Modeling Security Automata with process algebras and related results ⋆

Fabio Martinelli[1], Ilaria Matteucci[1,2]
{Fabio.Martinelli, Ilaria.Matteucci}@iit.cnr.it

Istituto di Informatica e Telematica - C.N.R., Pisa, Italy[1]
Dipartimento di Matematica, Università degli Studi di Siena[2]

Abstract. We define a set of process algebra operators (controllers) that mimic the security automata introduced by Schneider in [1] and by Ligatti and al. in [2], respectively. We show how to automatically build these controllers for specific security properties and then we extend these results to a discrete-time setting.

1 Overview

Recently, several papers tackled the formal definition of mechanisms for enforcing security policies (e.g., see [1,2,3,4,5]).

The focus of this paper is the study of the enforcement mechanisms introduced by Schneider in [1] and security automata developed by Ligatti and al. in [2,4].

In [1], Schneider deals with enforcement security properties in a systematic way. He discusses whether a given security property is enforceable and at what cost. To study those issues, 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. A security automaton defined in [1] is a triple $(\mathcal{Q}, q_0, \delta)$ where $\mathcal{Q}$ is a set of states, q_0 is the initial one and $\delta : Act \times \mathcal{Q} \rightarrow \mathcal{Q}$, where Act is a set of actions, is the transition function. A security automata processes a finite or infinite sequence $s_1 s_2 \ldots$ of input actions. At each step only one action is considered and for each action we calculate the *global state* Q' that is the the set of the possible states for the current action, i.e. if the automaton is checking the action s_i then $Q' = \bigcup_{q \in Q'} \delta(q, s_i)$. If the automaton can make a transition on given input symbol, i.e. Q' is not empty, 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. The security properties that can be enforced in this way corresponds to safety properties[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).

[1] There are several definition of safety: informally a property is a safety one, if whenever it does not hold in trace then it does not hold in a finite prefix of it.

It is possible to note that such automata are non-deterministic, although it is possible restricting ourselves to deterministic ones (e.g., when considering finite state automata).

Starting from the work of Schneider described above, Ligatti and al. in [2,4] have defined four different deterministic security automata which deal with finite sequence of actions. They define **truncation automata** (similar to Schneider's ones) which can recognize bad sequences of actions and halts program execution before 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's because both of them read one input action at a time. The **suppression automaton** has the ability to suppress individual program actions without terminating the program outright in addition to being able to halt program execution. The third automata 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 action sequences and insert or suppress security-relevant actions at will.

These works have been extended [6] studying how truncation automata and edit automata work on possible infinite sequence of input actions. In this way they analyze how certain non-safety properties may be enforced. This work comes back to the original Schneider's idea to deal with also possibly infinite sequences of actions.

In this paper we introduce process algebra operators able to mimic the behavior of the security automata briefly described above. The process algebras operators $Y \rhd_{\mathcal{K}} X$ (where $\mathcal{K}$ is the name of the corresponding automata) act as programmable (Y) controllers of a target system (X).

We can then exploit a huge theory for security analysis based on these formalisms. In particular, we show how to automatically build programs that allow to enforce security properties for whatever target system depending on the security automata one chooses.

As a matter of fact, most of the related works deal with the verification rather than with the synthesis problem. The synthesis is based on a satisfiability procedure for the μ-calculus that allows to obtain a model for a logical formula (in our framework a suitable property). We then exploit the previous work on timed process algebras for security by defining controllers that work in a timed setting as well as the corresponding synthesis techniques.

This paper is organized as follows. Section 2 presents the necessary background on process algebras, logic and security automata. Section 3 shows some process algebra operators (controllers) corresponding to the security automata under investigation. Section 4 shows how to automatically build programs that enforce desired security policies, in a given context and for each possible target system. Section 5 extends the results to a timed setting and section 6 shows a simple example.

2 Background

2.1 Operational semantics and process algebra

We recall a formal method for giving operational semantics to terms of a given languages. This approach is called Generalized Structured Operational Semantics (GSOS)(see

[7]). It permits to reason compositionally about the behavior of program term, e.g. the passage of time, timeouts, and so on.

***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 define the set of *Σ-terms* over W as the least set s.t. 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 s.t. $\gamma(f(t_1, \ldots, t_n)) = f(\gamma(t_1), \ldots \gamma(t_n))$.

