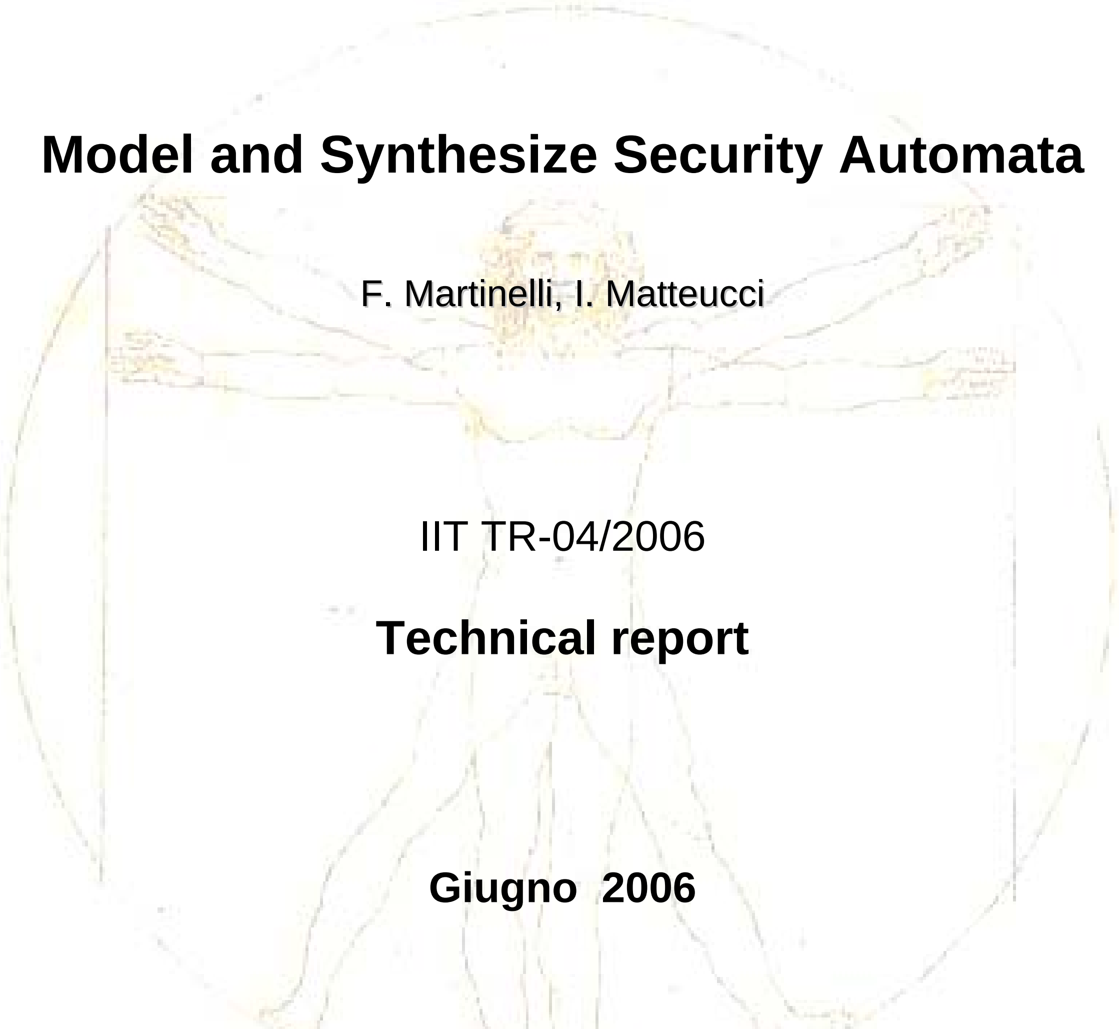
Consiglio Nazionale delle Ricerche



# Model and Synthesize Security Automata

F. Martinelli, I. Matteucci

IIT TR-04/2006

**Technical report**

**Giugno 2006**

**Istituto di Informatica e Telematica**

# Model and Synthesize Security Automata [*]

Fabio Martinelli[1], Ilaria Matteucci[1,2]
Istituto di Informatica e Telematica - C.N.R., Pisa, Italy[1]
{Fabio.Martinelli, Ilaria.Matteucci}@iit.cnr.it
Dipartimento di Scienze Matematiche ed Informatiche,
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 [18] and by Ligatti and al. in [4], respectively. We also show how to automatically build these controllers for given security policies.

## 1 Overview

Recently, several papers tackled the formal definition of mechanisms for enforcing security policies (e.g., see [3, 4, 7, 12, 14, 18]).

