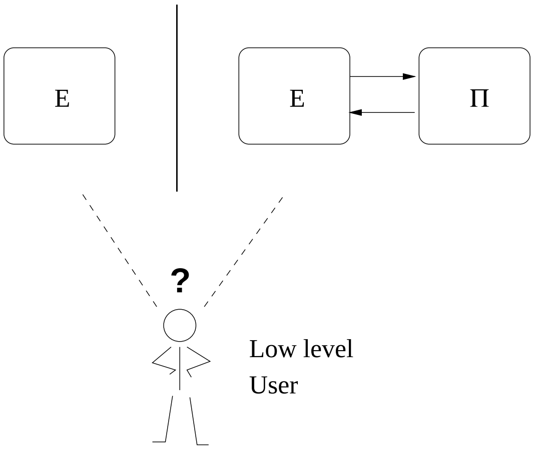


Fig. 8. BNDC intuition

PROOF. It immediately follows from the fact that $E \approx_B F$ implies $E \approx_T F$. The inclusions are proper because $E = \tau.l.0 + \tau.h.l.0$ is such that $E \in \mathit{NNI}, \mathit{SNNI}$ and $E \notin \mathit{BNNI}, \mathit{BSNNI}$. ■

Consider again $Access\_Monitor\_4$ containing the h_stop event. It is neither *BSNNI* nor *BNNI*, as observational equivalence is able to detect deadlocks. In particular, $Access\_Monitor\_4/Act_H$ can move to $\underline{0}$ through an internal action τ, while $Access\_Monitor\_4 \setminus Act_H$ is not able to reach (in zero or more τ steps) a state equivalent to $\underline{0}$.

Now we want to show that *BSNNI* and *BNNI* are still not able to detect some potential deadlocks due to high level activities. This will induce us to propose another property based on a different intuition. Let us consider $Access\_Monitor\_1$. We can prove that such a system is *BSNNI* as well as *BNNI*. However, the following two dangerous situations are possible: (i) a high level user makes a read request without accepting the corresponding output from the monitor (remember that communications in SPA are synchronous) and (ii) a high level user makes a write request and does not send the value to be written. In both cases we have a deadlock due to a high level activity that *BNNI* and *BSNNI* are not able to reveal. To solve this problem, we are going to present a stronger property, called *Bisimulation-based Non Deducibility on Compositions* (*BNDC*, for short). It is simply based on the idea of checking the system against all high level potential interactions. A system E is *BNDC* if for every high level process Π a low level user cannot distinguish E from $(E|\Pi) \setminus Act_H$. In other words, a system E is *BNDC* if what a low level user sees of the system is not modified by composing any high level process Π to E (see Figure 8).

Definition 12. *E is BNDC iff* $\forall \Pi \in \mathcal{E}_H,\ E/Act_H \approx_B (E \mid \Pi) \setminus Act_H$. ■

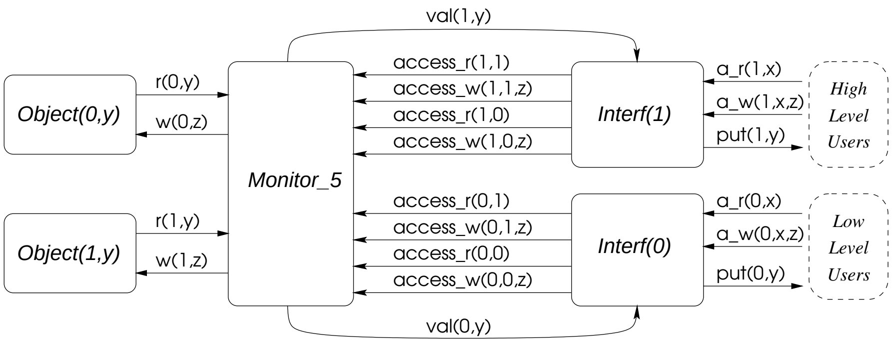


Fig. 9. The *BNDC Access_Monitor_5*

Example 6. We want to show that $Access\_Monitor\_1$ is not $BNDC$. Consider $\Pi = \overline{access\_r}(1,1).\underline{0}$. System $(Access\_Monitor\_1 \mid \Pi) \setminus Act_H$ will be blocked immediately after the execution of the read request by Π, moving to the following deadlock state:

$$((\overline{val}(1,0).Monitor \mid Object(0,0) \mid Object(1,0)) \setminus L \mid 0) \setminus Act_H$$

This happens because Π executes a read request and does not wait for the corresponding return value (action val). We conclude that Π can interfere with low level users. Since there are no possible deadlocks in $Access\_Monitor\_1/Act_H$, we find out that $(Access\_Monitor\_1|\Pi)\setminus Act_H \not\approx_B Access\_Monitor\_1/Act_H$ and so $Access\_Monitor\_1$ is not $BNDC$.

Moreover, there is another potential source of deadlock when a high level user makes a write request and does not send the value to be written. In particular, the high level user $\Pi' = \overline{access\_w}(1,0).\underline{0}$ will block system $(Access\_Monitor\_1|\Pi')\setminus Act_H$ immediately after the execution of the write request by Π', moving the system to the following deadlock state:

$$(((write(1,0).\overline{w}(1,0).Monitor + write(1,1).\overline{w}(1,1).Monitor) \mid Object(0,0) \mid \\ \mid Object(1,0)) \setminus L \mid 0) \setminus Act_H$$

Again, we have $(Access\_Monitor\_1|\Pi') \setminus Act_H \not\approx_B Access\_Monitor\_1/Act_H$. In order to obtain a $BNDC$ access monitor, we modify the monitor by adding an interface for each level which temporarily stores the output value of the monitor (passing it later to the users and thus making communication asynchronous) and that guarantees mutual exclusion within the same level; moreover, we use an atomic action for write request and value sending. Note that, because of the interface, actions $access\_r$, $access\_w$ and val become a_r, a_w and put, respectively. The resulting system is reported in Table 8 (see also Figure 9).

In such a system we have that $k \in \{0, 1, err\}$, $L = \{r, w\}$, $N = \{val, access\_r, access\_w\}$ and $a\_r(1,x)$, $a\_w(1,x)$, $put(1,y) \in Act_H$ $\forall x \in \{0,1\}$ and

$$
\begin{aligned}
Access\_Monitor\_5 &\stackrel{\text{def}}{=} (AM \mid Interf) \setminus N \\
AM &\stackrel{\text{def}}{=} (Monitor\_5 \mid Object(1,0) \mid Object(0,0)) \setminus L \\
Monitor\_5 &\stackrel{\text{def}}{=} access\_r(l,x). \\
&\quad (\ \mathbf{if}\ x \leq l\ \mathbf{then} \\
&\qquad\quad r(x,y).\overline{val}(l,y).Monitor\_5 \\
&\quad\ \ \mathbf{else} \\
&\qquad\quad \overline{val}(l,err).Monitor\_5) \\
&\quad + \\
&\quad access\_w(l,x,z). \\
&\quad (\ \mathbf{if}\ x \geq l\ \mathbf{then} \\
&\qquad\quad \overline{w}(x,z).Monitor\_5 \\
&\quad\ \ \mathbf{else} \\
&\qquad\quad Monitor\_5) \\
Interf &\stackrel{\text{def}}{=} Interf(0) \mid Interf(1) \\
Interf(l) &\stackrel{\text{def}}{=} a\_r(l,x).\overline{access\_r}(l,x).val(l,k).\overline{put}(l,k).Interf(l) \\
&\quad + \\
&\quad a\_w(l,x,z).\overline{access\_w}(l,x,z).Interf(l)
\end{aligned}
$$

Table 8. The *Access_Monitor_5* System.

$\forall y \in \{0, 1, err\}$, while the same actions with 0 as first parameter belong to Act_L. It is possible to verify that $Access\_Monitor\_5$ is *BNDC* using the automatically checkable property we are going to present. Table 9 summarizes the properties

Ver.	Description	*NNI*	*SNNI*	*BNNI*	*BSNNI*	*BNDC*
1	Multilevel rules	X	X	X	X	
2	Without write control					
3	With high signal for integrity	X				
4	With explicit high level deadlock	X	X			
5	With buffers and atomic write	X	X	X	X	X

Table 9. Properties satisfied by the different versions of the access monitor.

satisfied by the different versions of the access monitor. $\blacksquare$

The next theorem shows that the bisimulation-based properties are related in a different way with respect to the corresponding ones. Indeed, we have that *BNDC* is stronger than *BNNI* and *BSNNI*. Moreover $BSNNI \not\subset BNNI$ while for trace equivalence we had that $SNNI \subset NNI$. *BSNNI*

Theorem 1.

(i) *BNNI* $\not\subseteq$ *BSNNI and BSNNI* $\not\subseteq$ *BNNI.*
(ii) *BNDC* $\subset$ *BSNNI* $\cap$ *BNNI.*

Proof. (i) Let us consider the following agent $E = l.\overline{h}.l.h.l.0 + l.l.l.0 + l.l.0$; we have that $E \in BNNI$ and $E \notin BSNNI$. Let us now consider the agent $F = l.\overline{h}.l.h.l.0 + l.l.l.0 + l.0$; we have that $F \in BSNNI$ and $F \notin BNNI$.
(ii) To show that $BNDC \subseteq BSNNI$, consider $\Pi' = 0$ (the empty process). Then, by *BNDC* definition, we have that $E/Act_H \approx_B (E \mid 0) \setminus Act_H$ and so $E/Act_H \approx_B E \setminus Act_H$.
To show that $BNDC \subseteq BNNI$, consider $\Pi'' = \sum_{i \in Act_H \cap I} i.\Pi''$ (the process which accepts all the high inputs) then, by *BNDC* definition, we have that $E/Act_H \approx_B (E \mid \Pi'')\setminus Act_H$. Now it is sufficient to note that $(E \mid \Pi'')\setminus Act_H \approx_B (E \setminus_I Act_H)/Act_H$.
In order to show that the inclusion is strict we need a system which is both *BSNNI* and *BNNI*, and which is not *BNDC*. Such a system is $E = l.h.l.h.l.\underline{0} + l.l.l.\underline{0} + l.\underline{0}$. It is easy to see that it is *BSNNI* and *BNNI*. Moreover if we consider $\Pi''' = \bar{h}.\underline{0}$ we obtain that $(E \mid \Pi''') \setminus Act_H \approx_B l.l.\underline{0} + l.l.l.\underline{0} + l.\underline{0} \not\approx_B E$. $\blacksquare$

We need another simple result before we can draw a diagram of the relations among all the properties we have seen up to now.

Proposition 5. *BNNI* $\not\subseteq$ *SNNI.*

Proof. It is sufficient to consider agent $E = \bar{h}.l.0$ which is *BNNI* but not *SNNI*. $\blacksquare$

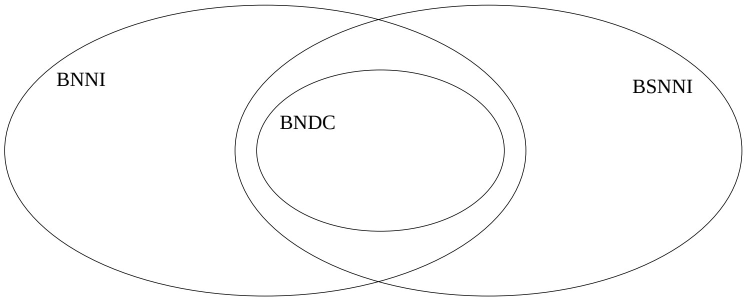


Fig. 10. The inclusion diagram for bisimulation-based properties

Figure 10 summarizes the relations among the bisimulation-based properties presented so far.

It is now interesting to study the trace equivalence version of *BNDC* called *NDC*. Indeed it could improve the *SNNI* property which is still our better proposal for the trace equivalence setting (no detection of deadlocks). Surprisingly, we find out that such property is exactly equal to *SNNI*.

Theorem 2. $NDC = SNNI$.

PROOF. We first prove that if $E \in NDC$ then $E \in SNNI$. By hypothesis, $E/Act_H \approx_T (E|0) \setminus Act_H$ for the specific $\Pi = 0$. Since $(E|0) \setminus Act_H \approx_T E \setminus Act_H$ then we have $E/Act_H \approx_T E \setminus Act_H$.

Now we want to prove that if $E \in SNNI$ then $E \in NDC$. By hypothesis, $E/Act_H \approx_T E \setminus Act_H$. Since $T(E \setminus Act_H) \subseteq T((E|\Pi) \setminus Act_H)$ then we have $T(E/Act_H) \subseteq T((E|\Pi) \setminus Act_H)$. Observe also that the reverse inclusion holds, in fact, if E and Π synchronize on a certain high action, then E/Act_H can always "hide" it. Hence $E/Act_H \approx_T ((E|\Pi) \setminus H)/Act_H$. ■

This, in a sense, confirms that the intuition behind *SNNI* (and so NI) is good at least in a trace equivalence model. Indeed, in such a model the fact that we check the system against every high level process is useless. It is sufficient to statically check if the hiding of high level actions corresponds to the restriction of such actions. This points out a critical point: *BNDC* is difficult to use in practice, because of the universal quantification on high level processes. It would be desirable to have an alternative formulation of *BNDC* which avoids universal quantification, exploiting local information only as for the trace-equivalence case; even if Martinelli has shown that *BNDC* is decidable over finite state processes [47], a solution to this problem is still to be found. In the same work, Martinelli also shows a negative fact regarding the verification of *BNDC*: it is not compositional, i.e., if two systems are *BNDC* their composition may be not so. This does not permit us to reduce the *BNDC*-security of a big system to the *BNDC*-security of its simpler subsystems and forces us to always prove *BNDC* over the whole system.

For these reasons, here we propose a sufficient condition for *BNDC*, namely the *SBSNNI* property, which exploits local information only and moreover is compositional (i.e., if two systems are *SBSNNI* their composition is *SBSNNI* too).

Definition 13. *(SBSNNI: Strong BSNNI)*
A system $E \in SBSNNI$ if and only if for all E' reachable from E we have $E' \in BSNNI$. ∎

SBSNNI is easily verifiable, as *BSNNI* is so; moreover, we can use it in order to check that a system is *BNDC* because of the following result.

Proposition 6. *$SBSNNI \subset BNDC$*

Proof. Let E be a system and Π a high level process. Let R be a relation defined as follows: $(E' \setminus Act_H, (E' \mid \Pi') \setminus Act_H) \in R$, for all E', Π' such that $E \Rightarrow E'$ and $\Pi \Rightarrow \Pi'$. We want to prove that if E is *SBSNNI* then R is a weak bisimulation up to $\approx_B$ (see [50]). There is only one non trivial case: $(E' \mid \Pi') \setminus Act_H \xrightarrow{\tau} (E'' \mid \Pi'') \setminus Act_H$. As there exists $h \in Act_H$ such that $E' \xrightarrow{h} E''$, then $E'/Act_H \xrightarrow{\tau} E''/Act_H$. By hypothesis, $E' \in BSNNI$ hence we have $E'/Act_H \approx_B E' \setminus Act_H$ and so there exists an agent E''' such that $E' \setminus Act_H \overset{\hat{\tau}}{\Longrightarrow} E''' \setminus Act_H$ and $E''' \setminus Act_H \approx_B E''/Act_H$. By hypothesis, $E'' \in BSNNI$ hence we also have that $E''/Act_H \approx_B E'' \setminus Act_H$ and so $E''' \setminus Act_H \approx_B E'' \setminus Act_H$. Since $(E'' \setminus Act_H, (E'' \mid \Pi'') \setminus Act_H) \in R$ and $E''' \setminus Act_H \approx_B E'' \setminus Act_H$ then R is a bisimulation up to $\approx_B$.