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:

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

where all variables are distinct; $\boldsymbol{x}$ and $\boldsymbol{y}$ are the vectors of all x_i and y_{ij} variables respectively; $m_i, n_i \ge 0$ 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 γ is *effective* for a term $f(s_1, \ldots, s_k)$ and a rule r if:

1. $\gamma(x_i) = s_i$ for $1 \le i \le k$;
2. for all i, j with $1 \le i \le k$ and $1 \le j \le m_i$, it holds that $\gamma(x_i) \xrightarrow{a_{ij}} \gamma(y_{ij})$;
3. for all i, j with $1 \le i \le k$ and $1 \le j \le n_i$, it holds that $\gamma(x_i) \overset{b_{ij}}{\not\longrightarrow}$,

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 γ for a rule r with operator f and action c s.t. $s = \gamma(g(\boldsymbol{x}, \boldsymbol{y}))$. There exists a unique transition relation induced by a *GSOS* system (see [7]) and this transition relation is *finitely branching*.

An example: CCS process algebra CCS of Milner (see [8]) 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 the 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

Prefixing: $\dfrac{}{a.x \xrightarrow{a} x}$

Choice: $\dfrac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \quad \dfrac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$

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

Restriction: $\dfrac{x \xrightarrow{a} x'}{x \backslash L \xrightarrow{a} x' \backslash L}$

Relabeling: $\dfrac{x \xrightarrow{a} x'}{x[f] \xrightarrow{a'} x'[f]}$

Table 1. *GSOS* system for CCS.

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.

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

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. 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 $via f$.

2.2 Strong and weak bisimulations

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.

A LTS over Act is a pair $(\mathcal{E}, \mathcal{T})$ where $\mathcal{T}$ is a ternary relation $\mathcal{T} \subseteq (\mathcal{E} \times Act \times \mathcal{E})$, known as a *transition relation*. We recall some useful relations between processes (see [8]).

Definition 1. *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:*

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

Moreover, 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.*

Referring to [7], let $\mathcal{G}$ be a $GSOS$ system, the strong bisimulation is a congruence w.r.t. the operations in $\mathcal{G}$, i.e., the strong bisimulation is preserved by all *GSOS* definable operators.

Another kind of equivalence 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 relation* is the following: $E \stackrel{\tau}{\Rightarrow} E'$ (or $E \Rightarrow E'$) if $E \stackrel{\tau}{\rightarrow}^* E'$ (where $\stackrel{\tau}{\rightarrow}^*$ is the reflexive and transitive closure of the $\stackrel{\tau}{\rightarrow}$ relation); for $a \neq \tau$, $E \stackrel{a}{\Rightarrow} E'$ if $E \stackrel{\tau}{\Rightarrow}\stackrel{a}{\rightarrow}\stackrel{\tau}{\Rightarrow} E'$.

Let $Der(E)$ 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 following definition.

Definition 2. *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* weak simulation *(denoted by $\preceq$) if, whenever $(E, F) \in \mathcal{R}$,*

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

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

It is important to note that every strong simulation is also a weak one (see [8]).

2.3 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 systems, this calculus uses fixpoint equations. Let a be in Act and X be a variable ranging over a finite set of variables V. 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 symbol **T** means *true* and **F** means *false*; $\wedge$ is the symbol for the conjunction of formulae, i.e. the conjunction $X_1 \wedge X_2$ holds iff both of the formulaeX_1 and X_2 hold, and $\vee$ is the disjunction of formulae and $X_1 \vee X_2$ holds when at least one of X_1 and X_2 holds. Moreover the meaning of $\langle a \rangle X$ (*possibility operator*) is 'it is possible to do an a-action to a state where X holds' and the meaning of $[a]X$ (*necessity operator*) is 'for all a-actions performed X holds'. $X =_\mu A$ 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. 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 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 semantic is in Table 2.

The following standard result of μ-calculus will be useful in the reminder of the paper.

