Through modeling to synthesis of security automata [1]

Fabio Martinelli[2]

Istituto di Informatica e Telematica - C.N.R., Pisa, Italy

Ilaria Matteucci[3]

Istituto di Informatica e Telematica - C.N.R., Pisa, Italy
Dipartimento di Scienze Matematiche ed Informatiche, Università degli Studi di Siena

Abstract

We define a set of process algebra operators, that we call *controller operators*, able to mimic the behavior of security automata introduced by Schneider in [17] and by Ligatti and al. in [3]. Security automata are mechanisms for enforcing security policies that specify acceptable executions of programs.
Here we give the semantics of four controllers that act by monitoring possible un-trusted component of a system in order to enforce certain security policies. Moreover, exploiting satisfiability results for temporal logic, we show how to automatically build these controllers for a given security policy.

Keywords: partial model checking, safety properties, automated synthesis of controllers.

1 Overview

Recently, several papers tackled the formal definition of mechanisms for enforcing *security policies* (e.g., see [2,3,6,10,11,17]). A *security policy* specifies acceptable executions of programs. Examples of security policies are *information flow, availability, access control* and so on (see [17]).

The focus of this paper is the study of *enforcement mechanisms* introduced by Schneider in [17] and *security automata* developed by Ligatti and al. in [3,6]. Security automata monitor execution steps of some system, herein called the *target*, and terminate the target's execution if it is about to violate the security policy being enforced.

Here we model these security automata by *process algebra operators* (see [12]), acting as *controller operators*. We propose a logical approach to the problem of monitoring systems in order to enjoy security policies. As matter of fact, we express security policies by a temporal logic formula and we exploit a huge theory of process algebra and temporal logic in order to synthesize controller operators.

[1] Work partially supported by CNR project “Trusted e-services for dynamic coalitions” and by EU-funded project “Software Engineering for Service-Oriented Overlay Computers”(SENSORIA) and by EU-funded project “Secure Software and Services for Mobile Systems ”(S3MS).

[2] Email: Fabio.Martinelli@iit.cnr.it

[3] Email: Ilaria.Matteucci@iit.cnr.it

In [17], Schneider defined *security automata* as a triple $(\mathcal{Q}, q_0, \delta)$ where $\mathcal{Q}$ is a set of states, q_0 is the initial state and, being Act the set of security-relevant actions, $\delta : Act \times \mathcal{Q} \rightarrow 2^{\mathcal{Q}}$ is the transition function. A security automaton processes a sequence of actions $a_1 a_2 \ldots$ one by one. For each action, the current global state $\mathcal{Q}'$ is calculated, by initially starting from $\{q_0\}$. As each a_i is read, the security automaton changes $\mathcal{Q}'$ in $\bigcup_{q \in \mathcal{Q}'} \delta(a_i, q)$. If the automaton can make a transition on a given action, i.e. $\mathcal{Q}'$ is not empty, then the target is allowed to perform that action. The state of the automaton changes according to transition rules. Otherwise the target execution is terminated. A security property that can be enforced in this way corresponds to a *safety property* (according to [17], a property is a safety one, if whenever it does not hold in a trace then it does not hold in any extension of this trace).

Starting from the work of Schneider described above, Ligatti and al. in [3,6] have defined four different kinds of security automata which deal with finite sequences of actions: the **truncation automaton** which can recognize bad sequences of actions and halts program execution before a security property is violated, but cannot otherwise modify program behavior. The behavior of these automata is similar to the behavior of security automata of Schneider. The **suppression automaton** can suppress individual program actions without terminating the program outright in addition to being able to halt program execution. The third automaton is the **insertion automaton**. It is able to insert a sequence of actions into the program actions stream as well as terminate the program. The last one is the **edit automaton**. It combines the power of suppression and insertion automaton hence it is able to truncate actions sequences and can insert or suppress security-relevant actions at will.

In this paper we introduce four process algebra operators $Y \triangleright_{\mathbf{K}} X$, where X is the target, Y is the *program controller*, i.e. the process that controls the behavior of the target, and $\mathbf{K}$ is the name of the corresponding automaton. These operators are able to mimic the behavior of the security automata briefly described above.

In order to express security policies we use μ-calculus formulae because many properties of systems are naturally specified by means of fixed points and it is very expressive.

Exploiting a huge theory for security analysis based on process algebra and using satisfiability procedure for the μ-calculus, we show how to automatically synthesize program controllers Y, depending on the kind of security automata one chooses. Moreover for truncation automata we show a method to build the maximal model.

This work represents a significant contribution to the previous works (see [3,6,7,17]), where the synthesis problem for the security automata was not addressed. In fact, most of the related works deal with the verification rather than with the synthesis problem.

Moreover, other approaches deal with the problem of monitoring the component X to enjoy a given property, by treating it as the whole system of interest. However, often not all the system needs to be checked (or it is simply not convenient to check it as a whole). Some components could be trusted and one would like to have a method to constrain only the un-trusted ones (e.g. downloaded applets). Similarly, it could not be possible to build a reference monitor for a whole distributed architecture, while it could be possible to have it for some of its components.

In our approach we actually start from a property ϕ that a system S must enjoy also when it is composed with a possibly untrusted component X. By using the *partial model checking* technique, the property ϕ is projected on another one, say ϕ', depending only on S and ϕ, that only the component X must satisfy. This allows one to monitor only the necessary/untrusted part of the system, here X. Thus we can now force X to enjoy ϕ' by using an appropriate controller $Y \triangleright_{\mathbf{K}} X$. (Note that as a special case we have the opportunity to treat X as a whole system as in other approaches).