Now we have that $E \setminus Act_H \approx_B (E \mid \Pi) \setminus Act_H$ for all $\Pi \in \mathcal{E}_H$. Since $E \in BSNNI$, then $E/Act_H \approx_B E \setminus Act_H$. Therefore $E/Act_H \approx_B (E \mid \Pi) \setminus Act_H$. This means that $E \in BNDC$.

The inclusion is strict because agent $E = l.h.l.0 + l.0 + l.l.0$ is *BNDC* but not *SBSNNI*. ∎

The next theorem states that *SBSNNI* is *compositional*, i.e., preserved by the parallel and the restriction operators. This is useful in the automatic check of this property because it allows to check it directly on the subsystems thus reducing the exponential explosion of the states due to all possible interleavings of parallel systems. We will study this in details in the next section. Similar results of compositionality hold for *NNI* and *SNNI* [24].

Theorem 3. *The following hold:*

(i) $E, F \in SBSNNI \Longrightarrow (E|F) \in SBSNNI$
(ii) $E \in SBSNNI, L \subseteq \mathcal{L} \Longrightarrow E \setminus L \in SBSNNI$

Proof. We need the following:

Lemma 1. $(E|F)/Act_H \approx_B E/Act_H|F/Act_H$.

Proof. Consider the following relation: $((E'|F')/Act_H, E'/Act_H|F'/Act_H) \in R$ if and only if $E \Rightarrow E'$ and $F \Rightarrow F'$. It is easy to prove that R is a bisimulation. Indeed the only non trivial case is the synchronization $(E'|F')/Act_H \xrightarrow{\tau}$

$(E''|F'')/Act_H$ which is simulated by $E'/Act_H|F'/Act_H \stackrel{\tau.\tau}{\rightarrow} E''/Act_H|F''/Act_H$. ■

Now we can prove the Theorem.

(i) Consider the relation $((E'|F') \setminus Act_H, E' \setminus Act_H|F' \setminus Act_H) \in R$ for all E', F' such that $E \Rightarrow E'$ and $F \Rightarrow F'$. If we prove that R is a weak bisimulation up to $\approx_B$ then, by hypothesis and Lemma 1, we obtain the thesis. We consider the only non trivial case: $(E'|F') \setminus Act_H \stackrel{\tau}{\rightarrow} (E''|F'') \setminus Act_H$ with $E' \stackrel{h}{\rightarrow} E''$ and $F' \stackrel{\bar{h}}{\rightarrow} F''$. Since $E'/Act_H \stackrel{\tau}{\rightarrow} E''/Act_H$ and, by hypothesis, $E' \in BSNNI$, we have that $\exists E'''$ such that $E' \setminus Act_H \stackrel{\hat{\tau}}{\Longrightarrow} E''' \setminus Act_H$ and $E''/Act_H \approx_B E''' \setminus Act_H$; finally, by hypothesis, $E'' \in BSNNI$, hence we obtain $E''' \setminus Act_H \approx_B E'' \setminus Act_H$. Repeating the same procedure for F' we have $\exists E''', F'''$ such that $E' \setminus Act_H|F' \setminus Act_H \stackrel{\hat{\tau}}{\Longrightarrow} E''' \setminus Act_H|F''' \setminus Act_H \approx_B E'' \setminus Act_H|F'' \setminus Act_H$. Since $((E''|F'') \setminus Act_H, E'' \setminus Act_H|\ F'' \setminus Act_H) \in R$, then R is a bisimulation up to $\approx_B$.

(ii) Consider the following relation $((E'/Act_H) \setminus L,\ (E' \setminus L)/Act_H) \in R$, for all E' such that $E \Rightarrow E'$ and for all $L \subseteq \mathcal{L}$. If we prove that R is a bisimulation up to $\approx_B$ then, by applying hypothesis and observing that $(E' \setminus L) \setminus Act_H \approx_B (E' \setminus Act_H) \setminus L$, we obtain the thesis. The only non trivial case is $(E'/Act_H) \setminus L \stackrel{\tau}{\rightarrow} (E''/Act_H) \setminus L$ with $E' \stackrel{h}{\rightarrow} E''$ and $h \in Act_H$. By hypothesis, $E' \in BSNNI$ hence we have that $(E'/Act_H) \setminus L \approx_B (E' \setminus Act_H) \setminus L$ and so $\exists E'''$ such that $(E' \setminus Act_H) \setminus L \stackrel{\hat{\tau}}{\Longrightarrow} (E''' \setminus Act_H) \setminus L \approx_B (E'''/Act_H) \setminus L$ and $(E''' \setminus Act_H) \setminus L \approx_B (E''/Act_H) \setminus L$. Obviously, we also have that $(E' \setminus L) \setminus Act_H \stackrel{\hat{\tau}}{\Longrightarrow} (E''' \setminus L) \setminus Act_H$ and so $(E' \setminus L)/Act_H \stackrel{\hat{\tau}}{\Longrightarrow} (E''' \setminus L)/Act_H$. We briefly summarize the proof: we had the synchronization $(E'/Act_H) \setminus L \stackrel{\tau}{\rightarrow} (E''/Act_H) \setminus L$ and we proved that there exists E''' such that $(E' \setminus L)/Act_H \stackrel{\hat{\tau}}{\Longrightarrow} (E''' \setminus L)/Act_H$ and $(E'''/Act_H) \setminus L \approx_B (E''/Act_H) \setminus L$. Since $((E'''/Act_H) \setminus L, (E''' \setminus L)/Act_H) \in R$ then R is a weak bisimulation up to $\approx_B$. ■

It is worthwhile noticing that *SBSNNI* was not the first sufficient condition proposed for *BNDC*. In [24] we introduced a property stronger than *SBSNNI*, but nevertheless quite intuitive, called *Strong BNDC* (*SBNDC*). This property just requires that before and after every high step, the system appears to be the same, from a low level perspective. More formally we have the following definition.

Definition 14. *(SBNDC: Strong BNDC)*
A system $E \in SBNDC$ if and only if $\forall E'$ reachable from E and $\forall E''$ such that $E' \stackrel{h}{\rightarrow} E''$ for $h \in Act_H$, then $E' \setminus Act_H \approx_B E'' \setminus Act_H$

We now prove that *SBNDC* is strictly stronger than *SBSNNI*. To this purpose, we need the following Lemma

Lemma 2. *$E \in BNDC \Leftrightarrow E \setminus Act_H \approx_B (E|\Pi) \setminus Act_H$ for all $\Pi \in \mathcal{E}_H$.*

Proof. It follows immediately from Theorem 1.(ii) and Definition 12. ■

Proposition 7. *SBNDC* $\subset$ *SBSNNI.*

PROOF. Let E be a system and Π a high level process. Let R be a relation defined as follows: $(E' \setminus Act_H, (E' \mid \Pi') \setminus Act_H) \in R$, for all E', Π' such that $E \Rightarrow E'$ and $\Pi \Rightarrow \Pi'$. We want to prove that if E is *SBNDC* then R is a bisimulation up to $\approx_B$ (see [50]). There is only one interesting case: $(E' \mid \Pi')\setminus Act_H \xrightarrow{\tau} (E'' \mid \Pi'')\setminus Act_H$. As there exists $h \in Act_H$ such that $E' \xrightarrow{h} E''$, then $E'' \setminus Act_H \approx_B E' \setminus Act_H$; since $(E'' \setminus Act_H, (E'' \mid \Pi'') \setminus Act_H) \in R$, then $(E'' \mid \Pi'') \setminus Act_H \approx_B E' \setminus Act_H$. So $E' \setminus Act_H \approx_B (E' \mid \Pi) \setminus Act_H$ (i.e., $E' \in BNDC$), for all E' reachable from E. As *BNDC* is stronger than *BSNNI* (Theorem 1.(ii)), we obtain the thesis. The inclusion is strict because agent $E = \tau.l.0 + l.l.0 + h.l.0$ is *SBSNNI* but not *SBNDC*. ∎

As for *SBSNNI*, we have a compositionality theorem:

Theorem 4. *The following hold:*

(i) $E, F \in SBNDC \Longrightarrow (E|F) \in SBNDC,$
(ii) $E \in SBNDC \Rightarrow E \setminus S \in SBNDC$, *if* $S \subseteq \mathcal{L}$.
(iii) $E, F \in SBNDC \Longrightarrow (E||F) \in SBNDC$

PROOF. (i) It must be proved that $\forall E', F' : E \Rightarrow E', F \Rightarrow F', \forall E'', F'' : (E' \mid F') \xrightarrow{h} (E'' \mid F'')$ with $h \in Act_H$, the following holds: $(E' \mid F') \setminus Act_h \approx_B (E'' \mid F'') \setminus Act_H$. Let R be the relation defined as follows: $((\tilde{E} \mid \tilde{F}) \setminus Act_H, (\tilde{E}' \mid \tilde{F}') \setminus Act_H) \in R, \forall \tilde{E}, \tilde{E}', \tilde{F}, \tilde{F}'$ such that $E \Rightarrow \tilde{E}, E \Rightarrow \tilde{E}', F \Rightarrow \tilde{F}, F \Rightarrow \tilde{F}'$ and $\tilde{E} \setminus Act_H \approx_B \tilde{E}' \setminus Act_H, \tilde{F} \setminus Act_h \approx_B \tilde{F}' \setminus Act_H$. It can be easily proved that R is a weak bisimulation (under the hypothesis $E, F \in SBNDC$); there is only one interesting case: $(\tilde{E} \mid \tilde{F}) \setminus Act_H \xrightarrow{\tau} (\tilde{E}'' \mid \tilde{F}'') \setminus Act_H$. Thus we have that $\tilde{E} \setminus Act_H \approx_B \tilde{E}'' \setminus Act_H$ and $\tilde{F} \setminus Act_H \approx_B \tilde{F}'' \setminus Act_H$ and so $((\tilde{E}'' \mid \tilde{F}'') \setminus Act_H, (\tilde{E}' \mid \tilde{F}') \setminus Act_H) \in R$.
(ii) Observe that $(E' \setminus S)\setminus Act_H \approx_B (E'' \setminus S)\setminus Act_H$ if and only if $(E' \setminus Act_H)\setminus S \approx_B (E'' \setminus Act_H) \setminus S$. As $\approx_B$ is a congruence for restriction [50] the thesis follows trivially.
(iii) Trivial from (i) and (ii). ∎

Figure 11 summarizes the relations among all the trace-based and bisimulation-based properties we have presented.

We end this section with a remark. In the automatic verification of properties, it can be very useful to work on a *reduced* system, i.e., a system equivalent to the original one, but with a smaller number of states. In fact, the tool we will present in the next section provides a procedure that minimizes the number of states, thus reducing a lot the time spent in the verification. In order to see if this proof strategy can be used also in our case, we need to prove that if a system E is *BNDC*, then any other observation equivalent system F is *BNDC* too. Indeed, the theorem below shows that this is the case, also for all the other security properties we have discussed in this section.

Theorem 5. *If* $E \approx_B F$*, then* $E \in X \Leftrightarrow F \in X$*, where* X *can be NNI, SNNI, NDC, BNNI, BSNNI, BNDC, SBSNNI, SBNDC.*

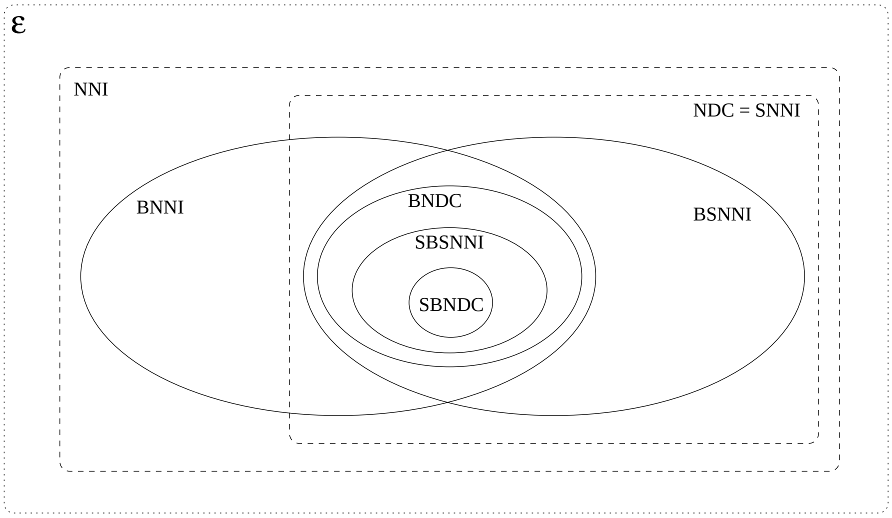


Fig. 11. The inclusion diagram for trace-based and bisimulation-based properties.

PROOF. It derives from the definition of the security properties, observing that trace and bisimulation equivalences are congruences with respect to $\_ \setminus L$ and $\_|\_$ operators of CCS [50]. It is possible to prove that they are also congruence with with respect to $\_ \setminus_I L$ and $\_/L$ operators of SPA. For trace equivalence the proof is trivial. In the following we prove the closure of weak bisimulation equivalence w.r.t. $\_ \setminus_I L$ and leave to the reader the other similar case of $\_/L$. Let $E \approx_B F$; we want to prove that $E \setminus_I L \approx_B F \setminus_I L$. Consider a bisimulation R such that $(E, F) \in R$; then define the relation P as follows: $(E' \setminus_I L, F' \setminus_I L) \in P$ if and only if $(E', F') \in R$. P is a bisimulation too, in fact if $E' \setminus_I L \xrightarrow{\mu} E'' \setminus_I L$ then $\mu \notin L \cap I$ and $E' \xrightarrow{\mu} E''$. So $\exists F''$ such that $F' \stackrel{\hat{\mu}}{\Longrightarrow} F''$; since $\mu \notin L \cap I$ then $F' \setminus_I L \stackrel{\hat{\mu}}{\Longrightarrow} F'' \setminus_I L$ (and vice versa for $F' \setminus_I L \xrightarrow{\mu} F'' \setminus_I L$). ∎

3.4 Building Channels by Exploiting Deadlocks

In Example 6 we have seen that *Access_Monitor_1* is not *BNDC* because of potential high level deadlocks. We said that a deadlock due to high level activity is visible from low level users, hence it gives some information about high level actions, and cannot be allowed. However, one could doubt that a high level deadlock is really dangerous and, in particular, that it can be exploited to transmit information from high to low. We demonstrate that it is indeed the case by simply showing that it is possible to build a 1-bit channel from high to low level using systems which contain high level deadlocks. In particular we obtain a 1-bit channel with some initial noise (before the beginning of the transmission), using three processes with high level deadlocks composed with other secure systems.