$[\![\mathbf{T}]\!]'_\rho = S \quad [\![\mathbf{F}]\!]'_\rho = \emptyset \quad [\![X]\!]'_\rho = \rho(X) \quad [\![A_1 \wedge A_2]\!]'_\rho = [\![A_1]\!]'_\rho \cap [\![A_2]\!]'_\rho$
$[\![A_1 \vee A_2]\!]'_\rho = [\![A_1]\!]'_\rho \cup [\![A_2]\!]'_\rho \quad [\![\langle a \rangle A]\!]'_\rho = \{s \mid \exists s' : s \xrightarrow{a} s' \text{ and } s' \in [\![A]\!]'_\rho\}$
$[\![[a]A]\!]'_\rho = \{s \mid \forall s' : s \xrightarrow{a} s' \text{ implies } s' \in [\![A]\!]'_\rho\}$

We use $\sqcup$ to represent union of disjoint environments. Let σ be in $\{\mu, \nu\}$, then $\sigma U.f(U)$ represents the σ fixpoint of the function f in one variable U.
$[\![\epsilon]\!]_\rho = [] \qquad [\![X =_\sigma AD']\!]_\rho = [\![D']\!]_{(\rho \sqcup [U'/X])} \sqcup [U'/X]$
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

Theorem 1 ([9]). *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 [10]). The intuitive idea underlying the *pmc* is the following: 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 the partial evaluation function for the parallel composition operator. The formula ϕ is specified by use the *equational μ-calculus*.

A useful result of partial model checking is the follow.

Lemma 1 ([10]). *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 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. Thus, the proposed technique is very flexible.

A lemma similar to lemma 1 holds for every operator of *CCS* (see [10]) and also for every new operator defined by *GSOS* rules. The partial model checking function for parallel operator, relabeling and restriction is give in tables 3 and 4.

2.4 Enforcement mechanisms and Security automata

In this paper we chose to follow the semantic approach given by Ligatti and al. in [2] to describe the behavior of four different kind of security automata. First of all we recall the definition of security automata. A *security automata* is a deterministic finite-state or countably infinite-state machine that is defined with respect to some system with action set Act. A program monitor can be formally modeled by a security automata. The authors restrict themselves to finite but arbitrarily long executions. Now we report all the definition and the semantics of automata that are given in [2]. We start giving

$$
\begin{array}{rl}
(D\downarrow X)//t = & (D//t)\downarrow X_t \\
\epsilon//t = & \epsilon \\
(X =_\sigma AD)//t = & ((X_s =_\sigma A//s)_{s\in Der(E)})(D)//t \\
X//t = & X_t \\
\langle a\rangle A//s = & \langle a\rangle(A//s) \vee \bigvee_{s\xrightarrow{a}s'} A//s',\ \text{if } a \neq \tau \\
\langle \tau\rangle A//s = & \langle \tau\rangle(A//s) \vee \bigvee_{s\xrightarrow{\tau}s'} A//s' \vee \bigvee_{s\xrightarrow{a}s'} \langle\overline{a}\rangle(A//s') \\
[a]A//s = & [a](A/\!/s) \wedge \bigwedge_{s\xrightarrow{a}s'} A//s',\ \text{if } a \neq \tau \\
[\tau]A//s = & [\tau](A/\!/s) \wedge \bigwedge_{s\xrightarrow{\tau}s'} A//s' \wedge \bigwedge_{s\xrightarrow{a}s'} [\overline{a}](A\ //s') \\
A_1 \wedge A_2//s = & (A_1//s) \wedge (A_2//s) \\
A_1 \vee A_2//s = & (A_1//s) \vee (A_2//s) \\
\mathbf{T}//s = & \mathbf{T} \\
\mathbf{F}//s = & \mathbf{F}
\end{array}
$$

Table 3. Partial evaluation function for parallel operator $E\|_-$.

Restriction:

$$
\begin{array}{ll}
X//\backslash L = X & \\
\langle a\rangle A//\backslash L = & \begin{array}{ll}\langle a\rangle(A//\backslash L) & \text{if } a \notin L \cup \bar{L} \\ \mathbf{F} & \text{if } a \in L\end{array} \\
[a]A//\backslash L = & \begin{array}{ll}[a](A//\backslash L) & \text{if } a \notin L \cup \bar{L} \\ \mathbf{T} & \text{if } a \in L\end{array} \\
A_1 \wedge A_2//\backslash L = (A_1//\backslash L) \wedge (A_2//\backslash L) & \\
A_1 \vee A_2//\backslash L = (A_1//\backslash L) \vee (A_2//\backslash L) & \\
\mathbf{T}//\backslash L = \mathbf{T} & \\
\mathbf{F}//\backslash L = \mathbf{F} &
\end{array}
$$

