The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties

Riccardo Focardi **Roberto Gorrieri**

Technical Report UBLCS-96-14

August 1996

Department of Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)

The University of Bologna Department of Computer Science Research Technical Reports are available in gzipped PostScript format via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` or via WWW at URL `http://www.cs.unibo.it/`. Plain-text abstracts organized by year are available in the directory `ABSTRACTS`. All local authors can be reached via e-mail at the address *last-name*`@cs.unibo.it`. Questions and comments should be addressed to `tr-admin@cs.unibo.it`.

Recent Titles from the UBLCS Technical Report Series

95-2 *Clepsydra Methodology*, P. Ciaccia, O. Ciancarini, W. Penzo, January 1995.

95-3 *A Unified Framework for the Specification and Run-time Detection of Dynamic Properties in Distributed Computations*, Ö. Babaoğlu, E. Fromentin, M. Raynal, January 1995 (Revised February 1995).

95-4 *Effective Applicative Structures*, A. Asperti, A. Ciabattoni, January 1995.

95-5 *An Open Framework for Cooperative Problem Solving*, M. Gaspari, E. Motta, A. Stutt, February 1995.

95-6 *Considering New Guidelines in Group Interface Design: a Group-Friendly Interface for the CHAOS System*, D. Bottura, C. Maioli, S. Mangiaracina, February 1995.

95-7 *Modelling Interaction in Agent Systems*, A. Dalmonte, M. Gaspari, February 1995.

95-8 *Building Hypermedia for Learning: a Framework Based on the Design of User Interface*, S. Mangiaracina, C. Maioli, February 1995.

95-9 *The Bologna Optimal Higher-Order Machine*, A. Asperti, C. Giovannetti, A. Naletto, March 1995.

95-10 *Synchronization Support and Group-Membership Services for Reliable Distributed Multimedia Applications*, F. Panzieri, M. Roccetti, March 1995 (Revised July 1995).

95-11 *The Inherent Cost of Strong-Partial View-Synchronous Communication*, Ö. Babaoğlu, R. Davoli, L.-A. Giachini, P. Sabattini, April 1995.

95-12 *On the Complexity of Beta-Reduction*, A. Asperti, July 1995.

95-13 *Optimal Multi-Block Read Schedules for Partitioned Signature Files*, P. Ciaccia, August 1995.

95-14 *Integrating Performance and Functional Analysis of Concurrent Systems with EMPA*, M. Bernardo, L. Donatiello, R. Gorrieri, September 1995.

95-15 *On Programming with View Synchrony*, Ö. Babaoğlu, A. Bartoli, G. Dini, September 1995.

95-16 *Generative Communication in Process Algebra*, P. Ciancarini, R. Gorrieri, G. Zavattaro, October 1995.

95-17 *Dynamic Declustering Methods for Parallel Grid Files*, P. Ciaccia, A. Veronesi, November 1995.

95-18 *Failure Detectors, Group Membership and View-Synchronous Communication in Partitionable Asynchronous Systems (Preliminary Version)*, Ö. Babaoğlu, R. Davoli, A. Montresor, November 1995.

96-1 *An Investigation on the Optimal Implementation of Processes*, C. Laneve, January 1996.

96-2 *Expansivity, Permutivity, and Chaos for Cellular Automata*, F. Fagnani, L. Margara, January 1996.

96-3 *Enriched View Synchrony: A Paradigm for Programming Dependable Applications in Partitionable Asynchronous Distributed Systems*, Ö. Babaoğlu, A. Bartoli, G. Dini, February 1996 (Revised July 1996).

96-4 *May and Must Testing in the Join-Calculus*, C. Laneve, March 1996.

96-5 *The Shape of Shade: a Coordination System*, S. Castellani, P. Ciancarini, D. Rossi, March 1996.

96-6 *Engineering Formal Requirements: an Analysis and Testing Method for Z Documents*, P. Ciancarini, S. Cimato, C. Mascolo, March 1996.

96-7 *Using Bayesian Belief Networks for the Automated Assessment of Students' Knowledge of Geometry Problem Solving Procedures*, M. Roccetti, P. Salomoni, March 1996.

96-8 *Virtual Interactions: An Investigation of the Dynamics of Sharing Graphs*, A. Asperti, C. Laneve, April 1996.

96-9 *Towards an Algebra of Actors*, M. Gaspari, April 1996.

96-10 *Mobile Petri Nets*, A. Asperti, N. Busi, May 1996.

96-11 *Communication Support for Critical Distributed Multimedia Applications: an Experimental Study*, F. Panzieri, M. Roccetti, May 1996.

96-12 *A Logic Coordination Language Based on the Chemical Metaphor*, P. Ciancarini, D. Fogli, M. Gaspari, July 1996.

96-13 *On the Parallelization of Concurrent Systems I: Basic Definitions and Properties*, F. Corradini, R. Gorrieri, D. Marchignol, August 1996.

96-14 *The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties*, R. Focardi, R. Gorrieri, August 1996.

The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties [1]

Riccardo Focardi [2] **Roberto Gorrieri** [3]

Technical Report UBLCS-96-14

August 1996

Abstract

The Compositional Security Checker (CoSeC for short) is a semantic-based tool for the automatic verification of some compositional information flow properties. The specifications given as inputs to CoSeC are terms of the Security Process Algebra, a language suited for the specification of concurrent systems where actions belong to two different levels of confidentiality. The information flow security properties which can be verified by CoSeC are some of those classified in [5]. They are derivations of some classic notions, e.g. Non Interference [8]. The tool is based on the same architecture of the Concurrency Workbench [2], from which some modules have been integrally imported. The usefulness of the tool is tested with the significative case-study of an access-monitor, presented in several versions in order to illustrate the relative merits of the various information flow properties that CoSeC can check.

1. Extended and revised version of [6] and [7]. Work partially supported by CNR and MURST.
2. Università di Venezia Ca' Foscari, Dipartimento di Matematica Applicata e Informatica, Via Torino 155, I – 30173 Venezia Mestre (Italy). E-mail: focardi@cs.unibo.it.
3. Università di Bologna, Dipartimento di Scienze dell'Informazione, Mura Anteo Zamboni 7, 40127 Bologna, Italy. E-mail: gorrieri@cs.unibo.it.