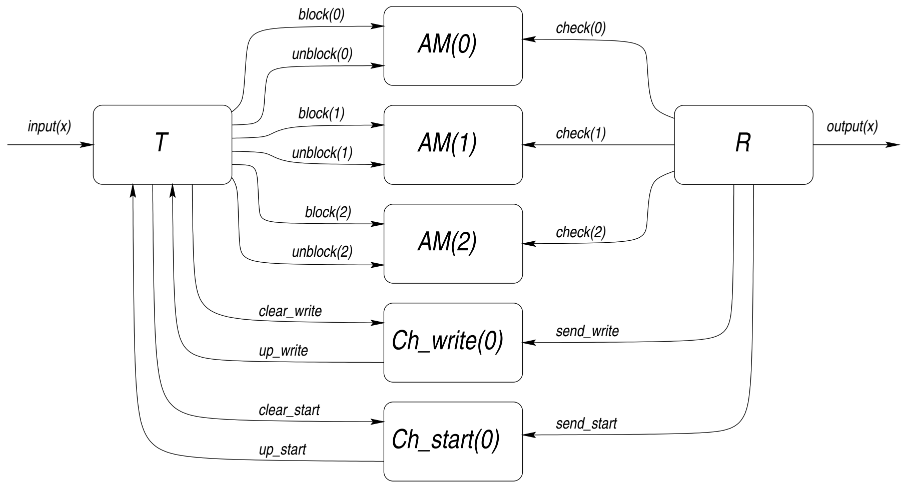


Fig. 12. The *Channel* process, a one-bit channel obtained by three processes with high level deadlocks.

Of the two high level deadlocks of process $Access\_Monitor\_1$ we only exploit the one due to write requests. So the following method can be applied also to systems with only one high level deadlock. Process $Channel$ is reported in Table 10 where $n \in \{0, 1, 2\}$, $x, y \in \{0, 1\}$; $\{block$, $unblock$, up_write, up_start, $input$, $clear\_write$, $clear\_start\} \subseteq Act_H$; $N = \{check$, $block$, $unblock$, $send\_write$, up_write, $clear\_write$, $send\_start$, up_start, $clear\_start\}$.

$Channel$ (see Figure 12) is the composition of three instances of Access Monitor ($AM(0)$,$AM(1)$ and $AM(2)$), two channels from low to high level ($Ch\_write(0)$ and $Ch\_start(0)$), a transmitter and a receiver (T and R). In particular $AM(n)$ is an instance of *Access_Monitor_1* where we call *check*(n) the reading request of low level users in low object, it is used to check if $AM(n)$ is in a deadlock state; $block(n)$ is a writing request by a high level user, it is used to block $AM(n)$; finally $unblock(n)$ is a write action and is used to unblock $AM(n)$ which was previously blocked with $block(n)$. $Ch\_write(x)$ moves to $Ch\_write(1)$ every time a $send\_write$ is executed by the receiver. $Ch\_write(1)$ can give to T the $\overline{up\_write}$ signal; it also ignores all the $send\_write$ signals. Moreover, when $clear\_write$ is executed by the transmitter this resets the process to $Ch\_write(0)$. So if R executes $\overline{send\_write}$ and after this, T executes $\overline{clear\_write}$ and up_write, then T will be blocked. $Ch\_start(0)$ is equal to $Ch\_write(0)$ with appropriate action relabeling.

R and T use the monitor $AM(2)$ for synchronization and $AM(0)$, $AM(1)$ for transmission. In particular T blocks $AM(2)$ and then it waits for a write enable signal (up_write). Afterward it blocks also monitors $AM(0)$ and $AM(1)$ moving to $T1$ which is the writing loop; $T1$ unblocks monitor $AM(2)$—which is the signal for R to start receiving the bit—and waits for a start writing signal

$$
\begin{aligned}
Channel &\stackrel{\text{def}}{=} (Ch\_write(0) \mid Ch\_start(0) \mid AM(0) \mid AM(1) \mid AM(2) \mid \\
&\qquad \mid R \mid T) \setminus N \\
AM(n) &\stackrel{\text{def}}{=} Access\_Monitor\_1[check(n)/access\_r(0,0), \\
&\qquad block(n)/access\_w(1,1), \\
&\qquad unblock(n)/write(1,0)] \setminus \{access\_r, access\_w, write, val\} \\
Ch\_write(x) &\stackrel{\text{def}}{=} send\_write.Ch\_write(1) \\
&\quad + \\
&\quad clear\_write.Ch\_write(0) \\
&\quad + \\
&\quad \ \mathbf{if}\ x = 1\ \mathbf{then} \\
&\qquad \overline{up\_write}.Ch\_write(0) \\
Ch\_start(0) &\stackrel{\text{def}}{=} Ch\_write(0)[send\_start/send\_write, up\_start/up\_write, \\
&\qquad clear\_start/clear\_write] \\
R &\stackrel{\text{def}}{=} \overline{send\_write}.R \\
&\quad + \\
&\quad \overline{check}(2).\overline{send\_start}. \\
&\qquad (\overline{check}(0).\overline{output}(0).R \\
&\qquad + \\
&\qquad \overline{check}(1).\overline{output}(1).R) \\
T &\stackrel{\text{def}}{=} \overline{block}(2).\overline{clear\_write}.up\_write.\overline{block}(0).\overline{block}(1).T1 \\
T1 &\stackrel{\text{def}}{=} \overline{clear\_start}.\overline{unblock}(2).up\_start.\overline{block}(2).\overline{clear\_write}. \\
&\quad input(y).\overline{unblock}(y).up\_write.\overline{block}(y).T1
\end{aligned}
$$

Table 10. The *Channel* process which exploits deadlocks

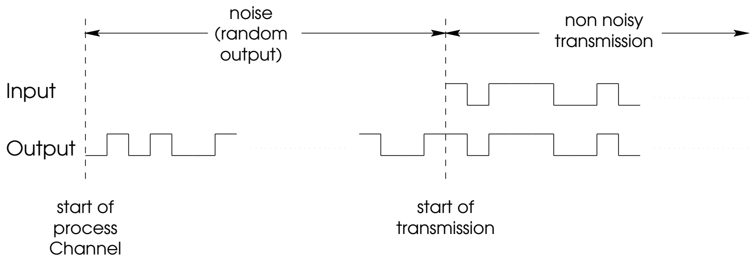


Fig. 13. Noise in *Channel* and C

(up_start). Then it blocks $AM(2)$ again, reads the high level value y, unblocks monitor $AM(y)$ (transmitting the value y) and waits for a write enable signal. When it receives such a signal, it blocks again monitor $AM(y)$ and moves to $T1$ in order to transmit another bit.

R can always send a writing enable signal moving again to R. Moreover it can check if $AM(2)$ is blocked. If such a monitor is not blocked (by T) it sends a start writing signal and checks if $AM(0)$ and $AM(1)$ are blocked. If it discovers that $AM(t)$, with $t = 0$ or $t = 1$, is not blocked, then it gives as output $\overline{output}(t)$. Finally it moves again to R in order to receive the next bit.

Note that if R executes $\overline{check}(2)$ before T has blocked monitor $AM(2)$ then R will give a non-deterministic output $\overline{output}(x)$. In fact T and R will synchronize and start transmitting the 1-bit messages as soon as T will execute $\overline{block}(2)$ blocking $AM(2)$. So we have some random output before the beginning of the transmission. It is possible to automatically check that

$$Channel \approx_B C$$

where

$$\begin{aligned}
C &\stackrel{\text{def}}{=} \tau.C' \\
&\quad + \tau.(\\
&\qquad \tau.(\tau.\overline{output}(0).C' + \tau.\overline{output}(1).C') \\
&\qquad + \tau.(\overline{output}(0).C + \tau.\overline{output}(0).C') \\
&\qquad + \tau.(\overline{output}(1).C + \tau.\overline{output}(1).C') \\
&\quad) \\
C' &\stackrel{\text{def}}{=} input(x).\overline{output}(x).C'
\end{aligned}$$

This automatic check can be done using the CoSeC tool we will present in the next section.

C can move to a 1-bit channel C' or can give some non-deterministic output (initial noise before the synchronization between T and R). Note that after moving to C' (the channel is ready to transmit) it will behave as a perfect 1-bit channel (see Figure 13).

3.5 Comparison with Related Work

The use of process algebras to formalize information flow security properties is not new. In [57] it is possible to find a definition of Non Interference given on CSP [39]. It looks like *SNNI* with some side conditions on acceptable low level actions. This definition is recalled in [4], where a comparison with another information flow property is reported.

More recent results based on the CSP model are contained in [56], where the authors introduce some information flow security properties based on the notion of *deterministic views* and show how to automatically verify them using the CSP model checker FDR [55].

The most interesting property is *lazy security* (*L-Sec*) which, however, requires the absence of non-determinism in the low view of the system (i.e., when hiding high actions through interleaving) and for this reason we think it could be too restrictive in a concurrent environment. For example, all the low non-deterministic systems – such as $E = l.\bar{l_1} + l.\bar{l_2}$ – are considered not secure. In this section we compare those properties with ours using a failure-equivalence version of *BNDC*, called *FNDC* (see also [18] for more details). The main result is that *BNDC* restricted to the class of low-deterministic and non-divergent processes is equal to *L-Sec*.

Here we give a definition of failure equivalence which does not correspond completely to the original one [39]. Indeed, it does not consider possible divergences but this is not a problem since our comparison will focus on the class of non-divergent processes. We prefer this definition because it is very simple and is implied by $\approx_B$.

We need some simple additional notations. We write $E \stackrel{\mu}{\not\Rightarrow}$ to indicate that $\nexists E'$ such that $E \stackrel{\mu}{\Longrightarrow} E'$ and $E \stackrel{K}{\not\Rightarrow}$ with $K \subseteq \mathcal{L}$ stands for $\forall \mu \in K, E \stackrel{\mu}{\not\Rightarrow}$.

Definition 15. *If $\gamma \in T(E)$ and if, after executing γ, E can refuse all the actions in set $X \subseteq \mathcal{L}$, then we say that the pair (γ, X) is a failure of the process E. Formally we have that:*

$$\mathit{failures}(E) \stackrel{\text{def}}{=} \{(\gamma, X) \subseteq \mathcal{L}^* \times \mathbb{P}(\mathcal{L}) \mid \exists E' \text{ such that } E \stackrel{\gamma}{\Longrightarrow} E' \text{ and } E' \stackrel{X}{\not\Rightarrow}\}$$

When $\mathit{failures}(E) = \mathit{failures}(F)$ we write $E \approx_F F$ (failure equivalence). ■

We identify a process E with its failure set. So if $(\gamma, X) \in \mathit{failures}(E)$ we write $(\gamma, X) \in E$. Note that $\gamma \in T(E)$ if and only if $(\gamma, \emptyset) \in E$. So $E \approx_F F$ implies $E \approx_T F$.

We also have that $E \approx_B F$ implies $E \approx_F F$:

Proposition 8. *$E \approx_B F$ implies $E \approx_F F$.*

Proof. Consider $E \approx_B F$ and $(\gamma, X) \in E$. We want to prove that $(\gamma, X) \in F$. Since $(\gamma, X) \in E$ we have that $\exists E'$ such that $E \stackrel{\gamma}{\Longrightarrow} E'$ and $E' \stackrel{X}{\not\Rightarrow}$. By definition of $\approx_B$ we know that since $E \stackrel{\gamma}{\Longrightarrow} E'$ then $F \stackrel{\gamma}{\Longrightarrow} F'$ and $E' \approx_B F'$ (it is sufficient to simulate every step of the execution $E \stackrel{\gamma}{\Longrightarrow} E'$). Suppose by contradiction that $\exists \mu \in X$ such that $F' \stackrel{\mu}{\Longrightarrow} F''$. Then, by definition of $\approx_B$ and since $E' \approx_B F'$, we obtain that $E' \stackrel{\mu}{\Longrightarrow} E''$ but this is not possible since $E' \stackrel{X}{\not\Longrightarrow}$. We obtain that $F' \stackrel{X}{\not\Longrightarrow}$ and so $(\gamma, X) \in F$. The symmetric case can be done for every $(\gamma, X) \in F$ which must belong also to E. ■

Lazy Security We now report the *lazy security* property [56] and we show that it can only deal with low-deterministic processes, i.e., processes which have a deterministic behaviour with respect to low level actions. Here we do not consider the *eager security* property (introduced in [56] to deal with output actions) since it supposes that high level actions happen instantaneously while in SPA, which has synchronous communications, both input and output actions can be delayed by users. We start with a formal definition of determinism.

Definition 16. *E is deterministic ($E \in Det$) if and only if whenever $\gamma a \in traces(E)$ then $(\gamma, \{a\}) \notin E$.* ■

So a process is deterministic if after every trace γ it cannot both accept and refuse a certain action a. We give another characterization for determinism. A system E is deterministic if and only if whenever it can move to two different processes E' and E'' executing a certain trace γ, such processes are failure equivalent.

Proposition 9. *$E \in Det$ if and only if for all $\gamma \in traces(E)$ we have that $E \stackrel{\gamma}{\Longrightarrow} E'$, $E \stackrel{\gamma}{\Longrightarrow} E''$ implies $E' \approx_F E''$.*

Proof. ($\Rightarrow$) Let $E \in Det$, $E \stackrel{\gamma}{\Longrightarrow} E'$, $E \stackrel{\gamma}{\Longrightarrow} E''$ and $(\delta, K) \in E'$. We want to prove that $(\delta, K) \in E''$. Since $E \stackrel{\gamma}{\Longrightarrow} E'$, we have that $(\gamma\delta, K) \in E$. By $E \in Det$ we obtain that $\forall a \in K, \gamma\delta a$ is not a trace for E. We also have that δ is a trace for E''; in fact, if E'' can execute only a prefix of δ, i.e. $E'' \stackrel{\alpha}{\Longrightarrow} E'''$ with $\delta = \alpha b \beta$, we have that E can execute trace $\gamma\alpha b$ (through E') and can refuse b after $\gamma\alpha$ (through E'') contradicting the determinism hypothesis. Now, since $\forall a \in K, \gamma\delta a \notin traces(E)$, we also have that $\forall a \in K, \delta a \notin traces(E'')$ and so $(\delta, K) \in E''$.
($\Leftarrow$) Trivial. ■

Corollary 1. *If $E \stackrel{\gamma}{\Longrightarrow} E'$ and $E \in Det$ then $E' \in Det$.*

Proof. We have to prove that $E' \stackrel{\delta}{\Longrightarrow} E''$ and $E' \stackrel{\delta}{\Longrightarrow} E'''$ implies $E'' \approx_F E'''$. Consider $E \stackrel{\gamma\delta}{\Longrightarrow} E''$ and $E \stackrel{\gamma\delta}{\Longrightarrow} E'''$ then by $E \in Det$ we have that $E'' \approx_F E'''$. ■

In the following we denote with $E|||F$ the interleaving without communication between agents E and F. It can be expressed in SPA as $(E[A/\mathcal{L}(E)] \mid F[B/\mathcal{L}(F)])[\mathcal{L}(E)/A, \mathcal{L}(F)/B]$ where $A, B \subseteq \mathcal{L}, A \cap B = \emptyset$ and $A/\mathcal{L}(E)$ is a bijective function which maps all the actions executable by E into actions of

A, with $\mathcal{L}(E)/A$ as inverse (the same holds for $B/\mathcal{L}(F)$ and $\mathcal{L}(F)/B$). This expression means that the actions in E and F are first relabelled using the two disjoint sets A and B, then interleaved (no communication is possible) and finally renamed to their original labels.

Recall that a process is *divergent* if it can execute an infinite sequence of internal actions τ. As an example consider the agent $A \stackrel{\text{def}}{=} \tau.A + b.\underline{0}$ which can execute an arbitrary number of τ actions. We define *Nondiv* as the set of all the non-divergent processes.

