Partial model checking, process algebra operators and satisfiability procedures for (automatically) enforcing security properties *

Fabio Martinelli[1], Ilaria Matteucci[1,2]
Istituto di Informatica e Telematica - C.N.R., Pisa, Italy[1]
{Fabio.Martinelli, Ilaria.Matteucci}@iit.cnr.it
Dipartimento di Matematica, Università degli Studi di Siena[2]

Abstract

In this paper we show how the partial model checking approach for the analysis of secure systems may also be useful for enforcing security properties. We define a set of process algebra operators that act as programmable controllers of possibly insecure components. The program of these controllers may be automatically obtained through the usage of satisfiability procedures for a variant of μ-calculus.

1 Overview

Many approaches for the analysis of security properties have been successfully developed in the last two decades. An interesting one is based on the idea that potential attackers should be analyzed as if they were un-specified components of a system; thus reducing security analysis to the analysis of *open* systems [11, 12, 14].

More recently there has been also interest on mechanisms and techniques to enforce security properties. A notable example is the security automata in [19] and some extensions proposed in [9].

The paradigm of analysis of security as analysis of open systems has been extended to cope with security protocols [14], fault tolerance [7] and recently access control based on trust management [15]. In this paper we enrich this theory with a method for (automatically) enforcing several security properties.

Basically, we define a set of process algebra operators. They act as programmable controllers of a component that must be managed in order to guarantee that the overall system satisfies a given security policy. Also, we develop a technique to automatically synthesize the appropriate controllers. This represent a significant contribution w.r.t. to the previous work in [9, 19], where this issue was not addressed. The synthesis is based on a satisfiability procedure for the μ-calculus.

Moreover, under certain hypothesis on the observation power of the enforcing controllers, we are able to enforce some non-interference properties (for finite-state systems) that were not intentionally addressed in [19], due to the specific assumptions they had on the enforcing mechanisms.

Our logical approach is also able to cope with composition problems, that have been considered as an interesting and challenging issue in [3].

This paper is organized as follows. Section 2 recalls the basic theory about the analysis of security properties, especially non-interference as properties of open systems. Section 3 explains our approach and Section 4 extends it to manage several kinds of enforcement mechanisms. Section 5 illustrates an example. Section 6 presents a discussion on related work and eventually Section 7 concludes the paper.

*Work partially supported by CNR project "Trusted e-services for dynamic coalitions" and by a CREATE-NET grant for the project "Quality of Protection (QoP)". A full version of this paper with the proofs appears as Technical report of IIT-CNR [16].

2 Background

In this section we briefly recall some technical machinery used in our approach and also a logical approach for dealing with information flow properties (and security properties in general).

2.1 A language for describing concurrent and distributed systems

The *Security Process Algebra* (SPA) [6] is used to describe concurrent and distributed systems and is derived from CCS process algebra of R. Milner [17]. The syntax of SPA is the following:

$$E ::= \mathbf{0} \mid \alpha.E \mid E_1 + E_2 \mid E_1 \| E_2 \mid E \backslash L \mid Z$$

where α is an action in Act, $L \subseteq \mathcal{L}$ and Z is a process constant that must be associated with a definition $Z \doteq E$. As usual, constants are assumed to be *guarded* [17], i.e. they must be in the scope of some prefix operator $\alpha.E'$. The set of SPA *processes* (i.e., terms with guarded constants), is denoted with $\mathcal{E}$, ranged over by $E, F, P, Q \ldots$. We will often use some common syntactic simplifications, e.g., omission of trailing $\mathbf{0}$'s as well as omission of brackets on restriction on a single action. $Sort(E)$ is used to denote the set of actions that occurs in the term E.

SPA operators have the following informal meaning:

- $\mathbf{0}$ is a process that does nothing;
- $\alpha.E$ is a process that can perform an α action and then behaves as E;
- $E_1 + E_2$ (*choice*) represents the nondeterministic choice between the two processes E_1 and E_2;
- $E_1 \| E_2$ (*parallel*) is the parallel composition of two processes that can proceed in an asynchronous way , synchronizing on complementary actions, represented by an internal action τ, to perform a communication.
- $E \backslash L$ (*restriction*) is the process E when actions in $L \cup \overline{L}$ are prevented.

The operational semantics of SPA terms is given in terms of Labeled Transitions Systems (LTS).