This paper is organized as follows. Section 2 presents the necessary background on process algebras and (Generalized) Structured Operational Semantics ($GSOS$), logic and security automata. Section 3 describes some process algebra operators (controllers) corresponding to security automata under investigation. Section 4 shows how to automatically build controller programs that enforce desired security policies. Section 5 shows how to build the maximal model for truncation automata. Section 6 shows a simple example and Section 7 concludes the paper.

2 Background

2.1 Operational semantics and process algebras

We recall a formal method for giving operational semantics to terms of a given language. This approach is called *Generalized Structured Operational Semantics* ($GSOS$) (see [4]). It permits to reason compositionally about the behavior of programs (terms).

2.1.1 $GSOS$ format

Let V be a set of variables, ranged over by $x, y, \ldots$ and let Act be a finite set of actions, ranged over by $a, b, c \ldots$. A *signature* Σ is a pair (F, ar) where:

- F is a set of function symbols, disjoints from V,
- $ar : F \mapsto \mathbb{N}$ is a *rank function* which gives the arity of a function symbol; if $f \in F$ and $ar(f) = 0$ then f is called a *constant symbol*.

Given a signature, let $W \subseteq V$ be a set of variables. It is possible to define the set of Σ-*terms* over W as the least set such that every element in W is a term and if $f \in F$, $ar(f) = n$ and $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term. It is also possible to define an *assignment* as a function γ from the set of variables to the set of terms such that $\gamma(f(t_1, \ldots, t_n)) = f(\gamma(t_1), \ldots \gamma(t_n))$. Given a term t, let $Vars(t)$ be the set of variables in t. A term t is *closed* if $Vars(t) = \emptyset$.

Now we are able to describe the *GSOS* format. A *GSOS* rule r has the following format:

$$\frac{\{x_i \xrightarrow{a_{ij}} y_{ij}\}^{1 \leq i \leq k}_{1 \leq j \leq m_i} \quad \{x_i \overset{b_{ij}}{\not\longrightarrow}\}^{1 \leq i \leq k}_{1 \leq j \leq n_i}}{f(x_1, \ldots, x_k) \xrightarrow{c} g(\boldsymbol{x}, \boldsymbol{y})} \tag{1}$$

where all variables are distinct; $\boldsymbol{x}$ and $\boldsymbol{y}$ are the vectors of all x_i and y_{ij} variables respectively; $m_i, n_i \geq 0$ and k is the arity of f. We say that f is the *operator* of the rule ($op(r) = f$) and c is the action. A *GSOS* system $\mathcal{G}$ is given by a signature and a finite set of *GSOS* rules. Given a signature $\Sigma = (F, ar)$, an assignment ζ is *effective* for a term $f(s_1, \ldots, s_k)$ and a rule r if:

(i) $\zeta(x_i) = s_i$ for $1 \leq i \leq k$;

(ii) for all i, j with $1 \leq i \leq k$ and $1 \leq j \leq m_i$, it holds that $\zeta(x_i) \xrightarrow{a_{ij}} \zeta(y_{ij})$;

(iii) for all i, j with $1 \leq i \leq k$ and $1 \leq j \leq n_i$, it holds that $\zeta(x_i) \overset{b_{ij}}{\not\longrightarrow}$,

The formal semantics of terms is described by a *labelled transition system* (LTS, for short). that is is a pair $(\mathcal{E}, \mathcal{T})$ where $\mathcal{E}$ is the set of terms and $\mathcal{T}$ is a ternary relation $\mathcal{T} \subseteq (\mathcal{E} \times Act \times \mathcal{E})$, known as a *transition relation*. The transition relation among closed terms can be defined in the following way: we have $f(s_1, \ldots, s_n) \xrightarrow{c} s$ iff there exists an *effective* assignment ζ for a rule r with operator f and action c such that $s = \zeta(g(\boldsymbol{x}, \boldsymbol{y}))$. There

$$\text{Prefixing: } \frac{}{a.x \xrightarrow{a} x} \qquad \text{Choice: } \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

$$\text{Parallel: } \frac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y} \qquad \frac{y \xrightarrow{a} y'}{x \| y \xrightarrow{a} x \| y'} \qquad \frac{x \xrightarrow{l} x' \quad y \xrightarrow{\bar{l}} y'}{x \| y \xrightarrow{\tau} x' \| y'}$$

$$\text{Restriction: } \frac{x \xrightarrow{a} x'}{x \backslash L \xrightarrow{a} x' \backslash L} \qquad \text{Relabeling: } \frac{x \xrightarrow{a} x'}{x[f] \xrightarrow{f(a)} x'[f]}$$

Table 1
GSOS system for CCS.

exists a unique transition relation induced by a *GSOS* system (see [4]) and this transition relation is *finitely branching*.

2.1.2 An example: CCS process algebra

CCS of Milner (see [13]) is a language for describing concurrent systems. Here, we present a formulation of Milner's CCS in the *GSOS* format.

The main operator is the *parallel composition* between processes, namely $E \| F$ because, as we explain better later, it permits to model the *parallel composition* of processes. The notion of communication considered is a synchronous one, i.e. both processes must agree on performing the communication at the same time. It is modeled by a simultaneous performing of complementary actions that is represented by a synchronization action (or internal action) τ.

Let $\mathcal{L}$ be a finite set of actions, $\bar{\mathcal{L}} = \{\bar{a} \mid a \in \mathcal{L}\}$ be the set of complementary actions where$\bar{}$is a bijection with $\bar{\bar{a}} = a$, Act be $\mathcal{L} \cup \bar{\mathcal{L}} \cup \{\tau\}$, where τ is a special action that denotes an internal computation step (or communication) and Π be a set of constant symbols that can be used to define processes with recursion. To give a formulation of CCS dealing with *GSOS*, we define the signature $\Sigma_{CCS} = (F_{CCS}, ar)$ as follows.

$$F_{CCS} = \{\mathbf{0}, +, \|\} \cup \{a.|a \in Act\} \cup \{\backslash L | L \subseteq \mathcal{L} \cup \bar{\mathcal{L}}\} \cup \{[f] | f : Act \mapsto Act\} \cup \Pi.$$

The function ar is defined as follows: $ar(\mathbf{0}) = 0$ and for every $\pi \in \Pi$ we have $ar(\pi) = 0$, $\|$ and $+$ are binary operators and the other ones are unary operators.