We can now present the *lazy security* property [56]. This property implies that the obscuring of high level actions by interleaving does not introduce any non-determinism. The obscuring of high level actions of process E by interleaving is obtained considering process $E|||RUN_H$ where $RUN_H \stackrel{\text{def}}{=} \sum_{h \in Act_H} h.RUN_H$. In such a process an outside observer is not able to tell if a certain high level action comes from E or from RUN_H.

L-Sec also requires that $E|||RUN_H$ is non-divergent. [9] This is equivalent to requiring that E is non-divergent, because RUN_H is non-divergent and the $|||$ operator does not allow synchronizations (which could generate new τ actions).

Definition 17. *$E \in$ L-Sec $\Leftrightarrow E|||RUN_H \in$ Det $\cap$ Nondiv.* ■

In the following we want to show that *L-Sec* can only analyze systems which are *low-deterministic*, i.e., where after any low level trace γ no low level action l can be both accepted and refused. The low-determinism requirement is not strictly necessary to avoid information flows from high to low level. So, in some cases, *L-Sec* is too strong. As an example consider the following non-deterministic system without high level actions: $E \stackrel{\text{def}}{=} l.l'.\underline{0} + l.l''.\underline{0}$. It is obviously secure but it is not low-deterministic and so it is not *L-Sec*. Formally we have that:

Definition 18. *E is low-deterministic ($E \in$ Lowdet) iff $E \setminus Act_H \in$ Det.* ■

The following holds:

Theorem 6. *L-Sec $\subseteq$ Lowdet.*

Proof. Let $E \in$ *L-Sec*. Consider a trace γa of $E \setminus Act_H$ and suppose that $(\gamma, \{a\}) \in E \setminus Act_H$. So there exists E' such that $E \setminus Act_H \stackrel{\gamma}{\Longrightarrow} E' \setminus Act_H$ and such that $E' \setminus Act_H \stackrel{a}{\not\Rightarrow}$. Since RUN_H cannot execute the low level action a then we have that $E'|||RUN_H \stackrel{a}{\not\Rightarrow}$ and so $(\gamma, \{a\}) \in E|||RUN_H$ because $E|||RUN_H \stackrel{\gamma}{\Longrightarrow} E'|||RUN_H$. Since γa is a trace for $E \setminus Act_H$ then it is also a trace for $E|||RUN_H$ and we obtain that $E|||RUN_H$ is not deterministic, contradicting the hypothesis. So $(\gamma, \{a\}) \notin E \setminus Act_H$ and $E \in$ *Lowdet*. ■

[9] Note that in [56] the non-divergence requirement is inside the deterministic one. This is because the authors use the failure-divergence semantics [10]. In this work we use the failure equivalence which does not deal with divergences. So, in order to obtain exactly the *L-Sec* property, we require the non-divergence condition explicitly.

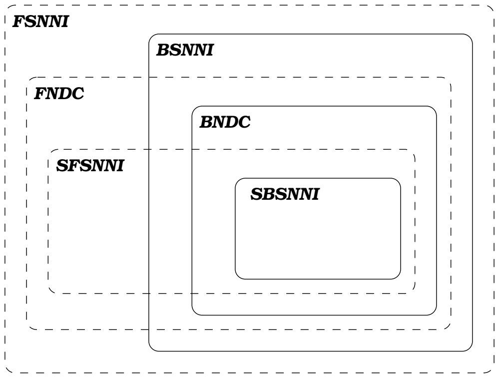


Fig. 14. Failure based and bisimulation based properties.

Failure Non Deducibility on Compositions Now we define the failure based security properties by simply substituting $\approx_B$ with $\approx_F$ in all the bisimulation based properties previously defined.

Definition 19. *(Failure based properties)*

(i) $E \in \mathit{FNDC} \Leftrightarrow E/\mathit{Act}_H \approx_F (E \mid \Pi) \setminus \mathit{Act}_H$, *for all* $\Pi \in \mathcal{E}_H$*;*
(ii) $E \in \mathit{FSNNI} \Leftrightarrow E/\mathit{Act}_H \approx_F E \setminus \mathit{Act}_H$*;*
(iii) $E \in \mathit{SFSNNI} \Leftrightarrow \forall E'$ *such that* $\exists \gamma : E \stackrel{\gamma}{\Longrightarrow} E'$ *we have* $E' \in \mathit{FSNNI}$. ■

Since bisimulation equivalence is stronger than failure equivalence, it can be proved that each of these new property is weaker then its corresponding bisimulation based one. E.g. $\mathit{BNDC} \subset \mathit{FNDC}$. Moreover we prove that some of the inclusion results we have for bisimulation based properties can be extended also to these new properties.

Theorem 7. *SFSNNI* $\subset$ *FNDC* $\subset$ *FSNNI.*

Proof. ($\mathit{SFSNNI} \subset \mathit{FNDC}$) Let E be a *SFSNNI* process. We have to prove that $(E|\Pi) \setminus \mathit{Act}_H \approx_F E/\mathit{Act}_H$ for every high level process Π.
We first prove that $(\gamma, K) \in (E|\Pi) \setminus \mathit{Act}_H$ implies $(\gamma, K) \in E/\mathit{Act}_H$. Consider $(\gamma, K) \in (E|\Pi) \setminus \mathit{Act}_H$, then $\exists E', \Pi'$ such that $(E|\Pi) \setminus \mathit{Act}_H \stackrel{\gamma}{\Longrightarrow} (E'|\Pi') \setminus \mathit{Act}_H \stackrel{K}{\not\Rightarrow}$. Hence $E' \setminus \mathit{Act}_H \stackrel{K}{\not\Rightarrow}$ because $\mathit{traces}(E' \setminus \mathit{Act}_H) \subseteq \mathit{traces}((E'|\Pi') \setminus \mathit{Act}_H)$. Now, since $E \in \mathit{SFSNNI}$ then $E' \setminus \mathit{Act}_H \approx_F E'/\mathit{Act}_H$; hence $E'/\mathit{Act}_H \stackrel{K}{\not\Rightarrow}$. Note that $E/\mathit{Act}_H \stackrel{\gamma}{\Longrightarrow} E'/\mathit{Act}_H$, hence $(\gamma, K) \in E/\mathit{Act}_H$.
We now prove that $(\gamma, K) \in E/\mathit{Act}_H$ implies $(\gamma, K) \in (E|\Pi) \setminus \mathit{Act}_H$. Consider $(\gamma, K) \in E/\mathit{Act}_H$. By hypothesis we have that $(\gamma, K) \in E \setminus \mathit{Act}_H$ and so $\exists E'$ such that $E \setminus \mathit{Act}_H \stackrel{\gamma}{\Longrightarrow} E' \setminus \mathit{Act}_H \stackrel{K}{\not\Rightarrow}$. Since $E \in$*SFSNNI* then $E'/\mathit{Act}_H \stackrel{K}{\not\Rightarrow}$. Hence we also have that $(E'|\Pi) \setminus \mathit{Act}_H \stackrel{K}{\not\Rightarrow}$ because $\mathit{traces}((E'|\Pi) \setminus \mathit{Act}_H) \subseteq$

traces(E'/Act_H). Since we have that $E \setminus Act_H \stackrel{\gamma}{\Longrightarrow} E' \setminus Act_H$ then $(E|\Pi) \setminus Act_H \stackrel{\gamma}{\Longrightarrow} (E'|\Pi) \setminus Act_H$ and so $(\gamma, K) \in (E|\Pi) \setminus Act_H$.

The inclusion is strict because agent $E \stackrel{\text{def}}{=} l.h.l.\underline{0} + l.\underline{0} + l.l.\underline{0}$ is *FNDC* but not *SFSNNI*.

(*FNDC* $\subset$ *FSNNI*) It is sufficient to consider $\Pi = \underline{0}$. We have that $(E|\underline{0}) \setminus Act_H \approx_F E \setminus Act_H$ and so, since $(E|\underline{0}) \setminus Act_H \approx_F E/Act_H$ we have $E/Act_H \approx_F E \setminus Act_H$.

The inclusion is strict because agent $E \stackrel{\text{def}}{=} l.h.l.h'.l.\underline{0} + l.\underline{0} + l.l.l.\underline{0}$ is *FSNNI* but not *FNDC*. ■

Figure 14 summarizes the inclusions among the presented security properties. It can be drawn using the previous inclusion results and the following remarks: *BNDC* $\not\subseteq$ *SFSNNI*, in fact agent $l.h.l.\underline{0} + l.\underline{0} + l.l.\underline{0}$ is *BNDC* but not *SFSNNI*; we also have that *BSNNI* $\not\subseteq$ *FNDC* because of agent $h.l.h'.l.\underline{0} + l.l.\underline{0}$; finally *SFSNNI* $\not\subseteq$ *BSNNI* because of agent $h.l.(l'.\underline{0} + l''.\underline{0}) + l.l'.\underline{0} + l.l''.\underline{0}$.

The next theorem shows that under the low-determinism assumption the properties *SFSNNI* and *FNDC* collapse into the same one. We need the following Lemma.

Lemma 3. *If* $E, \tilde{E} \in Det$, $E \stackrel{\gamma}{\Longrightarrow} E'$, $\tilde{E} \stackrel{\gamma}{\Longrightarrow} \tilde{E}'$ *and* $E \approx_F \tilde{E}$ *then* $E' \approx_F \tilde{E}'$.

Proof. We prove that if $(\delta, K) \in E'$ then $(\delta, K) \in \tilde{E}'$. Let $(\delta, K) \in E'$. Then $(\gamma\delta, K) \in E$ and by $E \approx_F \tilde{E}$ we obtain that $(\gamma\delta, K) \in \tilde{E}$. So $\exists \tilde{E}'', \tilde{E}'''$ such that $\tilde{E} \stackrel{\gamma}{\Longrightarrow} \tilde{E}'' \stackrel{\delta}{\Longrightarrow} \tilde{E}''' \stackrel{K}{\not\Longrightarrow}$, hence $(\delta, K) \in \tilde{E}''$. Since $\tilde{E} \in Det$ then by Proposition 9 and hypothesis we have that $\tilde{E}'' \approx_F \tilde{E}'$ and so $(\delta, K) \in \tilde{E}'$. We can prove in the same way that if $(\delta, K) \in \tilde{E}'$ then $(\delta, K) \in E'$. So $E' \approx_F \tilde{E}'$ ■

Theorem 8. *FNDC* $\cap$ *Lowdet* $\subseteq$ *SFSNNI.*

Proof. Since *FNDC* $\subset$ *FSNNI* and $E \in$ *FNDC*, we have that $E \setminus Act_H \approx_F E/Act_H$. By $E \in$ *Lowdet* we obtain $E/Act_H \in Det$. Now consider $E \stackrel{\gamma}{\Longrightarrow} E'$. We have to prove that $E'/Act_H \approx_F E' \setminus Act_H$. Let Π' be the high level process which executes exactly the complement of the high level projection of γ, i.e. the complement of the subsequence of γ composed by all the high level actions in γ. If γ' is the low level projection of γ we have that $(E|\Pi') \setminus Act_H \stackrel{\gamma'}{\Longrightarrow} (E'|\underline{0}) \setminus Act_H \approx_F E' \setminus Act_H$. Since $E \stackrel{\gamma}{\Longrightarrow} E'$ then $E/Act_H \stackrel{\gamma'}{\Longrightarrow} E'/Act_H$. By hypothesis we have that $(E|\Pi') \setminus Act_H \approx_F E/Act_H$. Since $E/Act_H \in Det$ then, by Lemma 3, we have that $E'/Act_H \approx_F (E'|\underline{0}) \setminus Act_H \approx_F E' \setminus Act_H$. ■

Corollary 2. *FNDC* $\cap$ *Lowdet* $=$ *SFSNNI* $\cap$ *Lowdet*.

Proof. Trivial by Theorems 8 and 7. ■

Comparison We now show that under the low-determinism and the non-divergence assumption the *BNDC* property is equal to *L-Sec*. We start proving this result for *FNDC*.

Theorem 9. *L-Sec* $\subseteq$ *SFSNNI.*

Proof. Let $E \in$ *L-Sec*. Then we have to prove that if $E \stackrel{\gamma}{\Longrightarrow} E'$ then $E' \setminus Act_H \approx_F E'/Act_H$. We first prove that if $(\delta, K) \in E'/Act_H$ then $(\delta, K) \in E' \setminus Act_H$. Consider $(\delta, K) \in E'/Act_H$. Then we have that $\exists E''$ such that $E'/Act_H \stackrel{\delta}{\Longrightarrow} E''/Act_H \stackrel{K}{\not\Rightarrow}$.

Now we want to prove that δ is a trace also for $E' \setminus Act_H$. Let $\delta = \delta_1 \delta_2 \dots \delta_n$ and consider the execution $E'/Act_H \stackrel{\delta_1}{\Longrightarrow} E'_1/Act_H \stackrel{\delta_2}{\Longrightarrow} \dots \stackrel{\delta_n}{\Longrightarrow} E''/Act_H$. Suppose that δ_i is the first action in δ that $E' \setminus Act_H$ is not able to execute. In other words we have that

$$E' \setminus Act_H \stackrel{\delta_1}{\Longrightarrow} E'_1 \setminus Act_H \stackrel{\delta_2}{\Longrightarrow} \dots \stackrel{\delta_{i-1}}{\Longrightarrow} E'_{i-1} \setminus Act_H \stackrel{\delta_i}{\not\Rightarrow}$$

This means that in order to execute δ_i, process E'_{i-1}/Act_H executes some hidden high level actions $h_1 \dots h_k$. So $E'_{i-1} \stackrel{h_1 \dots h_k \delta_i}{\Longrightarrow} E'_i$. If we execute such high level actions with RUN_H we obtain that $E|||RUN_H \stackrel{\gamma\delta_1 \dots \delta_{i-1} h_1 \dots h_k}{\Longrightarrow} E'_{i-1}|||RUN_H$. Since $E'_{i-1} \setminus Act_H \stackrel{\delta_i}{\not\Rightarrow}$ and $\delta_i \in Act_L$ then we obtain that $(\gamma\delta_1 \dots \delta_{i-1} h_1 \dots h_k, \{\delta_i\}) \in E|||RUN_H$. Moreover, if we execute actions $h_1 \dots h_k$ with E'_{i-1} we have that $E|||RUN_H \stackrel{\gamma\delta_1 \dots \delta_{i-1} h_1 \dots h_k \delta_i}{\Longrightarrow} E'_i|||RUN_H$ and so $\gamma\delta_1 \dots \delta_{i-1} h_1 \dots h_k \delta_i$ is a trace for $E|||RUN_H$. This means that $E|||RUN_H \notin Det$ hence $E \notin$ *L-Sec*. We obtain a contradiction, so no δ_i can be refused by $E' \setminus Act_H$ and δ is a trace for such process. So we have that $E' \setminus Act_H \stackrel{\delta}{\Longrightarrow} E''' \setminus Act_H$.