Definition 2.1 *A labeled transition system $(\mathcal{E}, \mathcal{T})$ (LTS) of concurrent processes over Act has the process expressions $\mathcal{E}$ as its states, and its transitions $\mathcal{T}$ are exactly which can be inferred from the transition rules for processes.*

The interested reader may find the formal definition of the semantics below:

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

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 \| E_2 \xrightarrow{a} E_1' \| E_2} \qquad \frac{E_2 \xrightarrow{a} E_2'}{E_1 \| E_2 \xrightarrow{a} E_1 \| E_2'} \qquad \frac{E_1 \xrightarrow{l} E_1' \quad E_2 \xrightarrow{\bar{l}} E_2'}{E_1 \| E_2 \xrightarrow{\tau} E_1' \| E_2'}$$

$$\frac{Z \doteq E \quad E \xrightarrow{\alpha} E'}{Z \xrightarrow{\alpha} E'} \qquad \frac{E_1 \xrightarrow{\alpha} E_1'}{E_1 \backslash L \xrightarrow{\alpha} E_1' \backslash L} (\alpha \notin L \cup \overline{L})$$

2.2 Strong and weak bisimulations

It is often necessary to compare processes that are expressed using different terms but have the same behavior. We recall some useful relations on processes.

Definition 2.2 *Let $(\mathcal{E}, \mathcal{T})$ be an LTS of concurrent processes, and let $\mathcal{R}$ be a binary relation over $\mathcal{E}$. Then $\mathcal{R}$ is called* strong simulation *(denoted by $\prec$) over $(\mathcal{E}, \mathcal{T})$ if and only if, whenever $(E, F) \in \mathcal{R}$ we have:*

$$\text{if } E \xrightarrow{a} E' \text{ then there exists } F' \in \mathcal{E} \text{ s. t. } F \xrightarrow{a} F' \text{ and } (E', F') \in \mathcal{R}$$

Now, we can define *strong bisimulation*:

Definition 2.3 *A binary relation $\mathcal{R}$ over $\mathcal{E}$ is said a* strong bisimulation *(denoted by $\sim$) over the LTS of concurrent processes $(\mathcal{E}, \mathcal{T})$ if both $\mathcal{R}$ and its converse are strong simulation.*

Another kind of bisimulation is the *weak bisimulation*. This relation is used when there is the necessity of understanding if systems with different internal structure - and hence different internal behavior - have the same external behavior and may thus be considered observationally equivalent. The notion of *observational relations* is the follow: $E \stackrel{\tau}{\Rightarrow} E'$ (or $E \Rightarrow E'$) if $E \xrightarrow{\tau}{}^{*} E'$ (where $\xrightarrow{\tau}{}^{*}$ is the reflexive and transitive closure of the $\xrightarrow{\tau}$ relation); for $a \neq \tau$, $E \stackrel{a}{\Rightarrow} E'$ if $E \stackrel{\tau}{\Rightarrow} \xrightarrow{a} \stackrel{\tau}{\Rightarrow} E'$. Let $DerE$ be the set of derivatives of E, i.e., the set of process that can be reached through the transition relations. Now we are able to give the two following definitions.

Definition 2.4 *Let $\mathcal{R}$ be a binary relation over a set of process $\mathcal{E}$. Then $\mathcal{R}$ is said to be a* weak simulation *(denoted by $\precsim$) if, whenever $(E, F) \in \mathcal{R}$,*

$$\text{if } E \xrightarrow{a} E' \text{ then there exists } F' \in \mathcal{E} \text{ s. t. } F \stackrel{a}{\Rightarrow} F' \text{ and } (E', F') \in \mathcal{R}.$$

Definition 2.5 *A binary relation $\mathcal{R}$ over $\mathcal{E}$ is said a* weak bisimulation *($\approx$) over the LTS of concurrent processes $(\mathcal{E}, \mathcal{T})$ if both $\mathcal{R}$ and its converse are weak simulation.*

Every strong simulation is also a weak one (see [17]).

2.3 Equational μ-calculus

Modal μ-calculus is a process logic well suited for specification and verification of systems whose behavior is naturally described using state changes by means of actions. It is a normal modal logic K augmented with recursion operators. It permits to express a lot of interesting properties like *safety* and *liveness* properties, as well as allowing us to express equivalence conditions over LTS.

In equational μ-calculus recursion operators are replaced by fixpoint equations. This permits to recursively define the properties of a given systems.