The focus of this paper is the study of the enforcement mechanisms introduced by Schneider in [18] and security automata developed by Ligatti and al. in [4, 7].

In [18], Schneider deals with the problem of enforcing 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 enforcement mechanisms (EM) that work by monitoring execution steps of a *target* system, herein and terminating its execution if it is about to violate the security property being enforced.

A security automaton defined in [18] 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 security-relevant actions, is the transition function. A security automata processes a sequence $a_1 a_2 \ldots$ of actions. At each step only one action is considered and for each action we calculate the *global state* $Q'$ that is the set of the possible states for the current action, i.e. if the automaton is checking the action $a_i$ then $Q' = \bigcup_{q \in Q'} \delta(a_i, q)$. If the automaton can make a transition on a given action, 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 execution is terminated. A security property that can be enforced in this way corresponds to a safety property (according to [18], a property is a safety

[*] 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).

one, if whenever it does not hold in trace then it does not hold in any extension of this trace).

Starting from the work of Schneider described above, Ligatti and al. in [4, 7] have defined four different deterministic security automata which deal with finite sequences of actions: the **truncation automaton** (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 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 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 insert or suppress security-relevant actions at will.

These works have been extended by studying how truncation automata and edit automata work on possible infinite sequence of actions (see [8]). 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 (see [15]) able to mimic the behavior of the security automata briefly described above. The process algebra operators $Y \triangleright_{\mathbf{K}} X$ (where $\mathbf{K}$ is the name of the corresponding automata) act as programmable controllers ($Y$) of a target system ($X$).

We can then exploit a huge theory for security analysis based on process algebra theory. In particular, depending on the kind of security automata one chooses, we show how to automatically build programs that allow to enforce security properties for whatever target system. Since many properties of systems are naturally specified by means of fixed points, the $\mu$-calculus is an expressive and important specification language.

We automatically synthesize the appropriate controlling program $Y$ for an operator $\triangleright_{\mathbf{K}}$, given the security property $\phi$ expressed by a $\mu$-calculus formula. The synthesis is based on a satisfiability procedure for the $\mu$-calculus that allows to obtain a model for a logical formula (in our framework a suitable property), i.e., it is possible to decide if there exists a model of a given logical formula. In particular, for truncation automata we show a method to build the maximal model.
This work represents a significant contribution to the previous works (see [4, 7, 8, 18]), 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. Our approach is that it actually starts from a property that the overall system must enjoy, say $\phi$ and, using the *partial model* *checking* technique, projects this property on another one that only the component $X$ must satisfy, say $\phi'$. This allows one to monitor only the necessary/untrusted part of the system. Thus we can now force $X$ to enjoy $\phi'$ 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 (SOS), logic and security automata. Section 3 describes some process algebra operators (controllers) corresponding to the 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 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 [5]). It permits to reason compositionally about the behavior of program 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* $\Sigma$ 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 $\Sigma$-*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 $\gamma$ 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:

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

where all variables are distinct; $\vec{x}$ and $\vec{y}$ 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 $\zeta$ is *effective* for a term $f(s_1, \ldots, s_k)$ and a rule $r$ if:

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

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 $\zeta$ for a rule $r$ with operator $f$ and action $c$ s.t. $s = \zeta(g(\vec{x}, \vec{y}))$. There exists a unique transition relation induced by a *GSOS* system (see [5]) and this transition relation is *finitely branching*.

### 2.1.2 An example: CCS process algebra

$CCS$ of Milner (see [16]) 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) $\tau$.

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 $^-$is a bijection with $\bar{\bar{a}} = a$, $Act$ be $\mathcal{L} \cup \bar{\mathcal{L}} \cup \{\tau\}$, where $\tau$ is a special action that denotes an internal computation step (or communication) and $\Pi$ 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 2. 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 Behavioral equivalence

It is often necessary to compare processes that are expressed using different terms but have the same behavior.

### 2.2.1 Strong and weak bisimulations

We recall some useful relations between processes (see [16]). Now we give some preliminary definition. In the following, we let $\hat{\tau} = \epsilon$ and for action $a \neq \tau$ $\hat{a} = a$.

**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 $(\mathcal{E}, \mathcal{T})$ iff, whenever $(E, F) \in \mathcal{R}$ we have: if $E \xrightarrow{a} E'$ then $\exists F' \in \mathcal{E}$ s. t. $F \xrightarrow{a} F'$ 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 [5], 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 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.