Now we want to prove that $(\delta, K) \in E' \setminus Act_H$. Let $E' \setminus Act_H \stackrel{\delta}{\Longrightarrow} E''' \setminus Act_H$ and suppose that $E''' \setminus Act_H$ can execute a certain action $a \in K \cap Act_L$ (the actions in $K \cap Act_H$ cannot be executed by such process) then $\gamma\delta a$ is a trace for $E|||RUN_H$. Now consider the sequence δ' obtained by adding to δ all the high level action executed by E' in order to reach E'' in the transition $E'/Act_H \stackrel{\delta}{\Longrightarrow} E''/Act_H$; i.e. $E' \stackrel{\delta'}{\Longrightarrow} E''$. Then we will have that $E'|||RUN_H \stackrel{\delta'}{\Longrightarrow} E''|||RUN_H$ and since $E''/Act_H \stackrel{K}{\not\Rightarrow}$ then $E''|||RUN_H \stackrel{a}{\not\Rightarrow}$ and so $(\gamma\delta', \{a\}) \in E|||RUN_H$. Now if $\gamma\delta a$ is a trace for $E|||RUN_H$ then also $\gamma\delta' a$ is, and so, again, we obtain that $E|||RUN_H \notin Det$ and $E \notin$ *L-Sec*. Hence $E''' \setminus Act_H \stackrel{a}{\not\Rightarrow}$ for every $a \in K$ and so $(\delta, K) \in E' \setminus Act_H$.

Now we prove that if $(\delta, K) \in E' \setminus Act_H$ then $(\delta, K) \in E'/Act_H$. Suppose $(\delta, K) \in E' \setminus Act_H$. Then we have that $\exists E''$ such that $E' \setminus Act_H \stackrel{\delta}{\Longrightarrow} E'' \setminus Act_H \stackrel{K}{\not\Rightarrow}$. Hence also $E'/Act_H \stackrel{\delta}{\Longrightarrow} E''/Act_H$. Suppose that E''/Act_H can execute a certain $a \in K \cap Act_L$ then consider δ' obtained by adding to δ all the high level actions executed by E' before a in the transition $E'/Act_H \stackrel{\delta}{\Longrightarrow} E''/Act_H \stackrel{a}{\Longrightarrow} E'''/Act_H$, i.e., such that $\delta' a$ is a trace for E'. We have that $\gamma\delta' a$ is a trace for $E|||RUN_H$. Now, $(\delta, \{a\}) \in E' \setminus Act_H$ with $a \in Act_L$ and so $(\delta, \{a\}) \in E'|||RUN_H$ which implies that $(\delta', \{a\}) \in E'|||RUN_H$ and finally $(\gamma\delta', \{a\}) \in E|||RUN_H$. This

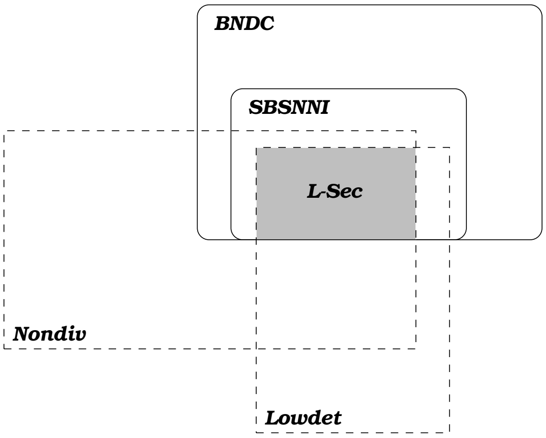


Fig. 15. Relations among properties.

contradict the fact that $E \in L\text{-}Sec$ and so $E''/Act_H \stackrel{a}{\not\Rightarrow}, \forall a \in K$. Hence $(\delta, K) \in E'/Act_H$. ∎

Theorem 10. *SFSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv* $\subseteq$ *L-Sec.*

Proof. Let $E \in SFSNNI \cap Lowdet \cap Nondiv$ and γa be a trace for process $E|||RUN_H$. We want to prove that $(\gamma, \{a\}) \notin E|||RUN_H$. It trivially holds if $a \in Act_H$ because in such a case it can always be executed by RUN_H. So let $a \in Act_L$. Suppose $E|||RUN_H \stackrel{\gamma}{\Longrightarrow} E'|||RUN_H \stackrel{a}{\not\Rightarrow}$ and consider the sequence γ' obtained removing all the high level actions from γ. Then $E/Act_H \stackrel{\gamma'}{\Longrightarrow} E'/Act_H$ and by hypothesis $E'/Act_H \approx_F E' \backslash Act_H$. Since $E'|||RUN_H \stackrel{a}{\not\Rightarrow}$ then $E' \backslash Act_H \stackrel{a}{\not\Rightarrow}$ and so $E'/Act_H \stackrel{a}{\not\Rightarrow}$ and $(\gamma', \{a\}) \in E/Act_H$. Since $E \in SFSNNI$ we obtain that $(\gamma', \{a\}) \in E \backslash Act_H$. Now γa is a trace for $E|||RUN_H$ and so $\gamma' a$ must be a trace for E/Act_H this means that $\gamma' a$ is also a trace for $E \setminus Act_H$. Since $E \in Lowdet$ then $E \backslash Act_H$ is deterministic. However we found that $\gamma' a$ is a trace for $E \backslash Act_H$ and $(\gamma', \{a\}) \in E \backslash Act_H$ obtaining a contradiction. So $E'|||RUN_H$ cannot refuse a and $(\gamma, \{a\}) \notin E|||RUN_H$. Hence $E|||RUN_H \in Det$ and since $E \in Nondiv$ we also have that $E|||RUN_H \in Nondiv$ ∎

Corollary 3. *SFSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv* $=$ *L-Sec.*

Proof. By Theorems 6 and 9 and by Definition 17 we find that L-Sec $\subseteq$ *SFSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv*. Finally by Theorem 10 we obtain the thesis. ∎

Note that by Corollary 2 we also have that $FNDC \cap Lowdet \cap Nondiv = L\text{-}Sec$. Now we show that this result also holds for *SBSNNI* and *BNDC*. We first prove that for deterministic processes $\approx_F$ becomes equal to $\approx_B$.

Proposition 10. $E \in Det, E \approx_F F \Longrightarrow E \approx_B F$.

Proof. If $E \in Det$ and $E \approx_F F$ we also have that $F \in Det$. Now it is sufficient to consider the relation $R \subseteq \mathcal{E} \times \mathcal{E}$ defined as follows: $(E', E'') \in R$ if and only if $\exists \gamma : E \stackrel{\gamma}{\Longrightarrow} E'$, $E \stackrel{\gamma}{\Longrightarrow} E''$. It is easy to show that R is a weak bisimulation. ■

Finally, the following holds.

Theorem 11. *BNDC* $\cap$ *Lowdet* $\cap$ *Nondiv* $=$ *SBSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv* $=$ *L-Sec*.

Proof. (*SBSNNI*$\cap$*Lowdet*$\cap$*Nondiv* $=$ *L-Sec*). We have that *SBSNNI*$\cap$*Lowdet*$\cap$ *Nondiv* $\subseteq$ *SFSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv* because *SBSNNI* $\subset$ *SFSNNI*. So by Theorem 10 *SBSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv* $\subseteq$ *L-Sec*.
Now we prove that *L-Sec* $\subseteq$ *SBSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv*. If $E \in$ *L-Sec* then by Corollary 3 we have that $E \in$ *SFSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv*. So $\forall E'$ such that $\exists \gamma : E \stackrel{\gamma}{\Longrightarrow} E'$ we have $E' \setminus Act_H \approx_F E'/Act_H$ with $E \setminus Act_H \in Det$. In particular we also have that $E \setminus Act_H \approx_F E/Act_H$ and since $E \setminus Act_H \in Det$, we obtain that $E/Act_H \in Det$. Note that $E/Act_H \stackrel{\gamma'}{\Longrightarrow} E'/Act_H$ where γ' is the sequence obtained removing all the high level actions from γ. Hence, by Corollary 1, $E'/Act_H \in Det$. Finally, by Proposition 10 we obtain that $E' \setminus Act_H \approx_B E'/Act_H$.
(*BNDC*$\cap$*Lowdet*$\cap$*Nondiv* $=$ *SBSNNI*$\cap$*Lowdet*$\cap$*Nondiv*) Trivial by *SBSNNI* $\subset$ *BNDC* $\subset$ *FNDC* and since *SBSNNI* $\cap$ *Lowdet* $\cap$ *Nondiv* $=$ *L-Sec* $=$ *FNDC* $\cap$ *Lowdet* $\cap$ *Nondiv*. ■

Figure 15 summarizes the relations among various properties and conditions. We have shown that *BNDC* and *SBSNNI* are equal to *L-Sec* when dealing with low-deterministic and non-divergent processes. In the next section we will introduce the CoSeC tool which is able to automatically check the *SBSNNI* property over finite state agents. This implies that for low-deterministic, non-divergent and finite-state processes it is possible to use the CoSeC in order to verify also the *L-Sec* property. In [56] it is shown how to use the FDR tool [55] to check the *L-Sec* property. It would be interesting to compare the performance of FDR and CoSeC for the verification of such a property.

We also want to point out that *SBSNNI* $\cap$ *Lowdet* can extend in a *fair* manner the *L-Sec* property to divergent processes. *L-Sec* assumes that processes cannot diverge. The semantics used by authors to define *L-Sec* is the failure-divergence one [10]. Failure-divergence semantics gives a so-called *catastrophic* interpretation of divergences, since in the presence of divergences a process may show any behaviour. We have already seen this problem when we used testing equivalence for the definition of our properties. On the other hand bisimulation gives a *fair* interpretation of divergences by assuming that an infinite loop of internal actions will be taken a finite number of times, i.e., soon or later we will exit from the loop. This is useful, for example, if we want to model a fair communication media, where a τ-loop represents the unbounded but finite losses of messages. The property *SBSNNI* $\cap$ *Lowdet* can be seen as an extension of *L-Sec* which gives a fair interpretation of divergences.

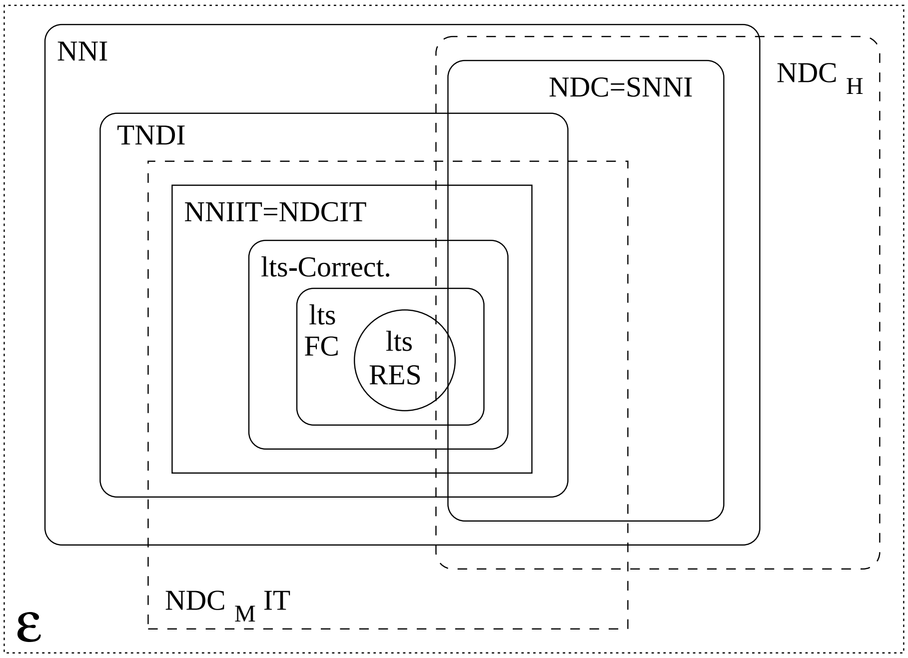


Fig. 16. The inclusion diagram for trace-based security properties.

A good reference about modelling NI in CSP can be found in this volume [58]. In such a paper, a new notion based on power bisimulation is also proposed. We intend to compare it with our bisimulation-based properties with the aim making our classification as much complete as possible.

3.6 Other Security Properties

In [24] we have compared our properties with a number of existing proposal. Here we just report the diagram (Figure 16) of the relations among such properties and we give the bibliographic references to them. In particular, TNDI derives from *Non Deducibility on Inputs* [62], lts-Correctability comes from *Correctability* [41], lts-FC is *Forward-Correctability* [41], lts-RES is a version of *Restrictiveness* [48]. Moreover NNIIT is *NNI* where we require that systems are *input total*. This means that in every state the system must be able to accept every possible input by the environment. The aim of this (quite restrictive) condition is to prevent users from deducing anything about the state of the system, even if they observe the inputs the system accepts. All the properties included in NNIIT requires this condition over input actions. NDCIT is *NDC* with input totality. Finally NDC$_H$ and NDC$_M$IT are parametric versions of *NDC* where the high level user can exploit only the actions in sets H and M to communicate with

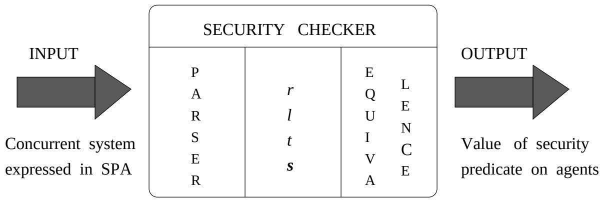


Fig. 17. Structure of the CoSeC

the system. Table 11 reports all the needed counterexamples which differentiate the properties. For more details, please refer to [24].

4 The Compositional Security Checker

In this Section we briefly describe the CoSeC structure and architecture. Before giving this description, we want to informally justify why the theory developed so far should be equipped with a tool. First of all, we need a tool to practically check the examples; even for small problems, the state space grows quite beyond our human ability to manage it. A tool is also useful to help intuition on what flaws these security properties really capture; only through the tool we can analyze easily a large number of examples, and refine our understanding of the security properties and of the tractability of validating them. Finally, a tool gives an idea of which kind of verification techniques engineers should become acquainted with in the future to certify their products (e.g., security protocols). As we will show in the next sections, the CoSeC tool has been obtained by modifying the Concurrency Workbench [14]. Part of the material contained in this Section has been published in [26, 32, 25].

4.1 Input-Output and Architecture

The inputs of CoSeC are concurrent systems expressed as SPA agents. The outputs are answers to questions like: "does this system satisfy that specific security property ?". The structure of CoSeC is described in Figure 17. In detail, the tool is able:

– to parse SPA agents, saving them in suitable environments as parse trees;
– to give a semantics to these parse trees, building the corresponding rooted labelled transition systems (RLTS for short); this phase terminates only if the SPA term generates a finite LTS.
– to check if an agent satisfies a certain security property; the routine implemented for this purpose verifies the equivalence of two particular agents modeled as RLTS. In this way, future changes to the language will not compromise the validity of the core of the tool.

	NNI	NDC SNNI	TNDI	lts-RES	lts-cor.	lts-FC	$\mathrm{NDC_H}$	NNIIT NDCIT	$\mathrm{NDC_M IT}$
NNI	$=$	**8**	2	2	2	2	9	10	10
NDC SNNI	$\subset$	$=$	**2**	2	2	2	$\subset$	**10**	10
TNDI	$\subset$	1	$=$	3	3	3	9	**10**	10
lts-RES	$\subset$	**1**	$\subset$	$=$	$\subset$	$\subset$	**9**	$\subset$	$\subset$
lts-cor.	$\subset$	1	$\subset$	6	$=$	6	9	$\subset$	$\subset$
lts-FC	$\subset$	1	$\subset$	**5**	$\subset$	$=$	9	$\subset$	$\subset$
$\mathrm{NDC_H}$	**7**	7	7	7	7	7	$=$	10	10
NNIIT NDCIT	$\subset$	1	$\subset$	3	**3**	3	9	$=$	$\subset$
$\mathrm{NDC_M IT}$	**4**	4	4	4	4	4	3	4	$=$

Let $T[x,y]$ be the table element contained in row x and column y. If $T[x,y] \in \{=, \subset\}$ then $xT[x,y]y$. If $T[x,y] = n$ then the agent n below is in x and is not in y. In the following Π_0 represents the Input-Total empty agent: $\Pi_0 = \sum_{i \in I} i.\Pi_0$.