We use the equational μ-calculus instead of modal μ-calculus because the former is very suitable for partial model checking, that is described later (see [1], [2]).

Let a be in Act and X be a variable ranging over a finite set of variables $Vars$.

Given the grammar:

$$A ::= X \mid \mathbf{T} \mid \mathbf{F} \mid X_1 \wedge X_2 \mid X_1 \vee X_2 \mid \langle a \rangle X \mid [a]X$$
$$D ::= X =_\nu AD \mid X =_\mu AD \mid \epsilon$$

where the meaning of $\langle a \rangle X$ is 'it is possible to do an a-action to a state where X holds' and the meaning of $[a]X$ is 'for all a-actions that are performed then X holds'. $X =_\nu A$ is a minimal fixpoint equation, where A is an assertion (i.e. a simple modal formula without recursion operator), and $X =_\mu A$ is a maximal fixpoint equation. Roughly, the semantic $[\![D]\!]$ of the list of equations D is the solution of the system of equations corresponding to

D. According to this notation, $[\![D]\!](X)$ is the value of the variable X, and $E \models D \downarrow X$ can be used as a short notation for $E \in [\![D]\!](X)$. The following result can be proved by putting together standard results for decision procedures for μ-calculus (see [20]).

Theorem 2.1 *Given a formula γ it is possible to decide in exponential time in the length of γ if there exists a model of γ and it is also possible to give an example of it.*

2.4 Partial model checking

Partial model checking (*pmc*) is a technique that was originally developed for compositional analysis of concurrent systems (processes) (see [2]). The intuitive idea underlying the pmc is the following: proving that $E\|F$ satisfies a formula ϕ is equivalent to prove that F satisfies a modified specification $\phi_{//_E}$, where $//_E$ is the partial evaluation function for the parallel composition operator(see [2]).In formula:

$$E\|F \models \phi \tag{1}$$

In order to describe how pmc function acts, we discuss, for instance, the partial evaluation rules for the formula $\langle\tau\rangle A$ w.r.t. the $\|$ operator. By inspecting the inference rules, we can note that the process $E\|F$ (with F unspecified component) can perform a τ action by exploiting one of the three possibilities:

- the process F performs an action τ going in a state F' and $E\|F'$ satisfies A; this is taken into account by the formula $\langle\tau\rangle(A_{//_E})$;
- the process E performs an action τ going in a state E' and $E'\|F$ satisfies A and this is considered by the disjunctions $\bigvee_{E \xrightarrow{\tau} E'} A_{//_{E'}}$, where every formula $A_{//_{E'}}$ takes into account the behavior of F in composition with a τ successor of E;
- the last possibility is that the τ action is due to the performing of two complementary actions by the two processes. So for every a-successor E' of E there is a formula $\langle\bar{a}\rangle(A_{//_{E'}})$.

With partial model checking we can reduce the previous property to:

$$F \models \phi_{//E} \tag{2}$$

Lemma 2.1 *Given a process $E\|F$ and a formula ϕ we have:*

$$E\|F \models \phi \textit{ iff } F \models \phi_{//E}$$

A similar lemma holds for every operator of SPA (see [1]).

In this way, it can be noticed that the reduced formula $\phi_{//E}$ depends only on the formula ϕ and on process E. No information is required on the process F which can represent a possible enemy. Thus, given a certain system E, it is possible to find the property that the enemy must satisfy in order to make a successful attack on the system. It is worth noticing that partial model checking functions may be automatically derived from the semantics rules used to define a language semantics (Structured Operational Semantics). Thus, the proposed technique is very

flexible. Here, we give the *pmc* function for parallel operator (that can be also found in [1, 2]).

$$\begin{array}{rcl}
(D{\downarrow}X)_{//t} & = & (D_{//t}){\downarrow}X_t \\
\epsilon_{//t} & = & \epsilon \\
(X =_\sigma AD)_{//t} & = & ((X_s =_\sigma A_{//s})_{s\in Der(t)})(D)_{//t} \\
X_{//t} & = & X_t \\
[a]\,A_{//s} & = & [a](A_{//s}){\wedge}\bigwedge_{s\xrightarrow{a}s'}A_{//s'} \text{ if } a{\neq}\tau \\
[\tau]\,A_{//s} & = & [\tau](A_{//s}){\wedge}\bigwedge_{s\xrightarrow{\tau}s'}A_{//s'}{\wedge}\bigwedge_{s\xrightarrow{a}s'}[\bar{a}](A_{//s'}) \\
(A_1{\wedge}A_2)_{//s} & = & ((A_1)_{//s}){\wedge}((A_2)_{//s}) \\
\mathbf{T}_{//s} & = & \mathbf{T}
\end{array}$$