The operational semantics of CCS closed terms is given by means of the *GSOS* system in table 1 and it is described by an LTS. We denote by $Der(E)$ the set of derivatives of a (closed) term E, i.e. the set of process that can be reached through the transition relation. Informally, a (closed) term $a.E$ represents a process that performs an action a and then behaves as E. The term $E + F$ represents the non-deterministic choice between the processes E and F. Choosing the action of one of the two components, the other is dropped. The term $E \| F$ represents the parallel composition of the two processes E and F. It can perform an action if one of the two processes can perform an action, and this does not prevent the capabilities of the other process. The third rule of parallel composition is characteristic of this calculus, it expresses that the communication between processes happens whenever both can perform complementary actions. The resulting process is given by the parallel composition of the successors of each component, respectively. The process $E \backslash L$ behaves like E but the actions in $L \cup \bar{L}$ are forbidden. To force a synchronization on an action between parallel processes, we have to set restriction operator in conjunction with parallel one. The process $E[f]$ behaves like the E but the actions are renamed $via f$.

2.1.3 Behavioral relation: Simulation

It is often necessary to compare processes that are expressed using different terms in order to understand if there exists some behavioral relation between two processes and which one (see [13]).

We present the notion of *observational relation* as follows.
$E \overset{\tau}{\Rightarrow} E'$ (or $E \Rightarrow E'$) if $E \xrightarrow{\tau}^{*} E'$ (where $\xrightarrow{\tau}^{*}$ is the reflexive and transitive closure of the $\xrightarrow{\tau}$ relation); $E \overset{a}{\Rightarrow} E'$ if $E \overset{\tau}{\Rightarrow}\xrightarrow{a}\overset{\tau}{\Rightarrow} E'$.[4]

Now we are able to give the following definition.

Definition 2.1 Let $(\mathcal{E}, \mathcal{T})$ be an LTS of concurrent processes, and let $\mathcal{R}$ be a binary relation over a set of process $\mathcal{E}$. Then $\mathcal{R}$ is said to be a *simulation* (denoted by $\preceq$) if, whenever $(E, F) \in \mathcal{R}$, if $E \xrightarrow{a} E'$ then $\exists F' \in \mathcal{E}$ s.t. $F \overset{a}{\Rightarrow} F'$ and $(E', F') \in \mathcal{R}$.

2.2 Equational μ-calculus and partial model checking

Equational 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 permits to express a lot of interesting properties like *safety* and *liveness* properties, as well as allowing to express equivalence conditions over LTS. In order to define recursively the properties of a given system, this calculus uses fixpoint equations. Let a be in Act and X be a variable ranging over a finite set of variables V. As we have already said, equational μ-calculus is based on fixpoint equations that substitute recursion operators. $X =_{\mu} A$ is a minimal fixpoint equation, where A is an assertion (i.e. a simple modal formula without recursion operator), and $X =_{\nu} A$ is a maximal fixpoint equation. The syntax of the assertions (A) and of the lists of equations (D) is given by the following grammar:

$$A ::= X \mid \mathbf{T} \mid \mathbf{F} \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \langle a \rangle A \mid [a]A$$

$$D ::= X =_{\nu} AD \mid X =_{\mu} AD \mid \epsilon$$

where the symbol $\mathbf{T}$ means *true* and $\mathbf{F}$ means *false*; $\wedge$ is the symbol of the standard conjunction of formulae, i.e. $A_1 \wedge A_2$ holds iff both of the formulae A_1 and A_2 hold, and $\vee$ is the disjunction of formulae, so $A_1 \vee A_2$ holds when at least one of A_1 and A_2 holds. Moreover $\langle a \rangle A$ is the (*possibility operator*). It means that "exists a transition labeled by a after that A holds". On the other hand, $[a]A$ is the (*necessity operator*) and means "for all transitions labeled by a, A holds". Roughly, the semantics $[\![D]\!]$ 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 set of values of the variable X, and $E \models D \downarrow X$ can be used as a short notation for $E \in [\![D]\!](X)$. The formal semantics is in Table 2. The following standard result of μ-calculus will be useful in the reminder of the paper.

Theorem 2.2 ([18]) *Given a formula ϕ it is possible to decide in exponential time in the length of ϕ if there exists a model of ϕ and it is also possible to give an example of such model.*

Partial model checking (*pmc*) is a technique that was originally developed for compositional analysis of concurrent systems (processes) (see [1]). In order to explain how partial model checking works, we give the intuitive idea underlying it describing *pmc* w.r.t. the parallel operator as follows: proving that $E \| F$ satisfies a formula ϕ $(E \| F \models \phi)$ is equivalent to proving that F satisfies a modified specification $\phi_{//_E}$ $(F \models \phi_{//E})$, where $//_E$ is

[4] Note that it is a short notation for $E \overset{\tau}{\Rightarrow} E_\tau \xrightarrow{a} E'_\tau \overset{\tau}{\Rightarrow} E'$ where E_τ and E'_τ denote intermediate states (not relevant in our framework).