First of all we present the notion of *observational relation* is the following: $E \overset{\tau}{\Rightarrow} E'$ (or $E \Rightarrow E'$) if $E \xrightarrow{\tau}^{*} E'$ (where $\xrightarrow{\tau}^{*}$ is the reflexive and transitive closure of the $\xrightarrow{\tau}$ relation); $E \overset{\hat{\alpha}}{\Rightarrow} E'$ if $E \overset{\tau}{\Rightarrow} \xrightarrow{\hat{\alpha}} \overset{\tau}{\Rightarrow} E'$.

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* 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 $(E', F') \in \mathcal{R}$. 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 [16]).

## 2.3 Equational $\mu$-calculus and partial model checking

Equational $\mu$-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:

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

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

where the symbol $\mathbf{T}$ means *true* and $\mathbf{F}$ means *false*; $\wedge$ is the symbol for the conjunction of formulae, i.e. the conjunction $A_1 \wedge A_2$ holds iff both of the formulae $A_1$ and $A_2$ hold, and $\vee$ is the disjunction of formulae and $A_1 \vee A_2$ holds when at least one of $A_1$ and $A_2$ holds. Moreover the meaning of $\langle a \rangle A$ (*possibility operator*) is ”it is possible to do an $a$-action to a state where $A$ holds” and the meaning of $[a]A$ (*necessity operator*) is ”for all $a$-actions performed $A$ 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 3 in appendix.

The following standard result of $\mu$-calculus will be useful in the reminder of the paper.

**Theorem 1 ([20])** *Given a formula $\phi$ it is possible to decide in exponential time in the length of $\phi$ if there exists a model of $\phi$ 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 [2]). The intuitive idea underlying the *pmc* is the following: proving that $E \| F$ satisfies a formula $\phi$ ($E \| F \models \phi$) 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 $\phi$ is specified by use the *equational $\mu$-calculus*. A useful result of partial model checking is the following.

**Lemma 1 ([2])** *Given a process $E \| F$ and a formula $\phi$ we have: $E \| F \models \phi$ iff $F \models \phi_{//E}$.*

The reduced formula $\phi_{//E}$ depends only on the formula $\phi$ 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 to make a successful attack on 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 1 holds for every process algebra operators (see [2]). The partial model checking functions for parallel operator, relabeling and restriction are given in Table 4 and Table 5 in appendix.

## 2.4 Characteristic formulae

A *characteristic formula* is a formula in equational $\mu$-calculus that completely characterizes the behavior of a (state in a) LTS modulo a chosen notion of behavioral relation. It is possible to define the notion of characteristic formula for a given finite state process $E$ w.r.t. weak bisimulation as follows (see [17]).

**Definition 3** *Given a finite state process $E$, its characteristic formula (w.r.t. weak bisimulation) $D_E \downarrow X_E$ is defined by the following equations for every $E' \in Der(E)$, $X_{E'} =_\nu (\bigwedge_{a;E'':E' \xrightarrow{a} E''} \langle\langle \hat{a} \rangle\rangle X_{E''}) \wedge (\bigwedge_{a \in Act}([a](\bigvee_{E'':E' \overset{\hat{a}}{\Rightarrow} E''} X_{E''})))$*

where $\langle\langle\hat{a}\rangle\rangle$ is the equivalent of the modality operator $\langle\hat{a}\rangle$ w.r.t. weak bisimulation, which can be introduce as abbreviation (see [17]):

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

The following lemma characterizes the power of these formulae.

**Lemma 2 ([17])** *Let $E_1$ and $E_2$ be two finite-state processes. If $\phi_{E_2}$ is characteristic for $E_2$ then:*

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

It is possible to define the notion of characteristic formula for a finite state process E w.r.t. weak simulation as follows.

**Definition 4** *Given a finite state process, its characteristic formula (w.r.t. weak simulation) $D_E \downarrow X_E$ is defined by the following equations for every $E' \in Der(E)$, $X_{E'} =_\nu \bigwedge_{a\in Act}([a](\bigvee_{E'':E' \overset{\hat{a}}{\Rightarrow} E''} X_{E''}))$*

Following the reasoning used in [17] for the definition of characteristic formula w.r.t strong bisimulation. The following proposition holds.

**Lemma 3** *Let $E$ be a finite-state process and let $\phi_{E,\preceq}$ be its characteristic formula w.r.t. weak simulation, $F \preceq E \Leftrightarrow F \models \phi_{E,\preceq}$*

## 2.5 Enforcement mechanisms and Security automata

In this paper we chose to follow the semantic approach given by Ligatti and al. in [4] to describe the behavior of four different kind of security automata.