2.5 Characteristic formulae

A *characteristic formula* is a formula in equational μ-calculus that completely characterizes the behavior of a (state in a) state-transition graph modulo a chosen notion of behavioral relation. It is possible to define the notion of characteristic formula for a given finite state process E w.r.t. weak bisimulation as follows (see [18]).

Definition 2.6 *Given a finite state process E, its characteristic formula (w.r.t. weak bisimulation) $D_E \downarrow X_E$ is defined by the following equations for every $E' \in Der(E)$, $a \in Act$:*

$$X_{E'} =_\nu \Big(\bigwedge_{a;E'':E'\xrightarrow{a}E''} \langle\langle a\rangle\rangle X_{E''}\Big) \wedge \Big(\bigwedge_{a}\Big([a]\Big(\bigvee_{E'':E'\stackrel{a}{\Rightarrow}E''} X_{E''}\Big)\Big)\Big)$$

where $\langle\langle a\rangle\rangle$ of the modality $\langle a\rangle$ which can be introduce as abbreviation (see [18]):

$$\langle\langle\epsilon\rangle\rangle\phi \stackrel{def}{=} \mu X.\phi \vee \langle\tau\rangle X \quad \langle\langle a\rangle\rangle\phi \stackrel{def}{=} \langle\langle\epsilon\rangle\rangle\langle a\rangle\langle\langle\epsilon\rangle\rangle\phi$$

The following lemma characterizes the power of these formulae.

Lemma 2.2 *Let E_1 and E_2 be two different finite-state processes. If ϕ_{E_2} is characteristic for E_2 then:*

1. *If $E_1 \approx E_2$ then $E_1 \models \phi_{E_2}$*
2. *If $E_1 \models \phi_{E_2}$ and E_1 is finite-state then $E_1 \approx E_2$.*

2.6 A logical approach for specifying and analyzing information flow properties

Information flow is a main topic in the theoretical study of computer security. We can find several formal definitions in the literature (see [10]). To describe this problem, we can consider two users, *High* and *Low* interacting with the same computer system. We ask if there is any flow of information from *High* to *Low*. The central property is the *Non Deducibility on composition* (*NDC*, see [6]): the low level users cannot infer the behavior of the high level user from the system because for the low level users the system is always the same. This idea can be represented as follow:

$$\forall \Pi \in \text{ High users } E \mid \Pi \equiv E \text{ w.r.t. Low users}$$

(where $\mid$ represents a suitable composition operator.) We study this property in term of *SPA* parallel composition operator and *bisimulation* equivalence.

We denote with $BNDC$ a security property called *Bisimulation Non Deducibility on Compositions* (see [6]).

Definition 2.7 *Let $\mathcal{E}_H = \{\Pi \mid Sort(\Pi) \subseteq H \cup \{\tau\}\}$ be the set of High users. $E \in BNDC$ if and only if $\forall \Pi \in \mathcal{E}_H$ we have $(E\|\Pi)\backslash H \approx E\backslash H$.*

By using the characteristic formula ϕ of the process $E\backslash H$, we may express information flow property in a logical way.

$$E \in BNDC \text{ iff } \forall \Pi \in S : (E\|\Pi)\backslash H \models \phi \tag{3}$$

Partial model checking function gives we have a method for reducing the verification of the previous property to a validity checking problem in μ-calculus (see [11]). As a matter of fact, the property 4 turns out to be equivalent to

$$E \in BNDC \text{ iff } \forall \Pi \in S : \Pi \models \phi' \tag{4}$$

where ϕ' is the formula obtained from ϕ after *pmc* w.r.t the process E (and the restriction operator). Thus, due the decidability of the validity problem for μ-calculus we have.

Proposition 2.1 *$BNDC$ is decidable for all finite state processes E.*

Our logical approach has been extended to cope with several security properties. Thus the approach we are going to introduce is applicable to a wide set of security properties.