$$[\![\mathbf{T}]\!]'_\rho = S \quad [\![\mathbf{F}]\!]'_\rho = \emptyset \quad [\![X]\!]'_\rho = \rho(X) \quad [\![A_1 \wedge A_2]\!]'_\rho = [\![A_1]\!]'_\rho \cap [\![A_2]\!]'_\rho$$

$$[\![A_1 \vee A_2]\!]'_\rho = [\![A_1]\!]'_\rho \cup [\![A_2]\!]'_\rho \quad [\![\langle a \rangle A]\!]'_\rho = \{s \mid \exists s' : s \xrightarrow{a} s' \text{ and } s' \in [\![A]\!]'_\rho\}$$

$$[\![[a]A]\!]'_\rho = \{s \mid \forall s' : s \xrightarrow{a} s' \text{ implies } s' \in [\![A]\!]'_\rho\}$$

We use $\sqcup$ to represent union of disjoint environments. Let ρ be the environment (a function from variables to values) and σ be in $\{\mu, \nu\}$, then $\sigma U.f(U)$ represents the σ fixpoint of the function f in one variable U.
$[\![\epsilon]\!]_\rho = []$ $\quad [\![X =_\sigma AD']\!]_\rho = [\![D']\!]_{(\rho \sqcup [U'/X])} \sqcup [U'/X]$
where $U' = \sigma U.[\![A]\!]'_{(\rho \sqcup [U/X] \sqcup \rho'(U))}$ and $\rho'(U) = [\![D']\!]_{(\rho \sqcup [U/X])}$.
It informally says that *the solution to $(X =_\sigma A)D$ is the σ fixpoint solution U' of $[\![A]\!]$ where the solution to the rest of the lists of equations D is used as environment.*

Table 2
Equational μ-calculus

$$X//[f] = X \quad \langle a \rangle A//[f] = \bigvee_{b:f(b)=a} \langle b \rangle (A//[f])$$

$$[a]A//[f] = \bigwedge_{b:f(b)=a} [b](A//[f]) \quad A_1 \wedge A_2//[f] = (A_1//[f]) \wedge (A_2//[f])$$

$$A_1 \vee A_2//[f] = (A_1//[f]) \vee (A_2//[f]) \quad \mathbf{T}//[f] = \mathbf{T} \quad \mathbf{F}//[f] = \mathbf{F}$$

Table 3
Partial evaluation function for relabeling operator.

the partial model checking function w.r.t. the parallel composition operator (see [1] for the formal definition). The formula ϕ is specified by the use of the *equational μ-calculus*. A useful result on partial model checking is the following.

Lemma 2.3 ([1]) *Given a process $E \| F$ and a formula ϕ we have: $E \| F \models \phi$ iff $F \models \phi_{//E}$.*

The reduced formula $\phi_{//E}$ depends only on the formula ϕ and on the 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 to successfully attack the system. It is worth noticing that partial model checking function may be automatically derived from the semantics rules used to define a language semantics. Thus, the proposed technique is very flexible.

A lemma similar to Lemma 2.3 holds for a great range of process algebra operators modeled by $GSOS$ (see [1,8]). The partial model checking functions for relabeling operator is given in Table 3.

2.2.1 Characteristic formulae

A *characteristic formula* is an equational μ-calculus formula that completely characterizes the behavior of a (state in an) LTS modulo a chosen notion of behavioral relation. Following the reasoning used in [5,14], we characterize a process w.r.t. simulation as follows.

Definition 2.4 Given a finite state process E, its characteristic formula (w.r.t. simulation) $D_E \downarrow X_E$ is defined by the following equations: for every $E' \in Der(E)$, $X_{E'} =_\nu \bigwedge_{a \in Act}([a](\bigvee_{E'':E' \overset{a}{\Rightarrow} E''} X_{E''}))$.

The following proposition holds.

Lemma 2.5 *Let E be a finite-state process and let $\phi_{E,\preceq}$ be its characteristic formula w.r.t. simulation, then $F \preceq E \Leftrightarrow F \models \phi_{E,\preceq}$.*

2.3 Enforcement mechanisms and Security automata

In this paper we choose to follow the approach given by Ligatti and al. in [3] to describe the behavior of four different kinds of security automata.

A *security automaton* at least consists of a (countable) set of states, say $\mathcal{Q}$, a set of actions Act and a transition (partial) function δ. Each kind of automata has a slightly different sort of transition function δ, and these differences account for the variations in their expressive power. The exact specification of δ is part of the definition of each kind of automaton. We use σ to denote a sequence of actions, $\cdot$ for the empty sequence and τ [5] to represent an internal action.

The execution of each different kind of security automata $\mathbf{K}$ is specified by a labeled operational semantics. The basic single-step judgment has the form $(\sigma, q) \xrightarrow{a}_{\mathbf{K}} (\sigma', q')$ where σ' and q' denote, respectively, the action sequence and the state after that the automaton takes a single step, and a denotes the action produced by the automaton. The single-step judgment can be generalized to a multi-step judgment $(\sigma, q) \overset{\gamma}{\Longrightarrow}_{\mathbf{K}}$ [6] (σ', q'), where γ is a sequence of actions, as follows.

$$\frac{}{(\sigma, q) \overset{\cdot}{\Longrightarrow}_{\mathbf{K}} (\sigma, q)}\ \text{(Reflex)} \qquad \frac{(\sigma, q) \xrightarrow{a}_{\mathbf{K}} (\sigma'', q'') \quad (\sigma'', q'') \overset{\gamma}{\Longrightarrow}_{\mathbf{K}} (\sigma', q')}{(\sigma, q) \overset{a;\gamma}{\Longrightarrow}_{\mathbf{K}} (\sigma', q')}\ \text{(Trans)}$$

The operational semantics for each security automaton is given below.