1) $Z = \sum_{i\in I} i.Z + \bar{h}.Z'$ and $Z' = \sum_{i\in I} i.Z' + \bar{l}.\Pi_0$
2) $h.l.0 + l.0 + l'.0$
3) $Z = \sum_{i\in I} i.Z + \bar{h}.(\Pi_0 + \bar{l}.\Pi_0)$
4) $Z = \sum_{i\in I} i.Z + h.(\Pi_0 + \bar{l}.\Pi_0)$ with $h \notin M \cup \bar{M}$
5) $Z = \sum_{i\in I} i.Z + \bar{l}.\Pi_0 + h.\Pi_0$
6) $Z = \sum_{i\in I} i.Z + \bar{h}.Z' + \bar{h}'.Z''$ and $Z' = \Pi_0 + h.Z' + \bar{l}.\Pi_0$ and $Z'' = \sum_{i\in I\setminus\{h\}} i.Z'' + h.\Pi_0 + \bar{l}.\Pi_0$
7) $h.l.0$ with $h \notin H \cup \bar{H}$
8) $\bar{h}.l.0$
9) $Z = \sum_{i\in I} i.Z + \bar{h}.Z'$ and $Z' = \sum_{i\in I} i.Z' + \bar{l}.\Pi_0$ with $\bar{h} \in H \cup \bar{H}$
10) 0

Table 11. The inclusion table for trace-based security properties.

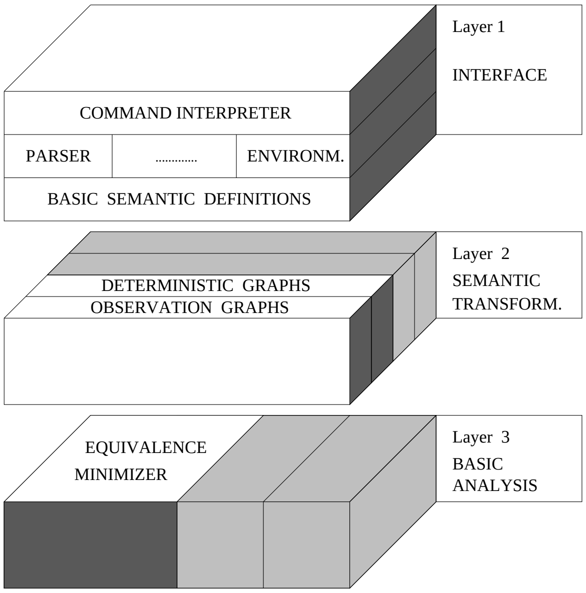


Fig. 18. CoSeC Architecture.

The CoSeC has the same general architecture of the CW [14]. In its implementation we have decided to exploit the characteristic of versatility and extensibility of CW. In particular CoSeC maintains the strongly modular architecture of CW.

Figure 18 shows the architecture of CoSeC. The modules of the system have been partitioned in three main layers: interface layer, semantic layer, analysis layer. In the interface layer we have the command interpreter. It allows us to define the agents and the set of high level actions; it also allows to invoke the security predicates and the utility functions on the behaviour of an agent. Then we have a parser which recognizes the SPA syntax of agents and stores them as parse trees in appropriate environments. The partition of the set of visible actions in the sets of high and low level actions has been obtained by defining the set of high level actions; by default, all the other possible actions are considered at low level. Then we have defined a function that, according to the operational semantic rule of SPA, provides all possible transitions for an agent. This function allows the construction of the LTS associated with an agent.

In the semantic layer, CoSeC uses two transformation routines to translate LTSs into deterministic and observation graphs [10] respectively. Since they both refer to processes modeled as LTSs, they have been imported from CW in CoSeC without any modification.

[10] An observation graphs [14] is obtained by obscuring the precise amount of internal computation. In particular the edges of the LTS are modified in order to reflect the $\rightsquigarrow$ relation defined as: $n \overset{\epsilon}{\rightsquigarrow} n'$ iff $n \overset{\tau}{\Longrightarrow} n'$ and $n \overset{a}{\rightsquigarrow} n'$ iff $n \overset{a}{\Longrightarrow} n'$.

In the analysis layer, CoSeC uses a routine of equivalence and one of minimization that belong to the analysis layer of CW. These are a slight modification of the algorithm by Kanellakis and Smolka [42] which finds a bisimulation between the roots of two finite state LTSs by partitioning their states. It is interesting to note that a simple modification of this algorithm can be used to obtain the minimization of a finite state LTS.

4.2 Checking the Information Flow Properties

Here we describe in details how the verification of Information Flow Properties proceeds. As we said before, we have properties which are in the following form:

$$E \text{ is } X\text{-secure if and only if } \mathcal{C}_X[E] \approx \mathcal{D}_X[E].$$

We have seen for *SNNI* that $\mathcal{C}_X[-] = - \setminus Act_H$ and $\mathcal{D}_X[-] = -/Act_H$. Hence, checking the X-security of E is reduced to the "standard" problem of checking semantic equivalence between two terms having E as a sub-term.

In the following we briefly explain how the system works in evaluating security predicates *NNI, SNNI, NDC, BNNI, BSNNI, SBSNNI*, and we discuss about their computational complexity. CoSeC computes the value of these predicates over finite state agents (i.e. agents possessing a finite state LTS), based on the definitions given in Section 3 that we report below in CoSeC syntax (for ease of parsing, in CoSeC the hiding and input restriction operators are represented by ! and ?, respectively.):

$$\begin{aligned}
E \in \mathit{NNI} &\Leftrightarrow E!Act_H \approx_T (E?Act_H)!Act_H \\
E \in \mathit{SNNI} \equiv \mathit{NDC} &\Leftrightarrow E!Act_H \approx_T E \setminus Act_H \\
E \in \mathit{BNNI} &\Leftrightarrow E!Act_H \approx_B (E?Act_H)!Act_H \\
E \in \mathit{BSNNI} &\Leftrightarrow E!Act_H \approx_B E \setminus Act_H \\
E \in \mathit{SBSNNI} &\Leftrightarrow E' \in \mathit{BSNNI},\ \forall E' \textit{ reachable from } E
\end{aligned}$$

As for CW, the inner computation of the CoSeC follows three main phases.

Phase a) (the same for all predicates) CoSeC builds the RLTSs of the two agents of which it wants to compute the equivalence. For example in the case of NNI, CoSeC computes the transition graph for $(E?Act_H)!Act_H$ and $E!Act_H$. In this phase we do not have any particular problem with complexity, except for the intrinsic exponential explosion in the number of states of the RLTS due to parallel composition.

Phase b) (This is split into two depending on the semantics requested by the security predicate)

b1: (for predicates *NNI, SNNI, NDC*) The two RLTSs obtained in Phase a) are transformed into deterministic graphs following the classic subset construction (see e.g. [40]). This algorithm has exponential complexity since it is theoretically possible, in the deterministic graph, to have a node for every subset of nodes in the original graph. However, experience shows that very often the number of obtained nodes is less than the number of nodes of the beginning graph because of the collapsing of the τ_transitions.

b2: (for predicates *BNNI* and *BSNNI*) The two RLTSs obtained in Phase a) are transformed into observation graphs using the classic algorithms for the product of two relations and the reflexive transitive closure of a relation. This transformation has a $O(n^3)$ complexity, in which n is the number of nodes in the original graph.

Phase c) (For all predicates) The general equivalence algorithm [42] is applied to the graphs obtained in Phase b). Time and space complexities of this algorithm are $O(k * l)$ and $O(k + l)$ respectively, where l is the number of nodes and k is the number of edges in the two graphs. This is not a limiting factor in the computation of the observational and trace equivalences. In particular, for observational equivalence, in most cases 80% of computation time is due to the routine for reflexive transitive closure of Phase b).

Since *SBSNNI* is verified by testing *BSNNI* over all the n states of the original graph, the resulting complexity will be n times the *BSNNI* complexity.

It is interesting to observe that the exponential explosion of the number of nodes of the transition graphs (Phase a), due to the operator of parallel composition, influences negatively the following phases, but it cannot be avoided because of its intrinsic nature. A solution to this problem for the predicates *NNI, SNNI, NDC* and *SBSNNI* could be based on the exploitation of compositional properties (see Section 4.5 for more details).

4.3 A Sample Session

The style used in specifying SPA agents in CoSeC is the same used for CCS agents in CW. For example the command line [11]

`Command:` **bi** $A \quad h.'l.'h.A + 'h.'l.A$

defines the agent $A \stackrel{\text{def}}{=} h.\bar{l}.\bar{h}.A + \bar{h}.\bar{l}.A$. As in CW the first letter of agents must be a capital letter and output actions have to be prefixed by $'$.

We assumed that the set of visible actions $\mathcal{L}$ is partitioned in two complete subsets Act_H and Act_L of high and low level actions respectively. With the command:

`Command:` **acth** $h \quad x$

we specify that $Act_H = \{h, 'h, x, 'x\}$. In this way we obtain that $h, 'h, x, 'x$ are considered as high level actions and any other action as low level one.

Now, we can check whether agent A is NNI secure:

[11] Here we use the typewriter style for CoSeC messages (such as the prompt "`Command:`"); the bold style for CoSeC commands and the italic style for the remaining text (such as agents and sets) inserted by users.

```
Command: nni  A
true
```

CoSeC tells us that A is NNI secure. Now we can check if agent A is SNNI secure too:

```
Command: snni  A
false
```

So A is *NNI* secure but is not *SNNI* secure. If we want to know why such a system is not *SNNI* we can use the *debugging* version of the *SNNI*:

```
Command: d_snni  A
false
Agent A!ActH
can perform action sequence 'l
which agent A\ActH
cannot
```

The tool shows a (low level) trace which distinguishes processes A/Act_H and $A \setminus Act_H$. The trace is $\bar{l}$ which can be executed only by the first one. This can be useful to understand why a process is not secure. Finally the command **quit** causes an exit to the shell.

4.4 An Example: Checking the Access Monitor

In this Section we use CoSeC to automatically check all the versions of the access monitor discussed in Example 6. Since CoSeC works on SPA agents we have to translate all the VSPA specifications into SPA. Consider once more $Access\_Monitor\_1$. Table 12 reports the translation of $Access\_Monitor\_1$ specification into the CoSeC syntax for SPA. [12] It has been used a new command **basi** which binds a set of actions to an identifier. Moreover, the \ character at the end of a line does not represent the restriction operator, but is the special character that permits to break in more lines the description of long agents and long action lists.

We can write to a file the contents of Table 12 and load it, in CoSeC, with command **if** <*filename*>. Now we can check that $Access\_Monitor\_1$ satisfies all the security properties except *SBSNNI* using the following command lines:

```
Command: bnni  Access_Monitor_1
true
Command: bsnni  Access_Monitor_1
true
Command: sbsnni  Access_Monitor_1
false: ('val_h1.Monitor | Object_l1 | Object_h1) \ L
```

[12] In the translation, we use values $\{l, h\}$ in place of $\{0, 1\}$ for the levels of users and objects in order to make the SPA specification clearer. As an example $access\_r(1, 0)$ becomes `access_r_hl`.

```
bi  Access_Monitor_1
 (Monitor  |  Object_l0  |  Object_h0)\ L

bi  Monitor
 access_r_hh.(rh0.'val_h0.Monitor + rh1.'val_h1.Monitor) + \
 access_r_lh.'val_l_err.Monitor + \
 access_r_hl.(rl0.'val_h0.Monitor + rl1.'val_h1.Monitor) + \
 access_r_ll.(rl0.'val_l0.Monitor + rl1.'val_l1.Monitor) + \
 access_w_hh.(write_h0.'wh0.Monitor + write_h1.'wh1.Monitor) + \
 access_w_lh.(write_l0.'wh0.Monitor + write_l1.'wh1.Monitor) + \
 access_w_hl.(write_h0.Monitor + write_h1.Monitor) + \
 access_w_ll.(write_l0.'wl0.Monitor + write_l1.'wl1.Monitor)

bi Object_h0
 'rh0.Object_h0 + wh0.Object_h0 + wh1.Object_h1

bi Object_h1
 'rh1.Object_h1 + wh0.Object_h0 + wh1.Object_h1

bi Object_l0
 'rl0.Object_l0 + wl0.Object_l0 + wl1.Object_l1

bi Object_l1
 'rl1.Object_l1 + wl0.Object_l0 + wl1.Object_l1

basi L
 rh0  rh1  rl0  rl1  wh0  wh1  wl0  wl1

acth
 rh0 rh1 wh0 wh1 access_r_hh access_r_hl val_h0 val_h1 val_h_err \
 access_w_hh access_w_hl write_h0 write_h1
```

Table 12. Translation of Access_Monitor_1 to CoSeC syntax for SPA.

Note that when CoSeC fails to verify *SBSNNI* on a process E, it gives as output an agent E' which is reachable from E and is not *BSNNI*.

So we have found that

$$Access\_Monitor\_1 \in \mathit{BSNNI}, \mathit{BNNI}$$

but

$$Access\_Monitor\_1 \notin \mathit{SBSNNI}$$

Since we have that $\mathit{SBSNNI} \subset \mathit{BNDC} \subset \mathit{BSNNI}, \mathit{BNNI}$, we cannot conclude whether $Access\_Monitor\_1$ is *BNDC* or not. However, using the output state E' of the *SBSNNI* verification, it is easy to find a high level process Π which can block the monitor. Indeed, in the state given as output by *SBSNNI*, the monitor is waiting for the high level action $'\mathtt{val\_h1}$; so, if we find a process Π which moves the system to such a state and does not execute the $\mathtt{val\_h1}$ action, we will have a high level process able to block the monitor. It is sufficient to consider $\Pi = {'access\_r\_hh}.0$. Agent $(Access\_Monitor\_1|\Pi) \setminus Act_H$ will be blocked immediately after the execution of the read request by Π, moving to the following deadlock state:

$$(('\mathtt{val\_h0.Monitor} \mid \mathtt{Object\_l0} \mid \mathtt{Object\_h0}) \setminus \mathtt{L} \mid \mathtt{0}) \setminus \mathtt{Act_H}$$

(this state differs from the one given as output by *SBSNNI* only for the values stored in objects). It is possible to verify that $Access\_Monitor\_1 \notin \mathit{BNDC}$ by checking that $(Access\_Monitor\_1|\Pi) \setminus Act_H \not\approx_B Access\_Monitor\_1/Act_H$ using the following command:

`Command:` **bi** $Pi \quad {'access\_r\_hh}.0$
`Command:` **eq**
`Agent:` $(Access\_Monitor\_1 \mid Pi) \setminus acth$
`Agent:` $Access\_Monitor\_1 \; ! \; acth$
`false`

As we said in Example 6, such a deadlock is caused by synchronous communications in SPA. Moreover, using the CoSeC output again, we can find out that also the high level process $\Pi' = {'access\_w\_hl}.0$ can block $Access\_Monitor\_1$, because it executes a write request and does not send the corresponding value. Hence, in Example 6 we proposed the modified system $Access\_Monitor\_5$ with an interface for each level and atomic actions for write request and value sending. We finally check that this version of the monitor is *SBSNNI*, hence *BNDC* too:

`Command:` **sbsnni** $\quad Access\_Monitor\_5$
`true`

4.5 State Explosion and Compositionality

In this section we show how the parallel composition operator can increase exponentially the number of states of the system, and then how it can slow down

agent	B	D	$B\|D\|B$	$B\|D\|D\|B$
state number	3	3	27	81
time spent	<1 sec.	<1 sec.	~11 sec.	~270 sec.