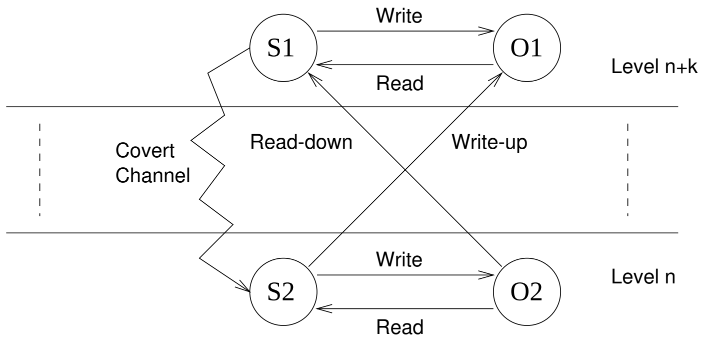

Figure 1. Information flows in multilevel security.

1 Introduction

Security is a crucial property of system behaviour, requiring a strict control over the information flow among parts of the system. The main problem is to limit, and possibly to avoid, the damages produced by malicious programs, called *Trojan Horses*, which try to broadcast secret information. There are several approaches to solve this problem.

In the *Discretionary Access Control* security (DAC for short), 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). An example of DAC is the file management in Unix where a user can decide the access possibilities of her/his files. So, if a user executes a Trojan Horse program, this can modify the security properties of user's objects.

A solution to this problem is the *Mandatory Access Control* (MAC for short), where access rules are imposed by the system. An example of MAC is *Multilevel Security* [1]: every object is bounded to a security level, and so every subject is; 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 no way to *downgrade* information, and its action is restricted inside such a level. There are 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 to a lower level object).

However, these access rules are not enough. It could be possible to indirectly transmit information using some system side effect. For example, if two levels – 'high' and 'low' – share some finite storage resource (e.g., a hard disk), it is possible to transmit data from level 'high' to level 'low' by using 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 ways of transmission, called *covert channels*, do not violate the two multilevel access rules (see Figure 1). Therefore it is necessary to integrate a MAC discipline with a covert channel analysis (see e.g. [19]).

An alternative, more general approach to security requires to control directly the whole flow of information, rather than the accesses of subjects to objects. To make this, it is necessary to choose a formal model of system behaviour and to define the information flow on such a model. By imposing some information flow rule, we can control any kind of transmission, be it direct or indirect.

In the literature, there are many different definitions of this kind based on several system models (see e.g. [8, 20]). In [5] we have uniformly rephrased them in the setting of *Security Process Algebra* (SPA, for short), then compared and classified, leading to our proposed notion of *Bisimulation Non Deducibility on Compositions* (*BNDC*, for short). SPA is an extension of CCS [14] – a language proposed to specify concurrent systems, whose programs are terms of an algebra comprising some operators for building systems bottom-up from smaller subsystems. The basic building blocks are the atomic activities, simply called *actions*; contrary to 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 [13], where the states are the terms of the algebra. The *operational* semantics for a SPA term E is the portion of the SPA labelled transition system (lts, for short) reachable from the initial state E. Then, in order to express that certain states are indistinguishable by an external observer, it is defined a *behavioural* semantics for SPA as an equivalence relation over terms/states. In this way the semantics of a certain SPA term becomes the class of terms observationally indistinguishable from it.

For some of the investigated information flow properties, we provided useful algebraic characterizations. They are all of the following 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 [4] for property X. Then, we can say:

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

Hence, checking the X-security of E is reduced to the "standard" problem of checking semantic equivalence between two terms having E as a subterm. In recent years a certain number of tools for checking semantic equivalence have been presented; among them, the Concurrency Workbench (CW for short) [2] is one of the most famous.

The first aim of this work is to present a tool called *Compositional Security Checker* which can be used to check automatically (finite state) SPA specifications against some information flow security properties. The tool has the same modular architecture of CW (Version 6.1), from which some modules have been integrally imported and some others only 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 [16]. 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 lts's; 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. To be more explicit, consider two agents E_1 and E_2; the number of states of the lts rooted in the state $E_1|E_2$ is equal to the product of the states of the lts's rooted in E_1 and E_2. Now if we have a compositional security property X, i.e. such that $F_1|F_2$ is X-secure whenever F_1 and F_2 are X-secure, then we can apply the following strategy: check the X-security of E_1 and E_2; if it is satisfied, then conclude that $E_1|E_2$ is X-secure, otherwise check the X-security of the whole agent $E_1|E_2$. Hence, for compositional security properties, CoSeC can avoid the complexity of the exponential state explosion if the components are also secure.

The second aim of the work is to assess the merits of the various information flow properties presented in this paper. We do this by showing, through a running example, which kind of flows are detectable by the various properties. In particular we want to show that certain properties are not appropriate to deal with some kind of information flows and so it is necessary to strengthen them or, if this is not possible, to follow a different approach.

The paper is organized as follows. In Section 2 we present SPA, together with the first version of our running example: an access monitor. In Section 3 we recall from [5] some of the security properties which are verified by CoSeC, giving some variations of the access monitor example. Section 4.1 reports the input-output behavior of CoSeC, while Section 4.2 describes the architecture of the tool. The implementation of the security predicates is the subject of Section 5. Then, a sample session with the interactive tool is described in Section 6.1 and Section 6.2 is devoted to the formal verification of the case-study. Section 7 is about the state explosion problem and the compositional algorithm. Finally, some concluding remarks about future research and related literature are reported in Section 8.

4. An SPA context $\mathcal{C}$ is an SPA term "with a hole". E.g., $\mathcal{C}[-] = F + -$. The insertion of E in the context $\mathcal{C}$, written as $\mathcal{C}[E]$, has the effect of filling the hole with E. In the example, $\mathcal{C}[E] = F + E$.

2 SPA, Semantic Equivalences and Value-Passing

2.1 The Model

The model used to represent systems is the labelled transition system model [13]. A labelled transition system is essentially an automaton which can have possibly infinite states. "A significant advantage of having the model based on a standard notion like the automaton is that extensive literature and well-developed intuition become immediately applicable to the problem domain" (quoted from the DoD-NCSC Integrity report [15]). In addition, labelled transition systems constitute the preferred semantic domain for the operational description of many concurrent languages.

Definition 2.1 *A* labelled transition system (lts) *is a triple* $(S, T, \rightarrow)$ *such that:*

- S *is a set of states*
- T *is a set of labels (actions)*
- $\rightarrow \subseteq S \times T \times S$ *is a set of labelled transitions* ■

$(S_1, \alpha, S_2) \in \rightarrow$ (or equivalently $S_1 \stackrel{\alpha}{\rightarrow} S_2$) means that the system can move from the state S_1 to the state S_2 through the action α.