Truncation automaton. The operational semantics of truncation automata is:
if $\sigma = a;\sigma'$ and $\delta(a, q) = q'$

$$(\sigma, q) \xrightarrow{a}_{T} (\sigma', q') \qquad \text{(T-Step)}$$

otherwise

$$(\sigma, q) \xrightarrow{\tau}_{T} (\cdot, q) \qquad \text{(T-Stop)}$$

Suppression automaton. It is defined as $(\mathcal{Q}, q_0, \delta, \omega)$ where $\omega : Act \times \mathcal{Q} \rightarrow \{-, +\}$ indicates whether or not the action in question should be suppressed (-) or emitted (+).
if $\sigma = a;\sigma'$ and $\delta(a, q) = q'$ and $\omega(a, q) = +$

$$(\sigma, q) \xrightarrow{a}_{S} (\sigma', q') \qquad \text{(S-StepA)}$$

if $\sigma = a;\sigma'$ and $\delta(a, q) = q'$ and $\omega(a, q) = -$

$$(\sigma, q) \xrightarrow{\tau}_{S} (\sigma', q') \qquad \text{(S-StepS)}$$

otherwise

$$(\sigma, q) \xrightarrow{\tau}_{S} (\cdot, q) \qquad \text{(S-Stop)}$$

Insertion automaton. It is defined as $(\mathcal{Q}, q_0, \delta, \gamma)$ where $\gamma : Act \times \mathcal{Q} \rightarrow Act \times \mathcal{Q}$ that specifies the insertion of an action into the sequence of actions of the program. It is necessary to note that in [3,6] the automaton inserts a finite sequence of actions instead

[5] In [3] internal actions are denoted by $\cdot$. According to the standard notation of process algebras, we use τ to denote an internal action.

[6] Consider a finite sequence of visible actions $\gamma = a_1, \ldots, a_n$. Here we use $\Rightarrow$ to denote automata computations. Before we use the same notation for process algebra computations. The meaning of the symbol will be clear from the context.

of only one action, i.e., using the function γ, it controls if a wrong action is performed. If it holds, the automaton inserts a finite sequence of actions, hence a finite number of intermediate states. Without loss of generality, we consider that it performs only one action. In this way we openly consider all intermediate states. Note that the domain of γ is disjoint from the domain of δ in order to have a deterministic automata.
if $\sigma = a; \sigma'$ and $\delta(a, q) = q'$

$$(\sigma, q) \xrightarrow{a}_I (\sigma', q') \tag{I-Step}$$

if $\sigma = a; \sigma'$ and $\gamma(a, q) = (b, q')$

$$(\sigma, q) \xrightarrow{b}_I (\sigma, q') \tag{I-Ins}$$

otherwise

$$(\sigma, q) \xrightarrow{\tau}_I (\cdot, q) \tag{I-Stop}$$

Edit automaton. It is defined as $(\mathcal{Q}, q_0, \delta, \gamma, \omega)$ where $\gamma : Act \times \mathcal{Q} \rightarrow Act \times \mathcal{Q}$ that specifies the insertion of a finite sequence of actions into the program's actions sequence and $\omega : Act \times \mathcal{Q} \rightarrow \{-, +\}$ indicates whether or not the action in question should be suppressed (-) or emitted (+). Also here ω and δ have the same domain while the domain of γ is disjoint from the domain of δ in order to have a deterministic automata.
if $\sigma = a; \sigma'$ and $\delta(a, q) = q'$ and $\omega(a, q) = +$

$$(\sigma, q) \xrightarrow{a}_E (\sigma', q') \tag{E-StepA}$$

if $\sigma = a; \sigma'$ and $\delta(a, q) = q'$ and $\omega(a, q) = -$

$$(\sigma, q) \xrightarrow{\tau}_E (\sigma', q') \tag{E-StepS}$$

if $\sigma = a; \sigma'$ and $\gamma(a, q) = (b, q')$

$$(\sigma, q) \xrightarrow{b}_E (\sigma, q') \tag{E-Ins}$$

otherwise

$$(\sigma, q) \xrightarrow{\tau}_E (\cdot, q) \tag{E-Stop}$$

3 Modeling security automata with process algebra

In this section we give the semantics of some process algebra operators, denoted by $Y \triangleright_{\mathbf{K}} X$ where $\mathbf{K} \in \{T, S, I, E\}$[7], that act as *controller operators*. These permit to control the behavior of the (possibly untrusted) component X, given the behavior of the control program Y.

3.1 Our controller operators in process algebra

Here we define our controller operators by showing their behavior trought semantics rules. We denote with E the program controller and with F the target. We work, without loss of generality, under the additional assumption that E and F never perform the internal action τ.

[7] We choose these symbols to denote four operators that have the same behavior of truncation, suppression, insertion and edit automata, respectively.

3.1.1 Truncation automata: $\rhd_T$

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

This operator models the truncation automaton that is similar to Schneider's automaton (when considering only deterministic automata, e.g., see [3,6]). Its semantics rule states that if F performs the action a and the same action is performed by E (so it is allowed in the current state of the automaton), then $E \rhd_T F$ performs the action a, otherwise it halts.

Proposition 3.1 *Let* $E^q = \sum_{a \in Act \setminus \{\tau\}} \begin{cases} a.E^{q'} \; \textit{iff} \; \delta(a,q) = q' \\ \mathbf{0} \; \textit{othw} \end{cases}$

be the control process and let F be the target. Each sequence of actions that is an output of a truncation automaton $(\mathcal{Q}, q_0, \delta)$ *is also derivable from $E^q \rhd_T F$ and vice-versa.*

3.1.2 Suppression automata: $\rhd_S$

$$\frac{E \xrightarrow{a} E' \; F \xrightarrow{a} F'}{E \rhd_S F \xrightarrow{a} E' \rhd_S F'} \qquad \frac{E \xrightarrow{-a} E' \quad F \xrightarrow{a} F'}{E \rhd_S F \xrightarrow{\tau} E' \rhd_S F'}$$

where $-a$ is a control action not in Act (so it does not admit a complementary action). As for the truncation automaton, if F performs the same action performed by E also $E \rhd_S F$ performs it. On the contrary, if F performs an action a that E does not perform and E can perform the control action $-a$ then $E \rhd_S F$ performs the action τ that *suppresses* the action a, i.e., a becomes not visible from external observation. Otherwise, $E \rhd_S F$ halts.