3 Our approach for enforcing security properties

Let S be a system, and let X be one component that may be dynamically changed (e.g., a downloaded mobile agent). We say that the system $S\|X$ enjoys a security property expressed by a logical formula ϕ if and only if for every behavior of the component X, the behavior of the system S enjoys that security property:

$$\forall X (S\|X)\backslash H \models \phi \tag{5}$$

where $H = Sort(X)$.

By using the partial model checking approach proposed in [12], we can focus on the properties of the possibly un-trusted component X, i.e.:

$$\forall X \quad X \models \phi_{S,\backslash H} \tag{6}$$

Thus, we may study whether a potential enemy could exists and, in particular, which are necessary and sufficient conditions that an enemy should satisfy for the purpose to alter the correct behavior of the system.

In order to protect the system we may simply check each process X before executing it or, if we do not have this possibility, we may define a controller that in any case forces it to behave correctly.

We may distinguish several situations[1] depending on the control one may have on the process X:

1. if X performs an action we may detect and intercept it;
2. in addition to 1), it is possible to know which are the possible next steps of X;
3. X whole code is known and we are able to model check it[2].

In the scenarios 1) and 2) we may imagine to develop some controllers that force the intruder to behave correctly, i.e. as prescribed by the formula $\phi_{S,\backslash H}$.

[1] The last two pose several decidability issues.

[2] We do not consider here the possibility of manipulate the code.

3.1 Enforcing security properties with programmable controllers

We wish to provide a framework where we are able to enforce specific security properties defining a new operator, say $Y \triangleright^* X$, that can permit to control the behavior of the component X, given the behavior of a control program Y.

Example 3.1 *Let E and F be two processes, and let $a \in Act$ be an action. We define a new operator $\triangleright'$ (controller operator) by these two rules:*

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E \triangleright' F \xrightarrow{a} E' \triangleright' F'} \tag{7}$$

$$\frac{E \xrightarrow{a} E'}{E \triangleright' F \xrightarrow{a} E' \triangleright' F} \tag{8}$$

This operator forces the system to make always the right action also if we do not know what action the agent X is going to perform.

Eventually, we would like that the overall system $S \| (Y \triangleright^* X)$ always enjoys the desired security properties regardless of the behavior of the component X. Thus, we want to find a control program Y such that:

$$\forall X (S \| Y \triangleright^* X) \backslash H \models \phi \tag{9}$$

Equivalently, by *pmc*, we get:

$$\exists Y \, \forall X \, (Y \triangleright^* X) \models \phi' \tag{10}$$

where $\phi' = \phi_{//(S,\backslash H)}$.

Note that differently from other approaches the control target and the controller are expressed in a similar formalism.

While the equation 10 should be the property to manage, it might not be easy. However, we note that if the controller operator satisfies the following additional property

Assumption 3.1 *For every X and Y, we have:*

$$Y \triangleright^* X \sim Y$$

then the property 10 is equivalent to:

$$\exists Y \; Y \models \phi' \tag{11}$$

As a matter of fact, the previous assumption permits us to conclude that $Y \triangleright^* X$ and Y are strongly equivalent on so they satisfy the same formulas. The formulation 11 is easier to be managed.

Refer to example 3.1, we are able to prove that the operator $\triangleright'$ enjoys Assumption 3.1.

Proposition 3.1 *The operator $\triangleright'$ enjoys Assumption 3.1.*

Note that for some properties, e.g. BNDC, it is sufficient that $Y \triangleright^* X$ and Y are weakly bisimilar. According to definition of weak bisimulation, $Y \triangleright^* X \approx X$ (since every strong simulation is also a weak one [17]) and thus it could be applied to enforce information flow properties (although in the scenario 1) it would not be very useful, since it could often override the high user instructions).

While designing such a process Y could not be difficult in principle, we can take advantage of our logical approach and obtain an automated procedure as follows.

3.2 Automated synthesis of controllers

In this subsection, we discuss how it is possible to find a program controller Y that is a model of ϕ', the formula in 11.

As a matter of fact, our logical approach is very useful.

The formula ϕ' is a μ-calculus formula, so, referring to the theorem 2.1, it is possible to decide if there exists a model of such ϕ'. The procedure returns also a model that will be our program for our controllers.

Unfortunately, the satisfiability procedure has complexity that is, in the worst case, exponential in the size of the formula.