2.2 The Language

In the following, systems will be specified using the *Security Process Algebra* (SPA for short), a slight extension of Milner's CCS [14], where the set of visible actions is partitioned into high level actions and low level ones in order to specify multilevel systems. [5]

SPA syntax is based on the following elements:

- A set $I = \{a, b, \ldots\}$ of *input* actions, a set $O = \{\bar{a}, \bar{b}, \ldots\}$ of *output* actions, and a set $\mathcal{L} = I \cup O$ of *visible* actions, ranged over by α.
- A function $\bar{\cdot} : \mathcal{L} \rightarrow \mathcal{L}$ such that if $a \in I$ then $\bar{a} \in O$ and if $\bar{a} \in O$ then $\bar{\bar{a}} = a \in I$. If $L \subseteq \mathcal{L}$, then $\bar{L} \stackrel{\mathrm{def}}{=} \{\bar{a} : a \in L\}$.
- Two sets Act_H and Act_L of high and low level actions such that $\overline{Act_H} = Act_H$, $\overline{Act_L} = Act_L$, $Act_H \cup Act_L = \mathcal{L}$ and $Act_H \cap Act_L = \emptyset$.
- A set $Act = \mathcal{L} \cup \{\tau\}$ of actions (τ is the internal, invisible action), ranged over by μ.
- A set K of constants, ranged over by Z.

The syntax of SPA *agents* is defined (as for CCS) as follows:

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

where $L \subseteq \mathcal{L}$ and $f : Act \rightarrow Act$ is such that $f(\bar{\alpha}) = \overline{f(\alpha)},\ f(\tau) = \tau$. Moreover, for every constant Z there must be the corresponding definition: $Z \stackrel{\mathrm{def}}{=} E$, and any occurrence of a constant in E must be inside a prefix context (the so-called *guardedness* condition).

The only difference with respect to CCS is the restriction operator $\setminus\!\setminus$; intuitively $E \setminus\!\setminus L$ 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 need this slight modification in order to define, below in this Section, an additional input restriction operator. Concerning the other operators, 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 choose [6] 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; if E can execute action μ, then $E[f]$ performs $f(\mu)$; finally, Z does what E does, if $Z \stackrel{\mathrm{def}}{=} E$.

---

5. 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.
6. For notational convenience, we use sometimes the $\sum$ operator (indexed on a set) to represent a general n-ary (or even infinitary) sum operator.

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

Table 1. The operational rules for SPA.

Now, we define the standard restriction operator of CCS (we will always use this one in the following) and two auxiliary operators, namely the hiding operator of CSP [10] and the input restriction operator, which are useful in characterizing some security properties in an algebraic style.

$$\begin{aligned} E \setminus L &\stackrel{\mathrm{def}}{=} E \setminus\setminus L \cup \bar{L} \\ E/L &\stackrel{\mathrm{def}}{=} E[f_L] \text{ where } f_L(x) = \begin{cases} x & \text{if } x \notin L \\ \tau & \text{if } x \in L \end{cases} \\ E \setminus_I L &\stackrel{\mathrm{def}}{=} E \setminus\setminus L \cap I \end{aligned}$$

Intuitively, $E \setminus L$ can execute all the actions E is able to do, provided that they do not belong to $L \cup \bar{L}$, while $E \setminus_I L$ requires that the actions of E do not belong to $L \cap I$; E/L turns all the actions in L into internal τ's.

2.3 Operational Semantics and Equivalences

Let $\mathcal{E}$ be the set of SPA agents, ranged over by E, 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 $\mathcal{E}_H \stackrel{\mathrm{def}}{=} \{E \in \mathcal{E} \mid \mathcal{L}(E) \subseteq Act_H \cup \{\tau\}\}$ and $\mathcal{E}_L \stackrel{\mathrm{def}}{=} \{E \in \mathcal{E} \mid \mathcal{L}(E) \subseteq Act_L \cup \{\tau\}\}$, respectively. Note that $\mathcal{E}_H \cup \mathcal{E}_L \subset \mathcal{E}$, i.e., there exist systems which executes both high and low level actions allowing communications between the two levels.

The operational semantics of SPA is the lts $(\mathcal{E}, Act, \rightarrow)$, where the states are the terms of the algebra and the transition relation $\rightarrow \subseteq \mathcal{E} \times Act \times \mathcal{E}$ is defined by structural induction as the least relation generated by the 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 "root" E.

Now we want to introduce the idea of observable behaviour: two systems should have the same denotation if and only if they cannot be distinguished by an external observer. The technique to obtain this is to 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 power of the observers. We recall two 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 instead considers more powerful observers that can discriminate agents also according to the time when the choices to follow one path or another in the lts have been taken. This equivalence is based on the concept of bisimulation [14].

Definition 2.2 *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 $\gamma = \alpha_1 \ldots \alpha_n \in \mathcal{L}^*$ be a sequence of actions; then $E \stackrel{\gamma}{\Longrightarrow} E'$ if and only if there exist $E_1, E_2, \ldots, E_{n-1} \in \mathcal{E}$ such that $E \stackrel{\alpha_1}{\Longrightarrow} E_1 \stackrel{\alpha_2}{\Longrightarrow} \cdots \stackrel{\alpha_{n-1}}{\Longrightarrow} E_{n-1} \stackrel{\alpha_n}{\Longrightarrow} E'$. We say that E' is* reachable *from E when $\exists \gamma : E \stackrel{\gamma}{\Longrightarrow} E'$ and we write $E \Longrightarrow E'$.* ■

Definition 2.3 *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 $T(E) = T(F)$.* ■

We recall also the definition of weak bisimulation [14] over SPA agents. 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 2.4 *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 observational equivalent, notation $E \approx_B F$, if there exists a weak bisimulation containing the pair (E, F). ■

In [14] it is proved that $\approx_B$ is an equivalence relation, indeed. Moreover, it is easy to see that $E \approx_B F$ implies $E \approx_T F$, as the latter is obtained by the former, forgetting the branching structure (i.e., the information about when choices are taken).

2.4 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 be done using such value passing calculus, because it originates more readable specifications than those written in pure SPA. Here we present a very simple example of value-passing agent showing how it can be translated to a pure SPA agent. Then we define VSPA syntax and we sketch the semantics by translating a generic VSPA agent to a SPA one.