Table 13. Number of states and time spent on a SPARC station 5.

the execution speed of security predicate verification. This is basically caused by the fact that we have all the possible interleaving of the actions executed by the parallel processes. In order to avoid this we exploit some results related to the compositionality of the proposed security properties. For some of the properties we have that if two systems are "secure" also their parallel composition is secure. So, the tool has a special feature that decomposes systems in their parallel components and then checks the properties over that components. If both of them are secure the tool will conclude that also the whole system is secure. Otherwise the check will be performed over the whole system. We prove that this method is correct and that it terminates. Moreover we show some modular versions of the Monitor that can be verified very efficiently with this compositional technique.

We start with a very simple example. Let us define in CoSeC the two agents B, D and the set Act_H of high level actions:

`Command:` **bi** $B \quad y.a.b.B + a.b.B$
`Command:` **bi** $D \quad {}'a.{}'b.(x.D + D)$
`Command:` **acth** $x \;\; y$

Let us check now if B and D are *SBSNNI* secure:

`Command:` **sbsnni** B
`true`
`Command:` **sbsnni** D
`true`

We have seen that *SBSNNI* is a compositional property, so the two agents $B|D|B$ and $B|D|D|B$ must also be *SBSNNI* secure. Hence the verification of these two agents could be reduced to the verification of their two basic components B and D only. The time spent in verifying *SBSNNI* directly on $B|D|B$ and $B|D|D|B$ is very long. Using the **size** command of CoSeC, which computes the number of states of an agent, we can fill in Table 13, which points out the exponential increase of the number of states and the consequent increase of the computation time for verification of *SBSNNI*.

CoSeC is able to exploit the compositionality of security properties through an algorithmic scheme we are going to present. For a certain compositional property P this scheme also requires the following condition: if $Z \stackrel{\mathrm{def}}{=} E$ and E is P-secure then also Z is P-secure. This condition is satisfied by all the above presented properties because of Theorem 5.

Definition 20. *(Compositional Algorithm) Let $P \subseteq \mathcal{E}$ be a set of SPA agents such that*

- $E, E' \in P \Longrightarrow E|E' \in P$
- $E \in P, L \subseteq \mathcal{L} \Longrightarrow E \setminus L \in P$
- $E \in P, Z \stackrel{\mathrm{def}}{=} E \Longrightarrow Z \in P$

and let A_P be a decision algorithm which checks if a certain agent $E \in \mathcal{E}_{FS}$ belongs to P; in other words, $A_P(E) = true$ if $E \in P$, $A_P(E) = false$ otherwise. Then we can define a compositional algorithm $A'_P(E)$ in the following way:

1) *if E is of the form $E' \setminus L$, then compute $A'_P(E')$; if $A'_P(E') = true$ then return true, else return the result of $A_P(E)$;*
2) *if E is of the form $E_1|E_2$, then compute $A'_P(E_1)$ and $A'_P(E_2)$; if $A'_P(E_1) = A'_P(E_2) = true$ then return true, else return the result of $A_P(E)$;*
3) *if E is a constant Z with $Z \stackrel{\mathrm{def}}{=} E'$, then return the result of $A'_P(E')$;*
4) *if E is not in any of the three forms above, then return $A_P(E)$.* ■

The compositional algorithm $A'_P(E)$ works as the given algorithm $A_P(E)$ when the outermost operator of E is neither the restriction operator, nor the parallel one, nor a constant definition. Otherwise, it applies componentwise to the arguments of the outermost operator; if the property does not hold for them, we cannot conclude that the whole system is not secure, and we need to check it with the given algorithm.

Note that the compositional algorithm exploits the assumption that property P is closed with respect to restriction and uses this in step 1. This could seem of little practical use, as the dimension of the state space for, let say, E is often bigger than that of $E \setminus L$. However, parallel composition is often used in the form $(A|B) \setminus L$ in order to force some synchronizations, and so if we want to check P over A and B separately, we must be granted that P is preserved by both parallel and restriction operators.

To obtain the result for $A'_P(F)$, we essentially apply – in a syntax-driven way – the four rules above recursively, obtaining a proof tree having (the value of) $A'_P(F)$ as the root and the various (values of) $A_P(E)$'s on the leaves for the subterms E of F on which the induction cannot be applied anymore. The following theorem justifies the correctness of the compositional algorithm, by proving that the evaluation strategy terminates and gives the same result as the given algorithm $A_P(F)$.

Theorem 12. *Let $F \in \mathcal{E}_{FS}$. If the agent E' occurring in step 1 belongs to $\mathcal{E}_{FS}$ each time the algorithm A'_P executes that step, then $A'_P(F)$ terminates and $A_P(F) = A'_P(F)$.*

Proof. First we want to prove that, in computing $A'_P(F)$, if the evaluation of the given algorithm A_P is required on an agent E, then E belongs to $\mathcal{E}_{FS}$. The proof is by induction on the proof tree for the evaluation of $A'_P(F)$. The base case is when F can be evaluated by step 4; as – by hypothesis – agent F is finite state, the thesis follows trivially. Instead, if F is of the form $E' \setminus L$, then – by the premise of this theorem – $E' \in \mathcal{E}_{FS}$, and the inductive hypothesis can be applied. In step 2, as $F = E_1|E_2$, we have that $E_1, E_2 \in \mathcal{E}_{FS}$, and the inductive hypothesis can be applied to prove the thesis. Similarly for step 3, as constant Z is finite state if and only if the defining agent E' is so. So, when the algorithm executes $A_P(E)$ in steps 1, 2, 3 or 4, it always terminates because $E \in \mathcal{E}_{FS}$.

To complete the proof concerning termination of the compositional algorithm, we still have to prove that the inductive evaluation, in steps 1, 2 and 3, cannot loop; in other words, that the proof tree for $A'_P(F)$ is finite. While it is obvious for cases 1 and 2 (no term can be a proper subterm of itself in a finite term), the thesis follows in case 3 because of the constant guardedness condition over SPA agents. It guarantees that the recursive substitution of non prefixed constants with their definitions terminates. Hence the computation of $A'_P(F)$ ends.

To prove that the result of the compositional algorithm A'_P is consistent with the one obtained by the given algorithm A_P, we observe that the four rules above guarantee this, using compositionality properties for steps 1 and 2. ■

The theorem above requires that – in evaluating $A'_P(E)$ – if E is in the form $E' \setminus L$, then E' must be finite state. In fact, if we consider a finite state system $E \setminus L$ such that $E \notin \mathcal{E}_{FS}$, then $A_P(E \setminus L)$ terminates while $A'_P(E \setminus L)$ possibly do not, because it tries to compute $A_P(E)$ on the non-finite state agent E. The premise of the theorem above trivially holds for agents in the class of *nets of automata*.

The CoSeC command **c_sbsnni** checks the *SBSNNI* property exploiting compositionality. Let us now compare the compositional algorithm w.r.t. the normal one, starting from $Access\_Monitor\_5$. The normal verification of the *SBSNNI* property on such a system requires a lot of time (about 16 minutes [13] on a SUN5 workstation) because of the above mentioned exponential state explosion due to parallel composition. We could hope to get a better result using the compositional algorithm. Table 14 reports the output of the compositional verification of $Access\_Monitor\_5$ where the symbols `[\]` and `[|]` represent steps 1 and 2 of the algorithm, respectively. This table shows that the algorithm fails in the verification of *SBSNNI* over *AM* and then succeeds in checking the system as a whole.[14] Hence, in this case, the compositional technique cannot help reducing the execution time.

However, we can modify *AM* in order to obtain a *SBSNNI* system by making (only!) high level communications asynchronous. This can be done adding a high level buffer between the monitor and the interface. The resulting system is reported in Table 15 where $j \in \{0, 1, err, empty\}$, $L = \{r, w, val(1, y)\}$, $N = \{res,\ access\_r,\ access\_w\}$ and $res(1, y) \in Act_H, \forall y \in \{0, 1, err, empty\}$, while the same actions with 0 as first parameter belong to Act_L. Note that we have modified the interface so that it is now able to wait until the high buffer is filled by the monitor.

[13] This value and all the following are obtained exploiting also the state minimization feature of the tool.

[14] The reason why it fails on *AM* is because *AM* is essentially $Access\_Monitor\_1$ with atomic write operations; hence, it is not *SBSNNI* because of the possible high level deadlock of Example 6 caused by synchronous communications. The interface was indeed introduced in order to make communications asynchronous.

```
Command: c_sbsnni   Access_Monitor_5
[\] Verifying AM | Interf
  [|] Verifying AM
    [\] Verifying Monitor_5 | Object_l0 | Object_h0
      [|] Verifying Monitor_5
      [|] Failed!
    [\] Failed!
        Verifying directly (Monitor_5 | Object_l0 | Object_h0)\L
  [|] Failed!
[\] Failed!
    Verifying directly (AM | Interf)\K
true
```

Table 14. Verification of *SBSNNI* on *Access_Monitor_5* with the compositional algorithm.

$$
\begin{aligned}
Access\_Monitor\_6 &\stackrel{\text{def}}{=} (AM\_6 \mid Interf\_6) \setminus N \\
AM\_6 &\stackrel{\text{def}}{=} ((Monitor\_5 \mid Object(1,0) \mid Object(0,0) \\
&\qquad \mid hBuf(empty)) \setminus L)[res(0,y)/val(0,y)] \\
hBuf(j) &\stackrel{\text{def}}{=} \overline{res}(1,j).hBuf(empty) + val(1,k).hBuf(k) \\
Interf\_6 &\stackrel{\text{def}}{=} Interf\_6(0) \mid Interf\_6(1) \\
Interf\_6(l) &\stackrel{\text{def}}{=} a\_r(l,x).\overline{access\_r}(l,x).Interf\_6\_reply(l) \\
&\quad + \\
&\quad a\_w(l,x,z).\overline{access\_w}(l,x,z).Interf\_6(l) \\
Interf\_6\_reply(l) &\stackrel{\text{def}}{=} res(l,y). \\
&\quad (\ \textbf{if}\ y = empty\ \textbf{then} \\
&\qquad\quad Interf\_6\_reply(l) \\
&\qquad \textbf{else} \\
&\qquad\quad \overline{put}(l,y).Interf\_6(l))
\end{aligned}
$$

Table 15. The *Access_Monitor_6*.

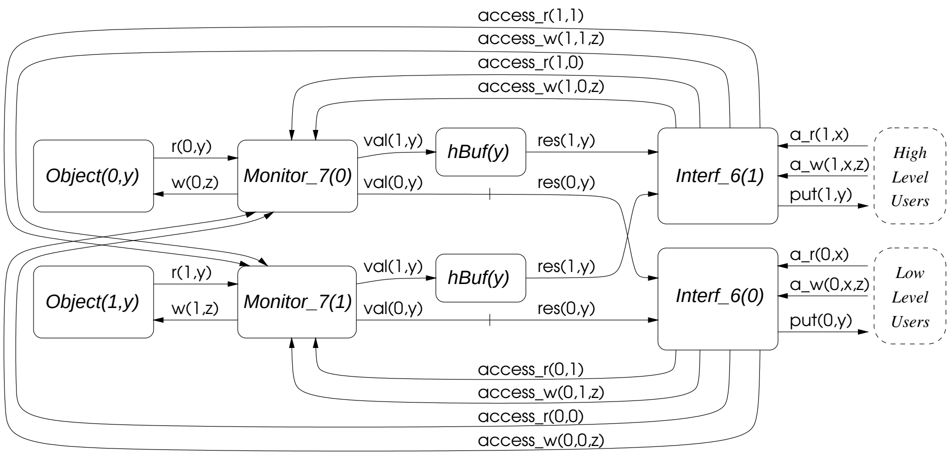


Fig. 19. The Modular $Access\_Monitor\_7$

Using the compositional algorithm, system $Access\_Monitor\_6$ can be checked very efficiently; the verification of *SBSNNI* takes about 90 seconds (see Table 16). We can also check that $Access\_Monitor\_5 \approx_B Access\_Monitor\_6$; so, as expected, the introduction of the buffer does not modify the behaviour of the monitor. This verification requires about 2 minutes. Note that, by Theorem 5, we can conclude that also $Access\_Monitor\_5$ is *SBSNNI*, even if a direct check takes (as we said) about 16 minutes.

```
Command: c_sbsnni   Access_Monitor_6
[\] Verifying AM_6 | Interf_6
  [|] Verifying AM_6
  [|] Verifying Interf_6
    [|] Verifying Interf_6_1
    [|] Verifying Interf_6_0
true
```

Table 16. Verification of *SBSNNI* on $Access\_Monitor\_6$ exploiting compositionality.

$Access\_Monitor\_6$ represents an example of successful application of the compositional checking; nonetheless, this does not mean that we cannot do better. Indeed, such a system is not defined in a very modular way and we hope that a more modular definition will lead to a more efficient compositional verification.

In fact, suppose we want to add other objects to $Access\_Monitor\_6$; in such a case, the size of AM_6 will increase exponentially with respect to the number of added objects. Now we present a rather modular version of the access monitor. The basic idea of this new version (Figure 19) is that every object has a "private" monitor which implements the access functions for such (single) object. To make this, we have decomposed process $Monitor\_5$ into two different processes, one for each object; then we have composed such processes to their respective objects together with a high level buffer obtaining the *SBSNNI*-secure $Modh$ and $Modl$ agents. In particular, $Monitor\_7(x)$ handles the accesses to object x ($x = 0$ low, $x = 1$ high). As in $Access\_Monitor\_6$, we have an interface which guarantees the exclusive use of the monitor within the same level and is able to read values from the high buffer. The resulting system is reported in Table 17 where $L = \{res, access\_r, access\_w\}$ and $Lh = \{r, w, val(1, y)\}$. Table 18 reports the output of the (successful) verification of the *SBSNNI* property for $Access\_Monitor\_7$. This task takes about 20 seconds on a SUN5 workstation, supporting our claim that a modular definition would help. Moreover, we can also check the new version of the monitor is functionally equivalent to the previous ones: in about 5 minutes, CoSeC is able to check that $Access\_Monitor\_7 \approx_B Access\_Monitor\_5$, and so also $Access\_Monitor\_7 \approx_B Access\_Monitor\_6$.

$$
\begin{array}{rl}
Access\_Monitor\_7 \stackrel{\text{def}}{=} & (Modh \mid Modl \mid Interf\_6) \setminus L \\
Modh \stackrel{\text{def}}{=} & ((Monitor\_7(1) \mid Object(1,0) \mid hBuf(empty)) \setminus Lh) \\
& \quad [res(0,y)/val(0,y)] \\
Modl \stackrel{\text{def}}{=} & ((Monitor\_7(0) \mid Object(0,0) \mid hBuf(empty)) \setminus Lh) \\
& \quad [res(0,y)/val(0,y)] \\
Monitor\_7(x) \stackrel{\text{def}}{=} & access\_r(l,x). \\
& (\ \textbf{if}\ x \leq l\ \textbf{then} \\
& \qquad r(x,y).\overline{val}(l,y).Monitor\_7(x) \\
& \quad \textbf{else} \\
& \qquad \overline{val}(l,err).Monitor\_7(x)) \\
& \quad + \\
& access\_w(l,x,z). \\
& (\ \textbf{if}\ x \geq l\ \text{then} \\
& \qquad \overline{w}(x,z).Monitor\_7(x) \\
& \quad \textbf{else} \\
& \qquad Monitor\_7(x))
\end{array}
$$

Table 17. The $Access\_Monitor\_7$.