Proposition 3.2 *Let*$E^{q,\omega} = \sum_{a \in Act \setminus \{\tau\}} \begin{cases} a.E^{q',\omega} \; \textit{iff} \; \omega(a,q) = + \; \textit{and} \; \delta(a,q) = q' \\ -a.E^{q',\omega} \; \textit{iff} \; \omega(a,q) = - \; \textit{and} \; \delta(a,q) = q' \\ \mathbf{0} \; \textit{othw} \end{cases}$

be the control process and let F be the target. Each sequence of actions that is an output of a suppression automaton $(\mathcal{Q}, q_0, \delta, \omega)$ *is also derivable from $E^{q,\omega} \rhd_S F$ and vice-versa.*

3.1.3 Insertion automata: $\rhd_I$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E \rhd_I F \xrightarrow{a} E' \rhd_I F'} \qquad \frac{E \not\xrightarrow{a} E' \quad E \xrightarrow{+a.b} E' \quad F \xrightarrow{a} F'}{E \rhd_I F \xrightarrow{b} E' \rhd_I F}$$ [8]

where $+a$ is an action not in Act. If F performs an action a that also E can perform, the whole system makes this action. If F performs an action a that E does not perform and E detects it by performing a control action $+a$ followed by an actio b, then the whole system perform b. It is possible to note that in the description of insertion automata in [3] the domains of γ and δ are disjoint. In our case, this is guarantee by the premise of the second rule in which we have that $E \not\xrightarrow{a} E'$, $E \xrightarrow{+a.b} E'$. In fact for the insertion automata, if a pair (a,q) is not in the domain of δ and it is in the domain of γ it means that the action a and the state q are not compatible so in order to change state an action different from a must be performed. It is important to note that it is able to insert new actions but it is not able to suppress any action performed by F.

[8] This means $E \xrightarrow{+a} E_a \xrightarrow{b} E'$. However we consider $+a.b$ as a single action, i.e. the state E_a is hide and we do not consider it in $Der(E)$.

Proposition 3.3 *Let* $E^{q,\gamma} = \sum_{a \in Act \setminus \{\tau\}} \begin{cases} a.E^{q',\gamma} \textit{ iff } \delta(a,q) \\ +a.b.E^{q',\gamma} \textit{ iff } \gamma(a,q) = (b,q') \\ \mathbf{0} \textit{ othw} \end{cases}$
be the control process and let F be the target. Each sequence of actions that is an output of an insertion automaton $(\mathcal{Q}, q_0, \delta, \gamma)$ *is also derivable from* $E^{q,\gamma} \triangleright_I F$ *and vice-versa.*

3.1.4 Edit automata: $\triangleright_E$

In order to do insertion and suppression together we define the following controller operator. Its rules are the union of the rules of the $\triangleright_S$ and $\triangleright_I$.

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E \triangleright_E F \xrightarrow{a} E' \triangleright_E F'} \qquad \frac{E \xrightarrow{-a} E' \quad F \xrightarrow{a} F'}{E \triangleright_E F \xrightarrow{\tau} E' \triangleright_E F'} \qquad \frac{E \overset{a}{\not\to} E' \quad E \xrightarrow{+a.b} E' \quad F \xrightarrow{a} F'}{E \triangleright_E F \xrightarrow{b} E' \triangleright_E F}$$

This operator combines the power of the previous two ones.

Proposition 3.4 *Let* $E^{q,\gamma,\omega} = \sum_{a \in Act \setminus \{\tau\}} \begin{cases} a.E^{q',\gamma,\omega} \textit{ iff } \delta(a,q) = q' \textit{ and } \omega(a,q) = + \\ -a.E^{q',\gamma,\omega} \textit{ iff } \delta(a,q) = q' \textit{ and } \omega(a,q) = - \\ +a.b.E^{q',\gamma,\omega} \textit{ iff } \gamma(a,q) = (b,q') \\ \mathbf{0} \textit{ othw} \end{cases}$
be the control process and let F be the target. Each sequence of actions that is an output of an edit automaton $(\mathcal{Q}, q_0, \delta, \gamma, \omega)$ *is also derivable from* $E^{q,\gamma,\omega} \triangleright_E F$ *and vice-versa.*

It is important to note that we introduced the control action $-a$ in the semantics of $\triangleright_S$ and $+a$ in the semantics of $\triangleright_I$ in order to find operators that were as similar as possible to suppression and insertion automata, respectively. Other definitions could be possible, although some attempts we made failed on defining a tractable semantics.

4 Synthesis of controller programs

Exploiting our framework we can build a program controller Y which allows to enforce a desired security property for any target system X. We present an extension of [10]. Here we have four different operators and in particular we have to deal with control actions.

Let S be a system, and let X be one component that may be dynamically changed (e.g., a downloaded mobile agent) that we consider a possibly untrusted one. We would like that for any actual behavior of X, the system $S \| X$ enjoys a security property expressed by a logical formula ϕ, i.e., $\forall X \ (S \| X) \models \phi$.

In order to protect the system we might simply check the correctness of each process X before it is executed or, if this is not possible (or not desirable), we may define a controller that, in any case, forces each process to behave correctly. Here, we study here how to build a program controller in order to force the unknown component to behave correctly. Thus, we want to find a control program Y such that:

$$\forall X \quad (S \| Y \triangleright_{\mathbf{K}} X) \models \phi \tag{2}$$

By using the partial model checking approach proposed in [9], we can focus on the properties of $Y \triangleright_{\mathbf{K}} X$, i.e.:

$$\exists Y \ \forall X \ (Y \triangleright_{\mathbf{K}} X) \models \phi' \tag{3}$$

where $\phi' = \phi_{//S}$.