As an example, consider the following buffer cell [14]:

$$\begin{array}{rcl} C & \stackrel{\mathrm{def}}{=} & in(x).C'(x) \\ C'(x) & \stackrel{\mathrm{def}}{=} & \overline{out}(x).C \end{array}$$

where x is a variable that can assume values in N (in the following we will simply write $x \in \mathrm{N}$). C reads a natural value n through action in and stores it in variable x, then such 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.

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

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

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

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

VSPA is very similar to the value-passing CCS introduced in [14]. 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:

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

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.

It is also necessary a definition for each constant $A(x_1,\ldots,x_m) \stackrel{\text{def}}{=} E$ where E is a VSPA agent which may contain no free variables except $x_1,\ldots,x_m$ (which must be distinct). In the following we will denote with $\mathcal{E}^+$ the set of all the VSPA agents. As in [14] the semantics of the value-passing calculus is given as a translation to 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 than it becomes a family of agents, one for each value of such variable. The translation can be given recursively on the structure of agents (see [14] for more details about this kind of translation).

Note that we do not partition the set of actions in 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 $x \in [1,n]$.

2.5 The Access Monitor

Here we give an example of 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 analyze and modify this example in the next sections in order to assess the merits of the various information flow properties we will propose.

Example 2.5 Consider the following system:

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

where $x,y,z,l \in \{0,1\}$, $L = \{r,w\}$ 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. The process $Access\_Monitor\_1$ (Figure 2) 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 only from the low one. Users interact with the monitor through the following access actions:

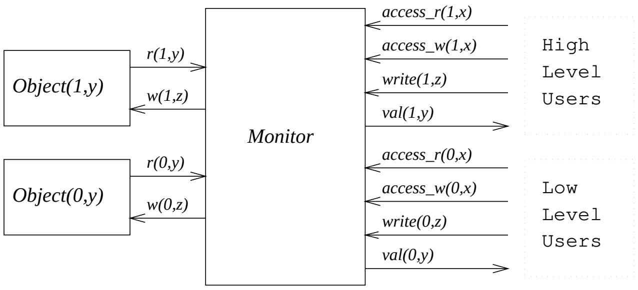

Figure 2. The Access Monitor for Example 2.5.

$access\_r(l,x), access\_w(l,x), write(l,z)$ 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 request is not executed and no error message is returned.

The complete translation of the VSPA agent $Access\_Monitor\_1$ to CoSeC syntax for pure SPA is reported in Table 2 of Section 6.2. As an example, here we provide the translation of $Object(x,y)$ to the pure calculus by means of the following four constant definitions:

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

■

3 Some Information Flow Properties

In this section we present some of the security properties (see [5] for more details) which can be verified using CoSeC. The common intuition behind all these properties is that the low level users should not be able to deduce anything about high level users' activity. We start with *Non-deterministic Non Interference* (*NNI*, for short), which is a natural generalization (assuming two user groups only) to non-deterministic systems of the classic notion of *Non Interference*, proposed for deterministic systems in [8]. Intuitively, the high level does not interfere with the low level if and only if for every trace γ, there exists a trace γ' that contains no high level input actions and is equivalent to γ for a low level user. This may be expressed algebraically as:

Definition 3.1 $E \in NNI \iff (E \setminus_I Act_H)/Act_H \approx_T E/Act_H.$ ■

Note that $Access\_Monitor\_1$ of Example 2.5 is *NNI*. In fact, the next example shows that *NNI* is able to detect if the multilevel access control rules are not implemented correctly in the monitor.

Example 3.2 Consider the following modified monitor [7] which does not control write accesses:

$$Access\_Monitor\_2 \stackrel{\text{def}}{=} (Monitor\_2 | Object(1,0) | Object(0,0)) \setminus L$$

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 previous versions of the Access Monitor.

$$
\begin{aligned}
Monitor\_2 \stackrel{\text{def}}{=}\ & access\_r(l,x).(\text{if } x \leq l \text{ then } r(x,y).\overline{val}(l,y).Monitor\_2 \\
& \text{else } \overline{val}(l,err).Monitor\_2) + \\
& + access\_w(l,x).write(l,z).\overline{w}(x,z).Monitor\_2
\end{aligned}
$$

Now it is possible for a high level user to write down (actions $access\_w(1,0)$ and $access\_w(1,1)$) 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

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

that cannot be a trace for $Access\_Monitor\_2$, since it represents the reading by a low level user of value 1 in the low object but the low object is initialized to value 0. Moreover it is not possible to obtain a trace for $Access\_Monitor\_2$ 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 a trace for $(Access\_Monitor\_2 \setminus_I Act_H)/Act_H$ too. In other words, it is not possible to find a trace γ'' 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 not for agent $(Access\_Monitor\_2 \setminus_I Act_H)/Act_H$, we conclude that $Access\_Monitor\_2$ is not *NNI*. ■

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

Example 3.3 Consider $Access\_Monitor\_1$, and suppose that we want to add a high level output signal which informs high level users about writings of low level users to the high level object. This could be useful to know the integrity of high level information. We obtain the following VSPA agent with the new $written\_up$ action:

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

where $\forall i \in \{0,1\}$, $written\_up(i) \in Act_H$. 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 to the high level object. If we purge γ from high level actions (i.e. from $written\_up$) 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 $Access\_Monitor\_3$ anymore because after every low level writings there must be a $written\_up$ action.

So, if a low level user succeeds in executing the two writing 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 $\overline{written\_up}$ and *NNI* is not able to detect this because it verifies only the high level input interferences over low level actions. ■

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

Definition 3.4 $E \in \mathit{SNNI} \Leftrightarrow E/\mathit{Act}_H \approx_T E \setminus \mathit{Act}_H$. ■

We have that $\mathit{Access\_Monitor\_3}$ is not *SNNI* since γ' is a trace for $\mathit{Access\_Monitor\_3}/\mathit{Act}_H$ and it is not a trace for $\mathit{Access\_Monitor\_3} \setminus \mathit{Act}_H$. We have the following obvious result, proved in [5]:

Proposition 3.5 *SNNI* $\subset$ *NNI*. ■

The next example shows 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 3.6 Suppose we have a high level action $\mathit{h\_stop}$ which explicitly stops the monitor. Obviously, in such a case there is a possible deadlock caused by a high level activity. Consider the following system:

$$\begin{aligned}
Access_Monitor_4 &\stackrel{\text{def}}{=} (Monitor_4 | Object(1,0) | Object(0,0)) \setminus L \\
Monitor_4 &\stackrel{\text{def}}{=} access_r(l,x).(\text{ if } x \leq l \text{ then } r(x,y).\overline{val}(l,y).Monitor_4 \\
&\quad \text{else } \overline{val}(l,err).Monitor_4) + \\
&\quad + access_w(l,x).write(l,z).(\text{ if } x \geq l \text{ then } \overline{w}(x,z).Monitor_4 \text{ else } Monitor_4) \\
&\quad + h_stop.\underline{0}
\end{aligned}$$

where $\mathit{h\_stop} \in \mathit{Act}_H$. It is *SNNI* because trace equivalence is not able to detect deadlocks and $\mathit{h\_stop}$ does not modify the low traces of the system. However, a low level user will certainly discover if the monitor stops, so it is important to detect such an event. ■

In order to detect this kind of flows, we introduce the bisimulation-based security properties *BNNI* and *BSNNI*, by substituting $\approx_B$ for $\approx_T$ in their SPA-based definitions. As expected, it can be proved [5] that each of these new properties is finer than its corresponding trace-based one. E.g. *BSNNI* $\subset$ *SNNI*.

Definition 3.7 *(Bisimulation NNI, SNNI)*
(i) $E \in \mathit{BNNI} \iff E/\mathit{Act}_H \approx_B (E \setminus_I \mathit{Act}_H)/\mathit{Act}_H$
(ii) $E \in \mathit{BSNNI} \iff E/\mathit{Act}_H \approx_B E \setminus \mathit{Act}_H$ ■

Considering again $\mathit{Access\_Monitor\_4}$, we find out that it is not *BSNNI*, as observation equivalence is able to detect deadlocks. In particular, $\mathit{Access\_Monitor\_4}/\mathit{Act}_H$ can move to $\underline{0}$ through an internal action τ, while $\mathit{Access\_Monitor\_4} \setminus \mathit{Act}_H$ 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 deadlocks due to high level activities. This will induce us to propose another property based on a different intuition. Let us consider $\mathit{Access\_Monitor\_1}$. 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 do 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), we proposed in [5]. 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

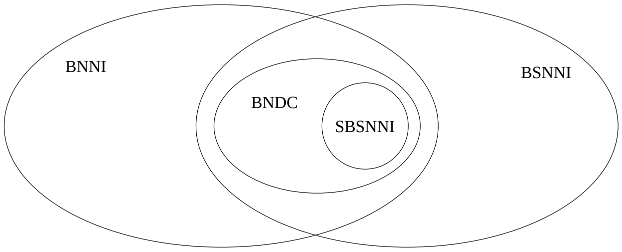

Figure 3. The inclusion diagram for bisimulation-based properties

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.

Definition 3.8 *E is BNDC if and only if* $\forall \Pi \in \mathcal{E}_H$, $E/Act_H \approx_B (E \mid \Pi) \setminus Act_H$. ■

Figure 3 summarizes the relations among the bisimulation-based security properties (including another property we are going to present later). So we have that *BNDC* is stronger than *BNNI* and *BSNNI*. Note that *BSNNI* $\not\subset$ *BNNI* while for trace equivalence we had that *SNNI* $\subset$ *NNI*. It is also interesting to observe that the trace based *BNDC* (*NDC*, for short) is exactly equal to *SNNI* (*NDC* = *SNNI*) [5]. So in the trace equivalence setting we gain nothing by introducing this new property.

Example 3.9 We want to show that $Access\_Monitor\_1$ is not *BNDC*. Consider $\Pi = \overline{access\_r}(1,1).\underline{0}$. System $(Access\_Monitor\_1|\Pi) \setminus Act_H$ will be deadlocked immediately after the execution of the read request by Π, blocking in the following 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 process $Access\_Monitor\_1/Act_H$, we find out that $(Access\_Monitor\_1|\Pi) \setminus Act_H \not\approx_B Access\_Monitor\_1/Act_H$, 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 do not send the value to be written. In particular, the high level user $\Pi' = \overline{access\_w}(1,0).\underline{0}$ will deadlock $(Access\_Monitor\_1|\Pi') \setminus Act_H$ immediately after the execution of the write request by Π', blocking in the following state:

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

Again, we have that $(Access\_Monitor\_1|\Pi') \setminus Act_H \not\approx_B Access\_Monitor\_1/Act_H$. 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 follows (see Figure 4):

$$\begin{aligned}
Access_Monitor_5 &\stackrel{\text{def}}{=} (AM \mid Interf) \setminus N \\
AM &\stackrel{\text{def}}{=} (Monitor_5|Object(1,0)|Object(0,0)) \setminus L \\
Monitor_5 &\stackrel{\text{def}}{=} access_r(l,x).(\text{ if } x \leq l \text{ then } r(x,y).\overline{val}(l,y).Monitor_5 \\
&\qquad \text{else } \overline{val}(l,err).Monitor_5) +
\end{aligned}$$

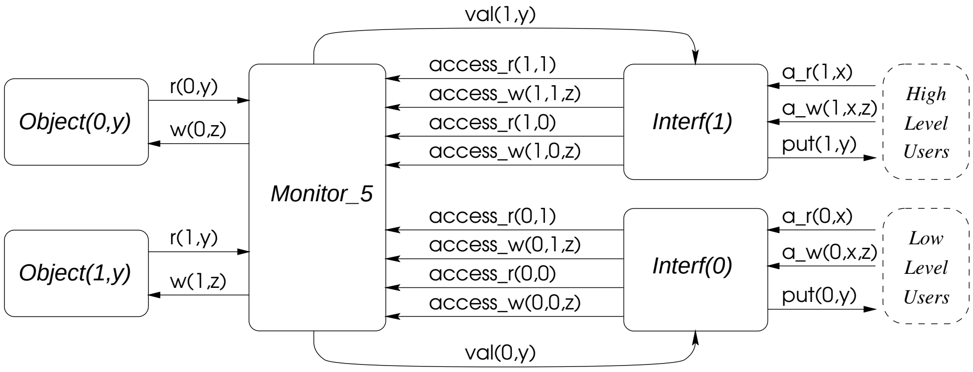

Figure 4. The *BNDC* $Access\_Monitor\_5$

$$
\begin{array}{rcl}
 & & + \, access\_w(l,x,z).(\text{ if } x \geq l \text{ then } \overline{w}(x,z).Monitor\_5 \text{ else } Monitor\_5) \\
Interf & \stackrel{\text{def}}{=} & Interf(0)|Interf(1) \\
Interf(l) & \stackrel{\text{def}}{=} & a\_r(l,x).\overline{access\_r}(l,x).val(l,k).\overline{put}(l,k).Interf(l) + \\
 & & + \, a\_w(l,x,z).\overline{access\_w}(l,x,z).Interf(l)
\end{array}
$$

where $k \in \{0, 1, err\}$, $L = \{r, w\}$, $N = \{val, access\_r, access\_w\}$ and $a\_r(1,x)$, $a\_w(1,x)$, $put(1,y) \in Act_H$ $\forall x \in \{0,1\}$ and $\forall y \in \{0,1,err\}$, while the same actions with 0 as first parameter belong to Act_L. It is possible to verify that $Access\_Monitor\_5$ is *BNDC* indeed. ∎