Relabeling:

$$
\begin{array}{l}
X//[f] = X \\
\langle a\rangle A//[f] = \bigvee_{b:f(a)=b} \langle b\rangle(A//[f]) \\
[a]A//[f] = \bigwedge_{b:f(a)=b} \langle b\rangle(A//[f]) \\
A_1 \wedge A_2//[f] = (A_1//[f]) \wedge (A_2//[f]) \\
A_1 \vee A_2//[f] = (A_1//[f]) \vee (A_2//[f]) \\
\mathbf{T}//[f] = \mathbf{T} \\
\mathbf{F}//[f] = \mathbf{F}
\end{array}
$$

Table 4. Partial evaluation function for restriction and relabeling operator.

some notation: with σ we denote a sequences of actions, $\cdot$ is the empty sequence, τ[2] models an internal action. It is important to note that it is used to indicate the action of stop but it is not really performed. Then there is the transition function δ that is a partial function $\delta : Act \times \mathcal{Q} \rightarrow \mathcal{Q}$, it indicates that the automata should accept the current input and move in a new state. The execution of each different kind of security automaton is specified by a labeled operational semantics where the basic single-step judgment has the form $(\sigma, q) \xrightarrow{\alpha} (\sigma', q')$ where σ' and q' denote the actions sequence and state after the automaton takes a single step, and α denotes the sequence of actions produced by the automaton. The single-step judgment can be generalized to a multi-step judgment($(\sigma, q) \stackrel{\alpha}{\Longrightarrow} (\sigma', q')$) as follows.

$$
\frac{}{(\sigma, q) \stackrel{\cdot}{\Longrightarrow} (\sigma', q')}\ \text{(Reflex)} \qquad \frac{(\sigma, q) \xrightarrow{\alpha_1} (\sigma'', q'') \quad (\sigma'', q') \stackrel{\alpha_2}{\Longrightarrow} (\sigma', q')}{(\sigma, q) \stackrel{\alpha_1;\alpha_2}{\Longrightarrow} (\sigma'', q'')}\ \text{(Trans)}
$$

The operational semantic for each security automaton is the following.

truncation automaton can recognize bad sequences of actions and halts program execution before a security property is violated, but cannot modify program behavior. These automata are similar to Schneider's original security monitors (see [2,4]).

[2] In [2] internal actions are denoted by $\cdot$. We use τ because we use CCS where internal actions are commonly denoted by τ.

The operational semantic of truncation automata is:

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

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

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

otherwise;

suppression automaton in addition to being able to halt program execution, has the ability to suppress individual program actions without terminating the program outright. It is define 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 (+).

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

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

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

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

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

otherwise

insertion automaton is able to insert a sequence of actions into the program actions stream as well as terminate the program. It is define 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 [2,4] the automaton inserts a finite sequence of actions instead of only one action, i.e., it controls if a wrong action is performed by function γ. If it holds, the automaton inserts a finite sequence of actions, hence there exists 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 state. Note that the domain of γ is disjointed from the domain of δ in order to have a deterministic automata;

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

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

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

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

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

otherwise

edit automaton combines the power of suppression and insertion automata. It is able to truncate action sequences and to insert or to suppress security-relevant actions at will. It is define as $(\mathcal{Q}, q_0, \delta, \gamma, \omega)$ where $\gamma : \mathcal{A} \times \mathcal{Q} \to \overrightarrow{\mathcal{A}} \times \mathcal{Q}$ that specifies the insertion of a finite sequence of actions into the program's action sequence and $\omega : \mathcal{A} \times \mathcal{Q} \to \{-, +\}$ indicates whether or not the action in question should be suppressed (-) or emitted (+). Also here the domain of γ is disjointed from the domain of δ in order to have a deterministic automata.

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

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

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

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

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

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

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

otherwise

3 Our goal

In this paper we give the semantic of some new process algebra operators that are *controller operators*, denoted by $Y \triangleright_{\mathbf{K}} X$ where $\mathbf{K} \in \{T, S, I, E\}$[3]. These can permit to control the behavior of the unknown component X, given the behavior of a control program Y.