3.3 Composition of properties

Our logical approach is able to struggle successfully with composition problems. If we should force many different security policies, we have only to force the conjunction of this policies. In formulas: let $\phi_1, \cdots, \phi_n$ be n different security policies, S be our system and X be an external agent, we have:

$$\forall X(S \| X) \backslash H \models \phi_1 \quad \ldots \quad \forall X(S \| X) \backslash H \models \phi_n$$

The following step to solve is reduce this n proposition to one in the following way:

$$\forall X(S \| X) \backslash H \models \bigwedge_{i=1,\cdots,n} \phi_i \tag{12}$$

If we assume $\bigwedge_{i=1,\cdots n} \phi_i = \phi$, we have the same situation that we have described by the formula 10.

4 Other controllers

We can define other controller operators as follows.

The controller $\triangleright''$ have two rules:

$$\frac{E \xrightarrow{a} E' \; F \xrightarrow{a} F'}{E \triangleright'' F \xrightarrow{a} E' \triangleright'' F'} \tag{13}$$

$$\frac{E \xrightarrow{a} E' \; F \overset{a}{\not\rightarrow} F'}{E \triangleright'' F \xrightarrow{a} E' \triangleright'' F} \tag{14}$$

This controller is the most complete: if the program E and the target F agree on the next action both can do it in a lock step, if F does not have a correct behavior, the process E issues an action, so the system maintains a correct behavior. Being able to give priorities to rule applications, definitely the first rule should have higher priority than the second one.

The following result holds.

Proposition 4.1 *The preposition 3.1 holds also for two operator: $\triangleright'$ and $\triangleright''$.*

Another interesting operator is described by the following rule:

$$\frac{E \xrightarrow{a} E' \; F \xrightarrow{a} F'}{E \triangleright''' F \xrightarrow{a} E' \triangleright''' F'} \tag{15}$$

However, it is useful to note that for this operator a weaker proposition holds.

Proposition 4.2 *Between $Y \triangleright''' X$ and Y holds the following relations:*

$$Y \triangleright''' X \prec Y$$

i.e. $Y \triangleright''' X$ and Y are strong similar but not bisimilar.

As a matter of fact, with this operator, we can ensure that the system is secure only w.r.t. security properties that are safety properties. Such properties are preserved under weak simulation (e.g. see [7]). Thus, we cannot enforce liveness properties through this controller.

4.1 Feasibility issues for our controllers

The introduction of a controller operator helps to guarantee a correct behavior of the entire system.

We discuss in this subsection, how and also if, these controllers ($\triangleright'$, $\triangleright''$ and $\triangleright'''$) can be effectively implemented.

However, the actual feasibility of these controllers depends on the scenarios we consider. In particular, we focus on scenarios 1) and 2).

For the first controller operator, $\triangleright'$, we can note that this operator need to check the next action or it can directly execute one correct action. Thus, it would be easily implementable in all the two scenarios.

The operator $\triangleright''$ cannot be implemented in the scenario 1): if we cannot decide a priori which are the possible next steps that the external agent is not able to perform, we cannot implement the second rule (14). In the scenario 2), such an operator would be implementable. It would be also possible in the scenario 2), if we could also know whether X is forced to make a specific action, to give priority to the first rule in order to allow always the correct actions of the target. Thus, controller $\triangleright''$ would be the most appropriate in this scenario.

The last controller operator can be implemented in any scenarios. As a matter of fact, it coincides with the monitors defined in [19].

5 A simple example

Consider the process $E = l.\mathbf{0} + h.h.l.\mathbf{0}$. The system E where no high level activity is present is weakly bisimilar to $l.\mathbf{0}$.

Consider the following equational definition (please note that F is a variable here):

$$F =_\nu ([\tau]F) \wedge [l]\mathbf{T} \wedge \langle\langle l \rangle\rangle \mathbf{T}$$

It asserts that a process may and must perform the visible action l.

As for the study of $BNDC$-like properties we can apply the partial evaluation for the parallel operator we obtain after some simplifications:

$$F_E \quad =_\nu \quad ([\tau]F_E) \wedge [\overline{h}]\langle\langle \overline{h} \rangle\rangle \mathbf{T}$$

which, roughly, expresses that after performing a visible $\overline{h}$ action, the system reaches a configuration s.t. it must perform another visible $\overline{h}$ action.

The information obtained through partial model checking can be used to enforce a security policy which prevents a system from having certain information leaks. In particular, if we use the definition of the controller as $\triangleright''$, we simply need to find a process that is a model for the previous formula, say $Y = \overline{h}.\overline{h}.\mathbf{0}$.