The above definition of *BNDC* is difficult to use in practice, because of the universal quantification on processes. We need an alternative formulation of *BNDC* which avoids universal quantification, exploiting local information only; however, a solution to this problem is still to be found. Nonetheless, 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 3.10 *(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 (see Appendix A for the proof).

Proposition 3.11 *SBSNNI* $\subset$ *BNDC* ∎

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 minor number of states. In fact, the Concurrency Workbench 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 observational equivalent system F is *BNDC* too. Indeed, the theorem below (see Appendix A for the proof) shows that this is the case, also for all the other security properties we have discussed in this section. For this reason, we imported in CoSeC the minimization procedure of the Concurrency Workbench.

Theorem 3.12 *If $E \approx_B F$, then $E \in X \Leftrightarrow F \in X$, where X can be NNI, SNNI, NDC, BNNI, BSNNI, BNDC, SBSNNI.* ∎

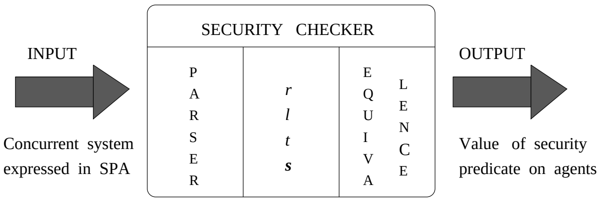

Figure 5. Structure of the CoSeC

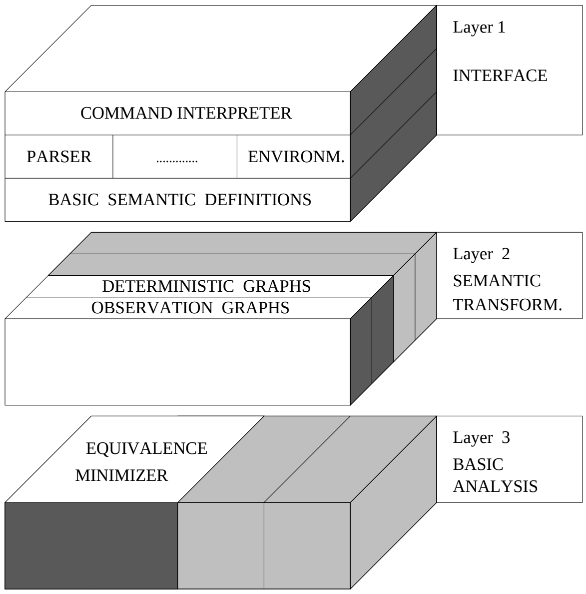

Figure 6. CoSeC Architecture.

4 What is the Compositional Security Checker

4.1 Input-Output

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 5. In detail, the tool is able:

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

4.2 Architecture

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

Figure 6 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 to an agent.

In the semantic layer, CoSeC uses two transformation routines to translate lts's to deterministic and observational graphs [2] respectively. Since they both refer to processes modeled as lts's, they have been imported from CW in CoSeC without any modification.

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 [12] which finds a bisimulation between the roots of two lts's 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.

5 Checking the Security Properties

Now, we want to explain briefly how the system works in evaluating security predicates *NNI, SNNI, NDC, BNNI, BSNNI, SBSNNI*, discussing, at the same time, 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: [8]

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

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

Phase a) (the same for all predicates) CoSeC builds the rlts's 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 rtls due to parallel composition.

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

- **b1:** (for predicates *NNI, SNNI, NDC*) The two rlts's obtained in Phase a) are transformed into deterministic graphs following the classic subset construction (see e.g. [11]). 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 rlts's obtained in Phase a) are transformed into observational 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 [12] 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

8. In the CoSeC the hiding and input restriction operators are respectively represented by ! and ?, for easy of parsing.

equivalences. In particular for the observational equivalence we have that 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 can not 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 7 for more details).

6 Using CoSeC

6.1 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 [9]

`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** $h \; x$

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:

`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. Finally the command **quit** causes an exit to the shell.

6.2 Checking the Access Monitor

In this Section we use CoSeC to automatically check all the versions of the access monitor discussed in Example 3.9. Since CoSeC works on SPA agents we have to translate all the VSPA specifications to SPA. Consider once more $Access\_Monitor\_1$. Table 2 reports the translation of $Access\_Monitor\_1$ specification to the CoSeC syntax for SPA. [10] 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 2 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:` $(\texttt{'val\_h1.Monitor} \mid \texttt{Object\_l1} \mid \texttt{Object\_h1}) \setminus \texttt{L}$

9. 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, sets) inserted by users.

10. 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 2. 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 $Access\_Monitor\_1 \in BSNNI, BNNI$ and $Access\_Monitor\_1 \notin SBSNNI$. Since $SBSNNI \subset BNDC \subset BSNNI, BNNI$, we still don't know if $Access\_Monitor\_1$ 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 deadlock the monitor. In fact, in the state given as output by *SBSNNI*, the monitor is waiting for the high level action `'val_h1`; so, if we find a process Π which leads the system to such a state and does not execute the `val_h1` action, we will have a high level process able to deadlock the monitor. It is sufficient to consider $\Pi = {}'access\_r\_hh.0$. System $(Access\_Monitor\_1|\Pi) \setminus Act_H$ will be deadlocked immediately after the execution of the read request by Π, blocking in the following state

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

(this state differs from the one given as output by *SBSNNI* only for the values stored in objects). It is possible to verify that $Access\_Monitor\_1$ is not *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** $Pi\ \ 'access\_r\_hh.0$
`Command:` **eq**
`Agent:` $(Access\_Monitor\_1 \mid Pi) \setminus acth$
`Agent:` $Access\_Monitor\_1\ !\ acth$
`false`