These operators model the enforcement mechanism and security automata, whose semantic we have given in the previous section. In this way it is possible to apply all results of the logical approach to these automata. In particular we are able to automatically synthesize the appropriate controlling program Y, for each operator $\triangleright_{\mathbf{K}}$ using the satisfiability procedure for the μ-calculus that allows to obtain a model for a logical formula.

3.1 Our controller operators in process algebra

We introduce the following controller operators: $\triangleright_T$, $\triangleright_S$, $\triangleright_I$ and $\triangleright_E$ in order to model truncation, suppression, insertion and edit automata, respectively. To be able to compare security automata with our controllers, it is crucial to have a rigorous definition

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

of the semantic rules that describe the behavior of each operator. We denote with E the program controller and with F the program whose behavior we have to control. We work, without loss of generality, under the additional assumption that E and F never performs internal action τ. It is important to remark that we consider finite-state process in order to be able to apply the Theorem 1.

Truncation automata: $\triangleright_T$

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

This operator ([5]) model Schneider's automaton (and truncation automaton). Its semantic rule means that if F performs the right action a, i.e., the same action performed by E, then $E \triangleright_T F$ performs the action a, otherwise it halts. The following proposition holds.

Proposition 1. *Every sequence of actions that is an output of a* truncation automata $(\mathcal{Q}, q_0, \delta)$ *is also derivable from* $\triangleright_T$ *and vice-versa.*

Suppression automata: $\triangleright_S$

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

where $-a$ is a control action not in Act, so it does not admit a complementary action. This action is made by the process E in order to verify if the process F performs the action a which is a not admitted action (E does not perform a). If F performs the same action performed by E also $E \triangleright_S F$ performs it. On the contrary, if F performs some action a that E does not perform then E checks it performing a control action $-a$; $E \triangleright_S F$ performs the action τ that permits to *suppress* the action a,i.e. a becomes not visible from external observation. Thus it is trivially modeled by the internal action τ. In every other cases $E \triangleright_S F$ halts. The following proposition holds.

Proposition 2. *Every sequence of actions that is an output of a* suppression automata $(\mathcal{Q}, q_0, \delta, \omega)$ *is also derivable from* $\triangleright_S$ *and vice-versa.*

Insertion automata: $\triangleright_I$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E \triangleright_I F \xrightarrow{a} E' \triangleright_I F'} \qquad \frac{E \not\xrightarrow{a} E' \quad E \xrightarrow{+a.b} E' \quad F \xrightarrow{a} F'}{E \triangleright_I F \xrightarrow{b} E' \triangleright_I F}$$ [4]

where $+a$ is an action not in Act, so it doesn't admit a complementary action. The process E performs $+a$ to verify if the process F is going to perform the action a or not. Informally this two rules mean that if F performs the same action performed by E also $E \triangleright_I F$ performs it. If F performs some action that E does not perform, E detects it performing a control action $+a$ and subsequently performs an action b. Thus the entire system performs the action b. It is possible to note that in the description of insertion

[4] 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 $DerE$.

automata in [2] the domain of γ and δ is disjoint. In our case this is guarantee by the premise of the second rule in which we have that $E \stackrel{a}{\not\longrightarrow} E'$, $E \stackrel{+a.b}{\longrightarrow} E'$. 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 a sequence of actions 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. The following proposition holds.

Proposition 3. *Every sequence of actions that is an output of a* insertion automata $(\mathcal{Q}, q_0, \delta, \gamma)$ *is also derivable from* $\triangleright_I$ *and vice-versa.*

Edit automata: $\triangleright_{\boldsymbol{E}}$ In order to do insertion and suppression together we define the following controller operator. Its rule is the union of the rules of the $\triangleright_S$ and $\triangleright_I$.

$$\frac{E \stackrel{a}{\rightarrow} E' \quad F \stackrel{a}{\rightarrow} F'}{E \triangleright_E F \stackrel{a}{\rightarrow} E' \triangleright_E F'} \qquad \frac{E \stackrel{-a}{\longrightarrow} E' \quad F \stackrel{a}{\rightarrow} F'}{E \triangleright_E F \stackrel{\tau}{\rightarrow} E' \triangleright_E F'} \qquad \frac{E \stackrel{a}{\not\rightarrow} E' \quad E \stackrel{+a.b}{\rightarrow} E' \quad F \stackrel{a}{\rightarrow} F'}{E \triangleright_E F \stackrel{b}{\rightarrow} E' \triangleright_E F}$$

This operator combines the power of the previous two ones. The following proposition holds.