Then, for any component X, we have $(E \| (Y \triangleright'' X)) \setminus \{h\}$ satisfies F.

For instance, consider $X = \overline{h}.\mathbf{0}$. The system

$$(E \| (Y \triangleright'' X)) \setminus \{h\} \xrightarrow{\tau} (h.l.\mathbf{0} \| (\overline{h} \triangleright'' \mathbf{0})) \setminus \{h\}$$

Thus, using the second rule the controller may force to issue another $\overline{h}$ and thus we eventually get

$$(h.l.\mathbf{0} \| (\overline{h} \triangleright'' \mathbf{0})) \setminus \{h\} \xrightarrow{\tau} (l.\mathbf{0} \| (\mathbf{0} \triangleright'' \mathbf{0})) \setminus \{h\} \approx l.\mathbf{0}$$

and so the system still preserve its security since the actions performed by the component X have been prevented from being visible outside. On the contrary, if the controller would not be present, there would be a deadlock after the first internal action.

6 Discussion on related work

In [13], we presented preliminary work based on different techniques for automatically synthesizing systems enjoying a very strong security property, i.e. SBSNNI (e.g., see [6]). That work did not deal with controllers.

Much of prior work is about the study of enforceable properties and related mechanisms.

In [19], Schneider deals with enforceable security properties in a systematic way. He discusses whether a given property is enforceable and at what cost. To study those questions, Schneider uses the class of enforceable mechanisms (EM) that work by monitoring execution steps of some system, herein called the *target*, and terminating the target's execution if it is about to violate the security property being enforced. The author asserts there isn't any EM (Execution Monitoring) that can enforce information flow because it can't be formalized like a safety property. The security automata defined in [19] have the follow behavior:

- If the automaton can make a transition on given input symbol, then the target is allowed to perform that step. The state of the automaton changes according to the transition rules.
- otherwise the target is terminated and we can deduce that security property can be violated.

He explicitly assumes to be in the scenario that we call 1).

We can note that our controller operator, $\triangleright'''$, have the same behavior of the security automata for enforcement that Schneider defines in his article.

The operator $\triangleright'''$ have only the following rule:

$$\frac{E \xrightarrow{a} E' \; F \xrightarrow{a} F'}{E \triangleright''' F \xrightarrow{a} E' \triangleright''' F}$$

Roughly speaking, if process F does the correct action then $E \triangleright''' F$ does a correct transaction else the system stops.

This fact is very important because, as we say in the proposition 4.2, $Y \triangleright''' X$ and Y are strongly similar but not bisimilar. So this two processes are not strongly equivalent and they don't satisfy all the same formulas. So, also with our formalism, we can not enforce information flow with this operator.

We can however define an operator in scenario 1) that enforces information flow property. The cost of this operation is that the behavior of the controller component may be completely neglected. Thus, from a practical point of view, our operator is not very useful.

However, we may notice that our work is a contribution w.r.t. the work of Schneider since it allows the automatic construction of the correct monitor.

Also in [3, 9] there is the idea that information flow can not be forced by an automaton. In both of these articles, many types of automata are illustrated. All of them are in the scenario 1). The automata waits for an action of the target. In particular, in [9] there are four different automata:

truncation automata it can recognize bad sequences of actions and halt program execution before the security property is violated, but cannot otherwise modify program behavior. These automata are similar to Schneider's original security monitor;

suppression automata in addition to being able to halt program execution, it has the ability to suppress individual program actions without terminating the program outright;

insertion automata it is able to insert a sequence of actions into the program action stream as well as terminate the program;

edit automata it combines the powers of suppression and insertion automata. It is able to truncate action sequences and insert or suppress security-relevant actions at will.

The interested reader may find in the full version of this paper (see [16]) the description of process algebras operators that mimic as such automata. (Since that truncation automata is the same automata is described in [19], we already defined a controller operator which has the same behavior.)

We use controller synthesis in order to force a system to verify security policy. The synthesis of controllers is, however, studied also in other research areas. We describe here two papers that deal with synthesize of controller in real-time.

In [4] the author describes an algorithm for synthesize controller from real-time specification. He presents an algorithm for specified in a subset of the internal temporal logic Duration calculus. The synthesized controllers are given as PLC-Automata. These are an abstract representation of a machine that periodically polls the input and has the possibility of measuring time.