As we said in Example 3.9, 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 deadlock $Access\_Monitor\_1$, because it executes a write request and does not send the corresponding value. Hence, in Example 3.9 we proposed the modified system $Access\_Monitor\_5$ with an interface for each level and atomic actions for write request and value

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

sending. We finally check that this version of the monitor is *SBSNNI*, hence *BNDC* too:
`Command:` **sbsnni** $Access\_Monitor\_5$
`true`

7 State Explosion and Compositionality

We now want to plain out how the parallel composition operator can increase exponentially the number of states of the system, and then how it can slow down the execution speed of security predicate verification. Let us define in CoSeC the two agents B, D and the set Act_H of high level actions:
`Command:` **bi** B $y.a.b.B + a.b.B$
`Command:` **bi** D $'a.'b.(x.D + D)$
`Command:` **acth** x y
Let us check now if B and D are *SBSNNI* secure:
`Command:` **sbsnni** B
`true`
`Command:` **sbsnni** D
`true`
As we will see, *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 3, which points out the exponential increase of the number of states and the consequent increase of the computation time for verification of *SBSNNI*. The next theorem states that *SBSNNI* is preserved by the parallel operator and the restriction one (see Appendix A for the proof). Similar results hold for *NNI* and *SNNI* [5].

Theorem 7.1 *The following hold:*
(i) $E, F \in SBSNNI \Longrightarrow (E|F) \in SBSNNI$
(ii) $E \in SBSNNI, L \subseteq \mathcal{L} \Longrightarrow E \setminus L \in SBSNNI$ ■

In the following $\mathcal{E}_{FS} \subseteq \mathcal{E}$ will denote the set of (closed and guarded) SPA agents with a finite lts. CoSeC is able to exploit the compositionality of security properties through an algorithmic schema we are going to present. For a certain compositional property P this schema also requires the following condition: if $Z \stackrel{\text{def}}{=} E$ and E is P-secure then also Z is P-secure. This condition is satisfied by all the above presented properties because of Theorem 3.12.

Definition 7.2 *(Compositional Algorithm) Let $P \subseteq \mathcal{E}$ be a set of SPA agents such that*

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

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 $A'_P(E_1) = A'_P(E_2) = \mathbf{true}$ then return $\mathbf{true}$, 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 7.3 *Let $F \in \mathcal{E}_{FS}$ be a finite state SPA agent. 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 guardedness condition over SPA agents. Hence the calculation of $A'_P(F)$ terminates.

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 so-called *nets of automata*:

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

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

Table 4. Verification of *SBSNNI* on $Access\_Monitor\_5$ with the compositional algorithm.

where for every constant Z there must be the corresponding definition $Z \stackrel{\text{def}}{=} p$ (with p guarded). Indeed, every agent of this form possesses a finite state lts. Nonetheless, 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 $D \stackrel{\text{def}}{=} i.(o.\underline{0}|D)$; as the outermost operator is a $+$, the evaluation of $A'_P(B)$ immediately calls for $A_P(B)$, and so it terminates.

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 [11] 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 4 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 checking the system as a whole.[12] Hence, in this case, the compositional technique cannot help in order to reduce 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.

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

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.

Using the compositional algorithm, $Access\_Monitor\_6$ can be checked very efficiently; the verification of property *SBSNNI* takes about 90 seconds (see Table 5). 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

11. This value and all the following are obtained exploiting also the state minimization feature of the tool.
12. The reason why it fails on AM is because AM is essentially $Access\_Monitor\_1$ with atomic writings; hence, it is not *SBSNNI* because of the possible high level deadlock of Example 3.9 caused by synchronous communications. The interface was introduced just in order to make communications asynchronous.

```
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 5. Verification of *SBSNNI* on $Access\_Monitor\_6$ exploiting compositionality.

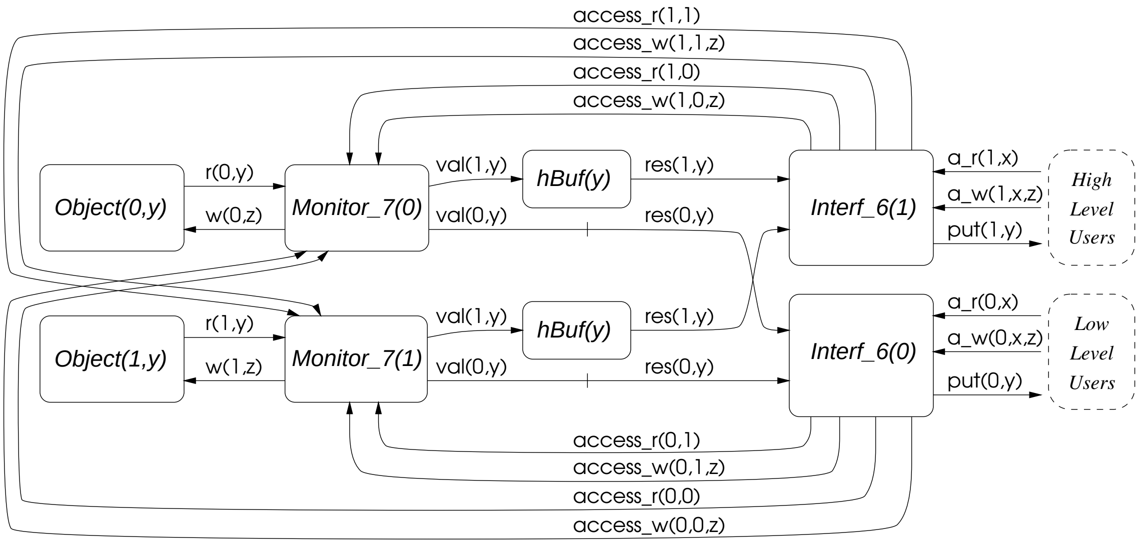

Figure 7. The Modular $Access\_Monitor\_7$

Theorem 3.12, we can conclude that also $Access\_Monitor\_5$ is *SBSNNI*, even if a direct check takes (as we said) about 16 minutes.

$Access\_Monitor\_6$ 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 the more the definition is modular, the more the compositional algorithm is efficient. 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 rather a modular version of the access monitor. The basic idea of this new version (Figure 7) 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.

$$
\begin{aligned}
Access\_Monitor\_7 &\stackrel{\text{def}}{=} (Modh|Modl|Interf\_6) \setminus L \\
Modh &\stackrel{\text{def}}{=} ((Monitor\_7(1)|Object(1,0)|hBuf(empty)) \setminus Lh) \\
&\quad\ [res(0,y)/val(0,y)] \\
Modl &\stackrel{\text{def}}{=} ((Monitor\_7(0)|Object(0,0)|hBuf(empty)) \setminus Lh)
\end{aligned}
$$