Proposition 4. *Every sequence of actions that is an output of an* edit automata $(\mathcal{Q}, q_0, \delta, \gamma, \omega)$ *is also derivable from* $\triangleright_E$ *and vice-versa.*

It is important to note that we introduced the control action $-a$ in the semantic of $\triangleright_S$ and $+a$ in the semantic 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 and tractable semantics (especially when trying to extend them to the timed setting).

4 Synthesis of controller programs

In our logical approach we are able to build a program controller Y using the Theorem 1. Such Y allows to enforce a desired security property for any target system X. We present here an extension of the reasoning we have done in [5]. In this case we have 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 an unknown agent, i.e. we do not know what is the behavior of X. At the beginning we have the system $S \| X$, and we want that it enjoys a security property expressed by a logical formula ϕ,i.e., $\forall X$ $(S \| X) \backslash L \models \phi$.

By using the partial model checking approach proposed in [11,12], we can focus on the properties of the possibly un-trusted component X, i.e., $\forall X$ $X \models \phi'$ where $\phi' = \phi_{//_{S,\backslash L}}$.

We wonder if there exists an implementation that can plugged into the system replacing the unspecified one, by satisfying some properties of the whole system. For this

reason we use the controller operators. We consider the previous equation where instead of X we put $Y \triangleright_{\mathbf{K}} X$ as follows

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

So we want to find a control program Y. In order to manage the universal quantification in (2), we prove the following proposition.

Proposition 5. *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 and*

$$f_S(a) = \begin{cases} a \text{ if } a \in Act \\ \tau \text{ if } a = -a \end{cases} \quad f_S(a) = \begin{cases} a \text{ if } a \in Act \\ \tau \text{ if } a = -a \end{cases} \quad f_I(a) = \begin{cases} a \text{ if } a \in Act \\ \tau \text{ if } a = +a \end{cases}$$

Moreover we consider equational μ-calculus formulae without $\langle\_\rangle$ modality, namely Fr_μ. It is easy to prove that this set of formulae is close for partial model checking function. The interest in this subclass of μ-calculus formulas is that it corresponds to the safety one. The following result holds.

Proposition 6. *Let* E *and* F *be two finite state processes and* $\phi \in Fr_\mu$. *If* $F \preceq E$ *then* $E \models \phi \Rightarrow F \models \phi$

Hence the equation (2) becomes $\exists Y$ s.t. $Y[f_{\mathbf{K}}] \models \phi'$. Applying partial model checking for relabeling function, we obtain

$$\exists Y \quad Y \models \phi'' \text{ where } \phi'' = \phi'_{//_{[f_{\mathbf{K}}]}} \tag{3}$$

for every $\mathbf{K}$. The formulation (3) is easier to be managed than (2). In particular, it is a satisfiability problem in μ-calculus and so it can be solved by obtaining a model Y.

5 Timed setting

In this section we extend to a timed setting the theory that we have developed above. First of all we show some notions useful to describe a very simple timed setting.

5.1 *GSOS* and CCS process algebra with time

We follow a simple approach, where time is discrete, actions are durationless and there is one special $tick$ action to represent the elapsing of time (see [13]). These are the feature of the so called *fictitious clock* approach of, e.g. [14,15,16]. A global clock is supposed to be update whenever all the processes of the system agree on this, by globally synchronizing an action $tick$. Hence, between the two global synchronizations on action $tick$ all the processes proceed asynchronously by performing durationless actions. So, the $tick$ action is important in parallel operator whose semantic, in this case, is enriched of this one more rule in addition of rules given in Table 1.

$$\frac{E_1 \xrightarrow{tick} E_1' \quad E_2 \xrightarrow{tick} E_2'}{E_1 \| E_2 \xrightarrow{tick} E_1' \| E_2'}$$

5.2 Behavioral equivalence

As done in [13], where security (in particular) information flow properties were defined in a timed setting, we consider the class of processes that do allow time proceed, the so-called *weakly time alive* processes. These represent *correct* attackers w.r.t. time. (As a matter of fact, it is not realistic that an intruder or a malicious agent can block the flow of time.)