In [5] the authors tackles the following problem: given a timed automaton restrict its transition relation in a systematic way so that all remaining behaviors satisfy certain properties. The problem is formulated using the notion of real-time game. A strategy for a given game is a rule that tells the controller how to choose between several possible actions in any game position.A strategy is winning if the controller, by following these rules, always wins (according to a given definition of winning) no matter what the environment does. There is the definition of Game automata and the authors gives a relation and using this relation is able to define a winning strategy for the game.

7 Conclusion and future work

We illustrated some preliminary results towards a uniform theory for enforcing security properties. With this work, we contribute to extend a framework based on process calculi and logical techniques that have been shown to be very suitable to model and verify several security properties. With respect to prior work, we also add the possibility to automatically build enforcing mechanisms.

Much work need to be done in order to make our approach more feasible in practice. We argue that there are many security properties whose corresponding controller may be built more efficiently. For instance, there are some cases in which the complexity of satisfiability problem is linear in the size of the formula (e.g., see [8]).

We argue that extending our approach to consider timed security properties should be possible and worth of investigation.

8 Acknowledgement

We thank the anonymous referees of FCS05 for valuable comments that helped us to improve this paper.

References

[1] H. Andersen. *Verification of Temporal Properties of Concurrent Systems*. PhD thesis, Department of Computer Science, Aarhus University, Denmark, June 1993.

[2] H. R. Andersen. Partial model checking. In *LICS '95: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, page 398. IEEE Computer Society, 1995.

[3] L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In I. Cervesato, editor, *Foundations of Computer Security: proceedings of the FLoC'02 workshop on Foundations of Computer Security*, pages 95–104, Copenhagen, Denmark, 25–26 July 2002. DIKU Technical Report.

[4] H. Dierks. Synthesising controllers from real-time specifications. In *ISSS '97: Proceedings of the 10th international symposium on System synthesis*, pages 126–133, Washington, DC, USA, 1997. IEEE Computer Society.

[5] A. P. E. Asarin, O. Maler and J. Sifakis. Controller synthesis for timed automata. In *Proc. System Structure and Control*. Elsevier, 1998.

[6] R. Focardi and R.Gorrieri. A classification of security properties. *Journal of Computer Security*, 3(1):5–33, 1997.

[7] S. Gnesi, G. Lenzini, and F. Martinelli. Logical specification and analysis of fault tolerant systems through partial model checking. *International Workshop on Software Verification and Validation (SVV), ENTCS.*, 2004.

[8] D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *Proc. of the 20th International Foundations of Computer Science 1995 (MFCS)*, pages 552–5662, Prague, 1995.

[9] J. Ligatti, L. Bauer, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1–2):2–16, Feb. 2005.

[10] G. Lowe. Semantic models for information flow. *Theor. Comput. Sci.*, 315(1):209–256, 2004.

[11] F. Martinelli. *Formal Methods for the Analysis of Open Systems with Applications to Security Properties*. PhD thesis, University of Siena, Dec. 1998.

[12] F. Martinelli. Partial model checking and theorem proving for ensuring security properties. In *CSFW '98: Proceedings of the 11th IEEE Computer Security Foundations Workshop*, page 44. IEEE Computer Society, 1998.

[13] F. Martinelli. Towards automatic synthesis of systems without informations leaks. In *Proceedings of Workshop in Issues in Theory of Security (WITS)*, 2000.

[14] F. Martinelli. Analysis of security protocols as *open* systems. *Theoretical Computer Science*, 290(1):1057–1106, 2003.

[15] F. Martinelli. Towards an integrated formal analysis for security and trust. *FMOODS 2005*, LNCS 3535, 2005.

[16] F. Martinelli and I. Matteucci. Partial model checking, process algebra operators and satisfiability procedures for (automatically) enforcing security properties. Technical report, IIT-CNR, March 2005.

[17] R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[18] M. Müller-Olm. Derivation of characteristic formulae. In *MFCS'98 Workshop on Concurrency*, volume 18 of *Electronic Notes in Theoretical Computer Science (ENTCS)*. Elsevier Science B.V., August 1998. 12 pages, MFCS'98 Workshop on Concurrency.

[19] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[20] R. S. Street and E. A. Emerson. An automata theoretic procedure for the propositional μ-calculus. *Information and Computation*, 81(3):249–264, 1989.