As a final remark, the compositional verification is more convenient only when building complex systems as parallel composition of simpler ones. For this reason, the tool offers to the user the choice between the normal and the compositional verification algorithms. It is up to the user to choose which one (s)he thinks could go better, or even to make them work in parallel.

```
Command: c_sbsnni   Access_Monitor_7
[\] Verifying Modh | Modl | Interf_6
  [|] Verifying Modh
  [|] Verifying Modl
  [|] Verifying Interf_6
    [|] Verifying Interf_6_1
    [|] Verifying Interf_6_0
true
```

Table 18. Verification of *SBSNNI* on $Access\_Monitor\_7$ exploiting compositionality.

5 Conclusion

In this paper we have proposed a formal model for the specification and analysis of information flow properties. We have adopted a particular algebraic style in the definition of such properties. Indeed we have always given properties which are parametric with respect to a notion of semantic equivalence. This is useful since we can change the discriminating power of a property by simply "plugging in" the appropriate equivalence notion. Moreover, in this way we have obtained very compact and simple definitions. We have also seen how this algebraic style can be very profitable when automatically checking the properties. Indeed we can reduce the task of checking a security property to the well studied problem of checking the semantic equivalence of two terms of the language.

We have seen that the main motivation for information flow properties is historically bound to system security, in particular to the detection of direct and indirect information flows inside a system. However we have obtained a very general setting where we can study if a certain class of users, the high level users, can in some way interfere with the low level ones. Indeed we have used this Non-Interference abstraction in order to model the absence of information flow: "if high level users cannot interfere with low level ones then no information flow is possible from high to low level".

We have tried to convince the reader that the properties we have proposed are satisfactory for the detection of information flows inside a system, in particular the *BNDC* property which can also detect flows due to potential high level deadlocks.

In recent papers [28, 3], the underlying model has been extended in order to deal with time and probability. Once an appropriate semantics equivalence has been defined in these new models, the *BNDC* property has been shown to naturally handle the new features of the model. In particular, in such models, *BNDC* has been shown to be able to detect timing and probabilistic convert channels, respectively.

Another aspect we are studying is the possibility of defining a criterion for evaluating the quality of information flow properties [33]. We are trying to do this by defining classes of properties which guarantee the impossibility of the construction of some "canonical" channels. We have seen, for example, that using some systems which are not *BNDC* it is possible to obtain a (initially noisy) perfect channel from high to low level. The aim is to classify the information flow properties depending on which kind of channels they effectively rule out.

We have seen that it is possible to automatically check almost all the properties we have presented. Indeed we are still looking for a good (necessary and sufficient) characterization of the *BNDC* property. We have also briefly presented the CoSeC tool. In [47], Martinelli has applied partial model checking techniques to the verification of *BNDC*, leading to the implementation of an automatic verifier [46] which is able to automatically synthetize the possible interfering high-level process.

As we have stated above, the setting we have proposed is quite general. We claim that information flow (or NI) properties could have a number of different applications since they basically capture the possibility for a class of users of modifying the behaviour of another user class. This generality has allowed to apply some variants of our properties to the analysis of cryptographic protocols [15, 30, 29, 31], starting from a general scheme proposed in [34]. This has been the topic of the second part of the course "Classification of Security Properties" at FOSAD'00 school, and we are presently working on a tutorial which will cover it [27].

This application of NI properties to network security is new to our knowledge. The interesting point is that they can be applied to the verification of protocols with different aims, e.g., authentication, secrecy, key-distribution. We have analyzed a number of different protocols, thanks to a new tool interface which permits to specify value-passing protocols and to automatically generate the enemy [15]; this has also allowed to find new anomalies in some cryptographic protocols [16].

In [19, 20, 11, 12], a new definition of entity authentication, which is based on explicit locations of entities, has been proposed. We are presently trying to characterize also this property through information flow. We also intend to carry the *BNDC* theory over more expressive process calculi, like, e.g., pi/spi-calculus [2] and Mobile Ambients [13]. This would allow to compare it with new recent security properties proposed on such calculi and reminiscent of some Non-Interference ideas (see, e.g., [37, 35]).

References

1. M. Abadi. "Secrecy by Typing in Security Protocols". *Journal of ACM*, 46(5):749–786, 1999.
2. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
3. A. Aldini. "Probabilistic Information Flow in a Process Algebra". To appear in proceedings of CONCUR'01, 2001.
4. P. G. Allen. "A Comparison of Non-Interference and Non-Deducibility using CSP". In *Proceedings of the Fourth IEEE Computer Security Foundations Workshop*, pages 43–54, Franconia, New Hampshire, June 1991.
5. D. E. Bell and L. J. La Padula. "Secure Computer Systems: Unified Exposition and Multics Interpretation". *ESD-TR-75-306, MITRE MTR-2997*, March 1976.
6. J. A. Bergstra and J. W. Klop. "Algebra of Communicating Processes with Abstraction". *Theoretical Computer Science*, 37:77–121, 1985.
7. P. Bieber and F. Cuppens. "A Logical View of Secure Dependencies". *Journal of Computer Security*, 1(1):99–129, 1992.
8. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. "Static Analysis of Processes for No Read-Up and No Write-Down". In *proc. of 2nd FoSSaCS'99*, Amsterdam, March 1999. Springer.
9. S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. "A Theory of Communicating Sequential Processes". *Journal of the Association for Computing Machinery*, 31(3):560–599, July 1984.
10. S. D. Brookes and A. W. Roscoe. "An Improved Failures Model for Communicating Processes". In *Proceedings of the Pittsburgh seminar on concurrency*, pages 281–305. Springer-Verlag, LNCS 197, 1985.
11. C. Bodei, P. Degano, R. Focardi, and C. Priami. "Authentication via Localized Names". In *Proceedings of CSFW'99*, pages 98–110. IEEE press, 1999.
12. C. Bodei, P. Degano, R. Focardi, and C. Priami. "Primitives for Authentication in Process Algebras". Theoretical Computer Science, to appear, 2001.
13. L. Cardelli and A. Gordon. "Mobile Ambients". In *proceedings of FoSSaCS'98*, pages 140–155. Springer LNCS 1378, 1998.
14. R. Cleaveland, J. Parrow, and B. Steffen. "The Concurrency Workbench: a Semantics Based Tool for the Verification of Concurrent Systems". *ACM Transactions on Programming Languages and Systems*, Vol. 15 No. 1:36–72, January 1993.
15. A. Durante, R. Focardi, and R. Gorrieri. "A Compiler for Analysing Cryptographic Protocols Using Non-Interference". *ACM Transactions on Software Engineering and Methodology*, 9(4):489–530, 2000.
16. A. Durante, R. Focardi, and R. Gorrieri. "CVS at Work: A Report on New Failures upon Some Cryptographic Protocols". In *proceedings of Mathematical Methods, Models and Architectures for Computer Networks Security*, pages 287–299, St. Petersburg, Russia, May 2001. LNCS 2052.
17. N. Durgin, J. Mitchell, and D. Pavlovic. "Protocol composition and correctness". In *proceedings of Workshop on Issues in the Theory of Security (WITS '00)*, University of Geneva, July 2000.
18. R. Focardi. "Comparing Two Information Flow Security Properties". In *Proceedings of Ninth IEEE Computer Security Foundation Workshop, (CSFW'96), (M. Merritt Ed.)*, pages 116–122. IEEE press, June 1996.
19. R. Focardi. "Located Entity Authentication". Technical Report CS98-5, University of Venice, 1998.

20. R. Focardi. “Using Entity Locations for the Analysis of Authentication Protocols”. In *Proceedings of Sixth Italian Conference on Theoretical Computer Science (ICTCS'98)*, November 1998.
21. R. Focardi. *Analysis and Automatic Detection of Information Flows in Systems and Networks*. PhD thesis, University of Bologna (Italy), 1999.
22. R. Focardi and R. Gorrieri. “An Information Flow Security Property for CCS”. In *Proceedings of the Second North American Process Algebra Workshop (NAPAW '93)*, TR 93-1369, Cornell (Ithaca), August 1993.
23. R. Focardi and R. Gorrieri. “A Taxonomy of Trace-based Security Properties for CCS ”. In *Proceedings Seventh IEEE Computer Security Foundation Workshop, (CSFW'94), (Li Gong Ed.)*, pages 126–136, Franconia (NH), June 1994. IEEE Press.
24. R. Focardi and R. Gorrieri. “A Classification of Security Properties for Process Algebras”. *Journal of Computer Security*, 3(1):5–33, 1994/1995.
25. R. Focardi and R. Gorrieri. “Automatic Compositional Verification of Some Security Properties”. In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 167–186, Passau (Germany), March 1996. Springer-Verlag, LNCS 1055.
26. R. Focardi and R. Gorrieri. “The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties”. *IEEE Transactions on Software Engineering*, 23(9):550–571, September 1997.
27. R. Focardi, R. Gorrieri, and F. Martinelli. “Classification of Security Properties (Part II: Network Security)”. Forthcoming.
28. R. Focardi, R. Gorrieri, and F. Martinelli. “Information Flow Analysis in a Discrete Time Process Algebra”. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW13), (P.Syverson ed.)*, pages 170–184. IEEE CS Press, July 2000.
29. R. Focardi, R. Gorrieri, and F. Martinelli. Message authentication through non-interference. In *Proc. of 8th International Conference in Algebraic Methodology and Software Technology* (AMAST), 2000.
30. R. Focardi, R. Gorrieri, and F. Martinelli. “Non Interference for the Analysis of Cryptographic Protocols”. In *Proceedings of ICALP'00*, pages 744–755. LNCS 1853, July 2000.
31. R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in security protocols as non-interference. In *Workshop on secure architectures and information flow*, volume 32 of *ENTCS*, 2000.
32. R. Focardi, R. Gorrieri, and V. Panini. “The Security Checker: a Semantics-based Tool for the Verification of Security Properties”. In *Proceedings Eight IEEE Computer Security Foundation Workshop, (CSFW'95) (Li Gong Ed.)*, pages 60–69, Kenmare (Ireland), June 1995. IEEE Press.
33. R. Focardi, R. Gorrieri, and R. Segala. “A New Definition of Multilevel Security”. In *proceedings of Workshop on Issues in the Theory of Security (WITS '00)*, University of Geneva, July 2000.
34. R. Focardi and F. Martinelli. “A Uniform Approach for the Definition of Security Properties”. In *Proceedings of World Congress on Formal Methods (FM'99)*, pages 794–813. Springer, LNCS 1708, 1999.
35. C. Fournet and M. Abadi. “Mobile Values, New Names, and Secure Communication”. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

36. J. A. Goguen and J. Meseguer. “Security Policy and Security Models”. In *Proceedings of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, April 1982.
37. M. Hennessy and J. Riely. “Information Flow vs. Resource Access in the Asynchronous Pi-Calculus”. In *proceedings of ICALP*, pages 415–427, 2000.
38. Y. Hirshfeld. “Bisimulation Trees and the Decidability of Weak Bisimulations”. Technical report, Tel Aviv University, 1996.
39. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
40. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation.*, pages 22–24. Addison-Wesley, 1979.
41. D. M. Johnson and F. J. Thayer. “Security and the Composition of Machines”. In *Proceedings of the Computer Security Foundations Workshop*, pages 72–89, June 1988.
42. P. Kanellakis and S.A. Smolka. “CCS Expressions, Finite State Processes, and Three Problems of Equivalence”. *Information & Computation 86*, pages 43–68, May 1990.
43. R. Keller. “Formal Verification of Parallel Programs”. *Communications of the ACM*, 19 (7):561–572, 1976.
44. G. Lowe. “Casper: A Compiler for the Analysis of Security Protocols”. *Journal of Computer Security*, 6:53–84, 1998.
45. G. Lowe and B. Roscoe. “Using CSP to detect Errors in the TMN Protocol”. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.
46. D. Marchignoli and F. Martinelli. Automatic verification of cryptographic protocols through compositional analysis techniques. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and the Analysis of Systems (TACAS)*, 1999.
47. F. Martinelli. “Partial Model Checking and Theorem Proving for Ensuring Security Properties”. In *Proceedings of the 11th Computer Security Foundation Workshop, (CSFW’98)*. IEEE press, 1998.
48. D. McCullough. “Noninterference and the Composability of Security Properties”. In *Proceedings, 1988 IEEE Symposium on Security and Privacy*, pages 177–186. IEEE Computer Society Press, April 1988.
49. J. K. Millen. “Hookup Security for Synchronous Machines”. In *Proceedings of the Third Computer Security Foundation Workshop III*. IEEE Computer Society Press, 1990.
50. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
51. J. C. Mitchell, M. Mitchell, and U. Stern. “Automated Analysis of Cryptographic Protocols Using Murϕ”. In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.
52. R. De Nicola and M. Hennessy. “Testing equivalences for processes”. *Theoretical Computer Science*, 34:83–133, 1984.
53. L. C. Paulson. “Proving Properties of Security Protocols by Induction”. In *10th Computer Security Foundations Workshop*, pages 70–83. IEEE Computer Society Press, 1997.
54. G. Plotkin. “A Structural Approach to Operational Semantics”. Technical Report DAIMI-FN-19, Aarhus University, 1981.
55. A. W. Roscoe. “Model Checking CSP”. In A. W. Roscoe (ed) *A Classical Mind*. Prentice Hall, 1994.
56. A. W. Roscoe, J. C. P. Woodcock, and L. Wulf. “Non-interference through Determinism”. In *Proceeding of European Symposium on Research in Computer Security 1994 (ESORICS’94)*, pages 33–53. Springer-Verlag LNCS 875, 1994.

57. P. Y. A. Ryan. “A CSP Formulation of Non-Interference”. In *Proceedings of the 1990 Computer Security Foundation Workshop III*, Franconia, 1990. IEEE press.
58. P.Y.A. Ryan. “Mathematical Models of Computer Security”. In this volume.
59. S. Schneider. “Verifying authentication protocols in CSP”. *IEEE Transactions on Software Engineering*, 24(9), September 1998.
60. G. Smith and D.M. Volpano. “Secure Information Flow in a Multi-Threaded Imperative Language”. In *Proc. of POPL*, pages 355–364, 1998.
61. L. J. Stockmeyer and A. R. Meyer. “Word problems requiring exponential time”. In *Proceedings of the 5th ACM Symposium on Theory of Computing*, pages 1–9, Austin, Texas, 1973.
62. D. Sutherland. “A Model of Information”. In *Proceedings of the 9th National Computer Security Conference*, pages 175–183. National Bureau of Standards and National Computer Security Center, September 1986.
63. C. R. Tsai, V. D. Gligor, and C. S. Chandersekaran. “On the Identification of Covert Storage Channels in Secure Systems”. *IEEE Transactions on Software Engineering*, pages 569–580, June 1990.
64. J. T. Wittbold and D. M. Johnson. “Information Flow in Nondeterministic Systems”. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 144–161. IEEE Computer Society Press, 1990.