Definition 3. *A process E is* directly weakly time alive iff $E \stackrel{tick}{\Longrightarrow}$ [5], *while it is* weakly time alive iff *for all $E' \in Der(E)$, we have E' is directly weakly time alive.*

Since $E \stackrel{\alpha}{\longrightarrow} E'$ implies $Der(E') \subseteq Der(E)$, it directly follows that if E is weakly time alive, then any derived E' of E is weakly time alive as well. Moreover, it is worthwhile noticing that the above property is preserved by the parallel composition.

The behavioral relation considered is the timed versions of weak bisimulation [8]. This equivalence permits to abstract to some extent from the internal behavior of the systems, represented by the invisible τ actions. [6]

Definition 4. *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* timed weak simulation*, denoted by $\preceq_t$, over $(\mathcal{E}, \mathcal{T})$ if and only if, whenever $(E, F) \in \mathcal{R}$ we have:*

- *if $E \stackrel{a}{\longrightarrow} E'$ then there exists F' s.t. $F \stackrel{a}{\Longrightarrow} F'$ and $(E', F') \in \mathcal{R}$,*
- *if $E \stackrel{tick}{\longrightarrow} E'$ then there exists F' s.t. $F \stackrel{tick}{\Longrightarrow} F'$ and $(E', F') \in \mathcal{R}$.*

Moreover, a binary relation $\mathcal{R}$ over $\mathcal{E}$ is said a timed weak bisimulation *(denoted by $\approx_t$) over the LTS of concurrent processes $(\mathcal{E}, \mathcal{T})$ if both $\mathcal{R}$ and its converse are timed weak simulation.*

5.3 Partial model checking with time

Introducing the new $tick$ action to model the elapsing of time, we have one more case to consider in the definition of partial model checking function. The $tick$ action cannot be consider as the other actions in Act. Hence we extend the *pmc* function to deal with time by adding the following rules

$$\langle tick\rangle A//s = \begin{cases} \langle tick\rangle A//s' & s \stackrel{tick}{\longrightarrow} s' \\ \mathbf{F} & otw \end{cases} \qquad [tick]A//s = \begin{cases} [tick]A//s' & s \stackrel{tick}{\longrightarrow} s' \\ \mathbf{T} & otw \end{cases}$$

It is easy to note that the insertion of $tick$ action affects only the pmc for parallel operator. The partial evaluation function for relabeling and restriction are not affected.

5.4 Our controller operators in a timed setting

In this section we study how the controller operators that we have define in Section 3.1 work in a timed setting. We want that $Y \triangleright_{\mathbf{K}} X$, for each $\mathcal{K}$, are processes that do allow time to proceed, so we prove that it is *weakly time alive*. Here we use the following notation: E and F are finite state processes. E is the program controller and F the program whose behavior we want to control. The following proposition holds.

[5] This means that we are no interested to the final state of the transition.

[6] Other equivalences are in between trace and bisimulation semantics. We do not intend to discuss here their relative merits.

Proposition 7. *Let E and F be two finite-state processes. If both E and F are weakly time alive, also $E \triangleright_{\mathbf{K}} F$ is weakly time alive.*

Dealing with time we do not change or modify the semantic of our controllers. Hence a proposition similar to proposition 5 holds. In particular, looking at the definition of weak timed simulation and at the proof of the proposition 5, given in appendix, the following proposition holds.

Proposition 8. *For every $\mathbf{K} \in \{T, S, I, E\}$ the following relation holds $E \triangleright_{\mathbf{K}} F \preceq_t E[f_{\mathbf{K}}]$ where $f_{\mathbf{K}}$ is a relabeling function definition of which depend on $\mathbf{K}$.*

We can then recast the results of the previous section in a timed setting.