In order to manage the universal quantification in (3), we prove the following proposition.

Proposition 4.1 *For every* $\mathbf{K} \in \{T, S, I, E\}$ $Y \triangleright_{\mathbf{K}} X \preceq Y[f_{\mathbf{K}}]$ *holds, where* $f_{\mathbf{K}}$ *is a relabeling function depending on* $\mathbf{K}$*. In particular,* f_T *is the identity function on* Act[9] *and*

$$f_S(a) = \begin{cases} \tau \ \textit{if}\ a = -a \\ a \ \ \textit{othw} \end{cases} \qquad f_I(a) = \begin{cases} \tau \ \textit{if}\ a = +a \\ a \ \ \textit{othw} \end{cases} \qquad f_E(a) = \begin{cases} \tau \ \textit{if}\ a \in \{+a, -a\} \\ a \ \ \textit{othw} \end{cases}$$

Now we restrict ourselves to a subclass of equational μ-calculus formulae that is denoted by Fr_μ. This class consists of equational μ-calculus formulae without $\langle \_ \rangle$. It is easy to prove that this set of formulae is closed under the partial model checking function and the following result holds.

Proposition 4.2 *Let* E *and* F *be two finite state processes and* $\phi \in Fr_\mu$*. If* $F \preceq E$ *then* $E \models \phi \Rightarrow F \models \phi$*.*

At this point in order to satisfy the formula (3) it is sufficient to have:

$$\exists Y \quad Y[f_{\mathbf{K}}] \models \phi'$$

To further reduce the previous formula, we can use the partial model checking function for relabeling operator. Hence, for every $\mathbf{K} \in \{T, S, I, E\}$ we calculate $\phi''_{\mathbf{K}} = \phi'_{//[f_{\mathbf{K}}]}$. Thus we obtain:

$$\exists Y \quad Y \models \phi''_{\mathbf{K}} \text{[10]} \tag{4}$$

In this way, we obtain a satisfiability problem in μ-calculus that can be solved by Theorem 2.2.

5 Synthesis of Maximal Model

In the previous section we have shown a method to synthesize a program controller for each of controller operators defined in section 3.1. As matter of fact, we find a deterministic process that does not perform τ actions and that is a model for a given μ-calculus formula.

In this section we define the notion of *maximal model* w.r.t. the simulation relation and show how it is possible to synthesize a *maximal program controller* Y for the operator $Y \triangleright_T X$.

We define the notion of maximal model w.r.t. the relation of simulation as follows. A process E is a *maximal model for a given formula* ϕ iff $E \models \phi$ and $\forall E'$ s.t. $E' \models \phi$, $E' \preceq E$ (see [15,16]). Informally, the maximal program controller Y is the process that restricts as little as possible the activity of the target X.

In order to find the maximal model we exploit the theory developed by Walukiewicz in [19].

Usually the discovered model is a non-deterministic process. In order to find a deterministic model we consider a subset of formulae of Fr_μ without $\vee$. This set of formulae is

[9] Here the set Act must be consider enriched by control actions.

[10] Even if the process Y performs some actions τ it is possible to obtain from Y another process Y' with only visible actions that is a deterministic model of ϕ.

called the *universal conjunctive μ-calculus formulae*, $\forall_\wedge\mu C$. It is easy to prove that $\forall_\wedge\mu C$ is closed under the partial model checking function (see [5]).

Proposition 5.1 *Given a formula $\phi \in \forall_\wedge\mu C$, a maximal deterministic model E of this formula exists.*

In order to generate the maximal model E, we find a model for $\phi \wedge \psi$ where $\psi = X$, $X =_\nu \bigwedge_{\alpha \in Act\setminus\{\tau\}}([\alpha]\mathbf{F} \vee (\langle\alpha\rangle X \wedge [\alpha]X))$. The formula ψ permits us to check all the actions in Act. Exploiting the theory of Walukievicz, we find a deterministic model E for $\phi \wedge \psi$ that does not perform τ actions. It is obviously a model of ϕ. The following lemma holds.

Lemma 5.2 *Let $E' \models \phi$ with $\phi \in \forall_\wedge\mu C$. Then the model of $\phi \wedge \psi$ E, is such that $E' \preceq E$.*

Hence E is the maximal model for ϕ.

6 A simple example

Consider the process $S = a.b.\mathbf{0}$ and consider the following equational definition $\phi = Z$ where $Z =_\nu [\tau]Z \wedge [a]W$ and $W =_\nu [\tau]W \wedge [c]\mathbf{F}$. It asserts that after every action a, an action c cannot be performed. Let $Act = \{a, b, c, \tau, \bar{a}, \bar{b}, \bar{c}\}$ be the set of actions. Applying the partial evaluation for the parallel operator we obtain, after some simplifications, the following system of equation, that we denoted with $\mathcal{D}$.

$$
\begin{array}{ll}
Z_{//S} =_\nu [\tau]Z_{//S} \wedge [\bar{a}]Z_{//S'} \wedge [a]W_{//S} \wedge W_{//S'} & Z_{//\mathbf{0}} = \mathbf{T} \\
W_{//S'} =_\nu [\tau]W_{//S'} \wedge [\bar{b}]W_{//\mathbf{0}} \wedge [c]\mathbf{F} & W_{//\mathbf{0}} = \mathbf{T} \\
Z_{//S'} =_\nu [\tau]Z_{//S'} \wedge [\bar{b}]Z_{//\mathbf{0}} \wedge [a]W_{//S'} & \\
W_{//S} =_\nu [\tau]W_{//S} \wedge [\bar{a}]W_{//S'} \wedge [c]\mathbf{F} &
\end{array}
$$

where $S \xrightarrow{a} S'$ so S' is $b.\mathbf{0}$.