```
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 6. Verification of *SBSNNI* on $Access\_Monitor\_7$ exploiting compositionality.

$$\begin{array}{rcl} & & [res(0,y)/val(0,y)] \\ Monitor\_7(x) & \stackrel{\text{def}}{=} & access\_r(l,x).(\text{ if } x \leq l \text{ then } r(x,y).\overline{val}(l,y). \\ & & Monitor\_7(x) \text{ else } \overline{val}(l,err).Monitor\_7(x)) + \\ & & + \, access\_w(l,x,z).(\text{ if } x \geq l \text{ then } \overline{w}(x,z).Monitor\_7(x) \\ & & \text{else } Monitor\_7(x)) \end{array}$$

where $L = \{res, access\_r, access\_w\}$ and $Lh = \{r, w, val(1,y)\}$. Table 6 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 checks that $Access\_Monitor\_7 \approx_B Access\_Monitor\_5$, and so also $Access\_Monitor\_7 \approx_B Access\_Monitor\_6$.

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.

8 Conclusion

8.1 Future Research

Among the possible extensions of the tool, we would like to mention the following:

- (i) A graphic interface: it should help the user when working with CoSeC, avoiding the need of memorizing the technical names.
- (ii) Extension to value-passing SPA: it would be useful that CoSeC accept specification written directly in VSPA. Moreover, one could exploit interesting techniques, such as the one of [9], that could be used to verify the equivalence of finite state value-passing SPA agents which have infinite state SPA translations.
- (iii) Integration with other tools: CoSeC is essentially built on top of CW, but it is not difficult to imagine how to build similar routines on top of any of the available tools for verification of systems represented as labelled transition systems.
- (iv) Addition of new information flow security properties: Even if we think *BNDC* is the right one, any new information flow property that has an algebraic characterization of the form seen in the Introduction can be included easily in CoSeC.

Other future research will be devoted to study the merits of the information flow properties presented here, in particular of *BNDC*. In a forthcoming report [4], we have defined the way to explicitly construct covert channels for all the versions of the Access Monitor that have been proved not *BNDC* in this paper, hence somehow supporting our claim. From a technical point of view, it would be nice to find a static, easily verifiable characterization of *BNDC*, instead of resorting – as we did here – to the stronger *SBSNNI*.

8.2 Related Work

In the literature, there is one relevant paper [18] which introduces some information flow security properties based on the notion of *deterministic views* and shows how to automatically verify them using the CSP model checker FDR [17]. The most interesting property is 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 $E = l.\bar{l_1} + l.\bar{l_2}$ – will be considered not secure. In [3] one of the authors has compared those properties with ours using a failure-equivalence version of *BNDC*, called *FNDC*. The first interesting result is that L-Sec is included in the class of low-deterministic and non-divergent processes. The main result of the paper is that *BNDC* is equal to L-Sec when applied to that particular class of systems. This implies that, for low-deterministic and non-divergent processes, we can use CoSeC to verify L-Sec. Moreover, we can see *BNDC* applied to low-deterministic processes as a *fair* extension of L-Sec to divergent processes.

References

[1] 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.

[2] 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.

[3] 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.

[4] R. Focardi and R. Gorrieri. "A Concrete Criterion for Information Flow Security". Forthcoming.

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

[6] 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.

[7] 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.

[8] 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.

[9] M. Hennessy and H. Lin. "Symbolic Bisimulations". Technical Report CSR1/92, University of Sussex, 1992.

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

[11] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation.*, pages 22–24. Addison-Wesley, 1979.

[12] P. Kanellakis and S.A. Smolka. "CCS Expression, Finite State Processes, and Three Problems of Equivalence". *Information & Computation 86*, pages 43–68, May 1990.

[13] R. Keller. "Formal Verification of Parallel Programs". *Communications of the ACM*, 19 (7):561–572, 1976.

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

[15] Department of Defense. "Integrity in Automated Information Systems". Technical Report C 79-91, National Computer Security Center, September 1991.

[16] G. Plotkin. "A Structural Approach to Operational Semantics". Technical Report DAIMI-FN-19, Aarhus University, 1981.

[17] A. W. Roscoe. "Model Checking CSP". In A. W. Roscoe (ed) *A Classical Mind*. Prentice Hall, 1994.
[18] 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.
[19] 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.
[20] 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.

A Proofs

Proposition 3.11 *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 [14]). 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 $E' \setminus Act_H \stackrel{\hat{\tau}}{\Longrightarrow} E''' \setminus Act_H$ 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*. ■

Theorem 3.12 *If $E \approx_B F$, then $E \in X \Leftrightarrow F \in X$, where X can be NNI, SNNI, NDC, BNNI, BSNNI, BNDC, SBSNNI.*

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 [14]. 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 $F' \setminus_I L \xrightarrow{\mu} F'' \setminus_I L$). ■

Theorem 7.1 *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:

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

PROOF. Consider the relation defined as follows: $((E'|F')/Act_H, E'/Act_H|F'/Act_H) \in R$ where $E \Rightarrow E'$ and $F \Rightarrow F'$. It is easy to prove that R is a bisimulation. Indeed the only non trivial case is the synchronization $(E'|F')/Act_H \xrightarrow{\tau} (E''|F'')/Act_H$ with the corresponding move $E'/Act_H|F'/Act_H \xrightarrow{\tau \cdot \tau} E''/Act_H|F''/Act_H$. ■

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 A.1, we obtain the thesis. We consider the only non trivial case: $(E'|F') \setminus Act_H \xrightarrow{\tau} (E''|F'') \setminus Act_H$ with $E' \xrightarrow{h} E''$ and $F' \xrightarrow{\bar{h}} F''$. Since $E'/Act_H \xrightarrow{\tau} E''/Act_H$ 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 $(E'/Act_H) \setminus L \xrightarrow{\tau} (E''/Act_H) \setminus L$ with $E' \xrightarrow{h} E''$ 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 $(E'/Act_H) \setminus L \xrightarrow{\tau} (E''/Act_H) \setminus L$ 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 $\approx_B$. ■