6 A simple example

Consider the process $S = a.b.\mathbf{0}$ and consider the following equational definition $Z =_\nu [\tau]Z \wedge [a][[c]]^7\mathbf{F}$. It asserts that after every action a cannot be perform an action b. Let $Act = \{a, b, c, \tau, \bar{a}, \bar{b}, \bar{c}\}$ be the set of actions. We can apply the partial evaluation for the parallel operator we obtain after some simplifications the following system of equation, that we denoted with $\mathcal{D}$

$Z_{//_S} =_\nu [\tau]Z_{//_S} \wedge [\bar{a}]Z_{//_{S'}} \wedge [a]W_{//_S} \wedge W_{//_{S'}}$
$W_{//_{S'}} =_\nu [\tau]W_{//_{S'}} \wedge [\bar{b}]\mathbf{T} \wedge [c]\mathbf{F}$
$Z_{//_{S'}} =_\nu [\tau]Z_{//_{S'}} \wedge [\bar{b}]\mathbf{T} \wedge [a]W_{//_{S'}}$
$W_{//_S} =_\nu [\tau]W_{//_S} \wedge [\bar{a}]W_{//_{S'}} \wedge [c]\mathbf{F}$

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 and we obtain that $\mathcal{D}_{//_{f_S}}$ is

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

It is easy to 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 $\mathcal{D}$. For instance, consider $X = a.c.\mathbf{0}$. Looking at the first rules 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 preserve its security since the actions performed by the component X have been prevented from being visible outside.

[7] We define $[[c]]\phi$ as $\neg\langle\langle c\rangle\rangle\neg\phi$ which is the formula that captures in the strong logic the diamond modality in the weak transition systems (see [17]).

Acknowledgement We thank the anonymous referees of WITS06 for valuable comments that helped us to improve this paper.

References

1. Schneider, F.B.: Enforceable security policies. ACM Transactions on Information and System Security **3**(1) (2000) 30–50
2. Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies. In Cervesato, I., ed.: Foundations of Computer Security: proceedings of the FLoC'02 workshop on Foundations of Computer Security, Copenhagen, Denmark, DIKU Technical Report (2002) 95–104
3. Bartoletti, M., Degano, P., Ferrari, G.: Policy framings for access control. In: Proceedings of the 2005 workshop on Issues in the theory of security table of contents, Long Beach, California (2005) 5 – 11
4. Ligatti, J., Bauer, L., Walker, D.: Edit automata: Enforcement mechanisms for run-time security policies. International Journal of Information Security **4**(1–2) (2005) 2–16
5. Martinelli, F., Matteucci, I.: Partial model checking, process algebra operators and satisfiability procedures for (automatically) enforcing security properties (2005) Presented at the International Workshop on Foundations of Computer Security (FCS05).
6. Ligatti, J., Bauer, L., Walker, D.: Enforcing non-safety security policies with program monitors. In: 10th European Symposium on Research in Computer Security (ESORICS). (2005)
7. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. J.ACM **42**(1) (1995)
8. Milner, R.: Communicating and mobile systems: the π-calculus. Cambridge University Press (1999)
9. Street, R.S., Emerson, E.A.: An automata theoretic procedure for the propositional μ-calculus. Information and Computation **81**(3) (1989) 249–264
10. Andersen, H.R.: Partial model checking. In: LICS '95: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society (1995) 398
11. Martinelli, F.: Partial model checking and theorem proving for ensuring security properties. In: CSFW '98: Proceedings of the 11th IEEE Computer Security Foundations Workshop, IEEE Computer Society (1998)
12. Martinelli, F.: Towards automatic synthesis of systems without informations leaks. In: Proceedings of Workshop in Issues in Theory of Security (WITS). (2000)
13. R.Focardi, R.Gorrieri, F.Martinelli: Real-time Information Flow Analysis. IEEE JSAC (2003)
14. Corradini, F., D'Ortenzio, D., Inverardi, P.: On the relationships among four timed process algebras. Fundam. Inform. **38**(4) (1999) 377–395
15. Hennessy, M., Regan, T.: A temporal process algebra. In: FORTE '90: Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, North-Holland (1991) 33–48
16. Ulidowski, I., Yuen, S.: Extending process languages with time. In: AMAST '97: Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology, London, UK, Springer-Verlag (1997)
17. Müller-Olm, M.: Derivation of characteristic formulae. In: MFCS'98 Workshop on Concurrency. Volume 18 of Electronic Notes in Theoretical Computer Science (ENTCS)., Elsevier Science B.V. (1998)