The information obtained through partial model checking can be used to enforce a security policy. In particular, choosing one of the four operators and using its definition we simply need to find a process $Y[f_{\mathbf{K}}]$, where $\mathbf{K}$ depend on the chosen controller, that is a model for the previous formula.

In this simple example we choose the controller operator $\triangleright_S$. Hence we apply the partial model checking for relabeling function f_S to the previous formula, that we have simplified replacing $W_{//\mathbf{0}}$ and $Z_{//\mathbf{0}}$ by $\mathbf{T}$ (and assumed that Y can only suppress c actions). We obtain $\mathcal{D}_{//f_S}$ as follows.

$$
\begin{array}{l}
Z_{//S,f_S} =_\nu [\tau]Z_{//S,f_S} \wedge [-c]Z_{//S,f_S} \wedge [\bar{a}]Z_{//S',f_S} \wedge [a]W_{//S,f_S} \wedge W_{//S',f_S} \\
W_{//S',f_S} =_\nu [\tau]W_{//S',f_S} \wedge [-c]W_{//S',f_S} \wedge [\bar{b}]\mathbf{T} \wedge [c]\mathbf{F} \\
Z_{//S',f_S} =_\nu [\tau]Z_{//S',f_S} \wedge [-c]Z_{//S',f_S} \wedge [\bar{b}]\mathbf{T} \wedge [a]W_{//S',f_S} \\
W_{//S,f_S} =_\nu [\tau]W_{//S,f_S} \wedge [-c]W_{//S,f_S} \wedge [\bar{a}]W_{//S',f_S} \wedge [c]\mathbf{F}
\end{array}
$$

We can note the process $Y = a. - c.\mathbf{0}$ is a model of $\mathcal{D}_{//f_S}$. Then, for any component X, we have $S\|(Y \triangleright_S X)$ satisfies ϕ. For instance, consider $X = a.c.\mathbf{0}$. Looking at the first rule of $\triangleright_S$, we have:
$(S\|(Y \triangleright_S X)) = (a.b.\mathbf{0}\|(a. - c.\mathbf{0} \triangleright_S a.c.\mathbf{0})) \xrightarrow{a} (a.b.\mathbf{0}\|(-c.\mathbf{0} \triangleright_S c.\mathbf{0}))$
Using the second rule we eventually get:
$(a.b.\mathbf{0}\|(-c.\mathbf{0} \triangleright_S c.\mathbf{0})) \xrightarrow{\tau} (a.b.\mathbf{0}\|\mathbf{0} \triangleright_S \mathbf{0})$

and so the system still preserves its security since the actions performed by the component X have been prevented from being visible outside.

7 Conclusion and Future work

We illustrated some results towards a uniform theory for enforcing security properties. With this work, we extended a framework based on process calculi and logical techniques, that have been shown to be very suitable to model and verify several security properties, to tackle also synthesis problems of secure systems.

As future work we plan to implement the theory here showed in order to generate the program controllers and to extend it in other application scenarios as the time-based ones.

Acknowledgement We thank the anonymous referees of STM06 for valuable comments that helped us to improve this paper.

References

[1] Andersen, H. R., *Partial model checking*, in: *LICS '95: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science* (1995), p. 398.

[2] Bartoletti, M., P. Degano and G. Ferrari, *Policy framings for access control*, in: *Proceedings of the 2005 workshop on Issues in the theory of security*, Long Beach, California, 2005, pp. 5 – 11.

[3] Bauer, L., 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* (2002), pp. 95–104.

[4] Bloom, B., S. Istrail and A. R. Meyer, *Bisimulation can't be traced*, J.ACM **42** (1995).

[5] Gnesi, S., 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).

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

[7] Ligatti, J., L. Bauer and D. Walker, *Enforcing non-safety security policies with program monitors*, in: *10th European Symposium on Research in Computer Security (ESORICS)*, 2005.

[8] Martinelli, F., "Formal Methods for the Analysis of Open Systems with Applications to Security Properties," Ph.D. thesis, University of Siena (1998).

[9] Martinelli, F., *Partial model checking and theorem proving for ensuring security properties*, in: *CSFW '98: Proceedings of the 11th IEEE Computer Security Foundations Workshop* (1998).

[10] Martinelli, F. and I. Matteucci, *Partial model checking, process algebra operators and satisfiability procedures for (automatically) enforcing security properties*, Presented at the International Workshop on Foundations of Computer Security (FCS05) (2005).

[11] Martinelli, F. and I. Matteucci, *Modeling security automata with process algebras and related results* (2006), presented at the 6th International Workshop on Issues in the Theory of Security (WITS '06) - Informal proceedings.

[12] Milner, R., *Synthesis of communicating behaviour*, in: *Proceedings of 7th MFCS*, Poland, 1978.

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

[14] Müller-Olm, M., *Derivation of characteristic formulae*, in: *MFCS'98 Workshop on Concurrency*, Electronic Notes in Theoretical Computer Science (ENTCS) **18** (1998).

[15] Riedweg, S. and S. Pinchinat, *Maximally permissive controllers in all contexts*, in: *Workshop on Discrete Event Systems*, Reims, France, 2004.

[16] Riedweg, S. and S. Pinchinat, *You can always compute maximally permissive controllers under partial observation when they exist.*, in: *Proc. 2005 American Control Conference.*, Portland, Oregon, 2005.

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

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

[19] Walukiewicz, I., "A Complete Deductive System for the μ-Calculus," Ph.D. thesis, Institute of Informatics, Warsaw University (1993).