A *security automaton* at least consist of a (countable) set of states, $\mathcal{Q}$, a set of actions $Act$ and a transition (partial) function $\delta : Act \times \mathcal{Q} \to \mathcal{Q}$. We use also $\sigma$ to denote a sequences of actions, $\cdot$ for the empty sequence and $\tau$[1] to represent an internal action.

The execution of each different kind of security automata is specified by a labeled operational semantics. The basic single-step judgment has the form $(\sigma, q) \xrightarrow{a} (\sigma', q')$ where $\sigma'$ and $q'$ denote the action sequence and state after the automaton takes a single step, and $a$ denotes the sequence of actions produced by the automaton. The single-step judgment can be generalized to a multi-step judgment ($(\sigma, q) \overset{\gamma}{\Longrightarrow}$[2] $(\sigma', q')$), where $\gamma$ is a sequences of actions, as follows.

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

The operational semantics for each security automaton is the following.

[1]In [4] internal actions are denoted by $\cdot$. We use $\tau$ because we use process algebras where internal actions are commonly denoted by $\tau$.

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

**Truncation automaton.** The operational semantic of truncation automata is:

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

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

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

otherwise.

**Suppression automaton.** It is define as $(\mathcal{Q}, q_0, \delta, \omega)$ where $\omega : Act \times \mathcal{Q} \to \{-, +\}$ indicates whether or not the action in question should be suppressed (-) or emitted (+).

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

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

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

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

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

otherwise.

**Insertion automaton.** It is define as $(\mathcal{Q}, q_0, \delta, \gamma)$ where $\gamma : Act \times \mathcal{Q} \to Act \times \mathcal{Q}$ that specifies the insertion of an action into the sequence of actions of the program. It is necessary to note that in [4, 7] 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 $\gamma$. 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 $\gamma$ is disjoint from the domain of $\delta$ in order to have a deterministic automata;

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

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

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

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

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

otherwise.

**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 action sequence and $\omega : Act \times \mathcal{Q} \rightarrow \{-, +\}$ indicates whether or not the action in question should be suppressed (-) or emitted (+). Also here the domain of $\gamma$ is disjoint from the domain of $\delta$ 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 Modeling security automata with process algebra

In this Section we give the semantics of some process algebra operators that act as *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 (possibly untrusted) component $X$, given the behavior of the control program $Y$.

### 3.1 Our controller operators in process algebra

To compare security automata with our controllers, it is crucial to have a rigorous definition of the semantic rules that describe the behavior of each operator. 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 $\tau$.

#### 3.1.1 Truncation automata: $\triangleright_T$

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

This operator models the truncation automaton that is similar to Schneider's automaton (when considering only deterministic automata, e.g., see [4, 7]). Its semantic 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 $E \triangleright_T F$ performs the action $a$, otherwise it halts. The following proposition holds.

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

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

### 3.1.2 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). As for the truncation automaton, if $F$ performs the same action performed by $E$ also $E \triangleright_S F$ 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 $E \triangleright_S F$ performs the action $\tau$ that *suppresses* the action $a$, i.e., $a$ becomes not visible from external observation. Otherwise, $E \triangleright_S F$ halts. The following proposition holds.

**Proposition 2** *Each sequences 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.*

### 3.1.3 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 $Act$. 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 $+a$, then the whole system perform the an action $b$. It is possible to note that in the description of insertion automata in [4] the domains of $\gamma$ and $\delta$ 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 $\delta$ and it is in the domain of $\gamma$ 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 $F$. The following proposition holds.

**Proposition 3** *Each sequences 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.*

### 3.1.4 Edit automata: $\triangleright_E$

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 $\triangleright_I$.

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

[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 $Der(E)$.

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

**Proposition 4** *Each sequences 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.

## 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 [12]. 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 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 $\phi$, i.e., $\forall X$ $(S\|X)\backslash L \models \phi$. In order to protect the system we may simply check the correctness of each process $X$ before it is executed or, if it is not possible (or not desirable), we may define a controller that, in any case, forces each process to behave correctly.

We study here how to build a program controller in order to force the intruder to behave correctly, i.e. as prescribed by the formula $\phi$. Thus, we want to find a control program $Y$ such that:

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

By using the partial model checking approach proposed in [11], 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 $\phi' = \phi_{//S,\backslash L}$. In order to manage the universal quantification in (3), 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 on* $Act$[5] *and*

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

Now we restrict ourselves to a subclass of equational $\mu$-calculus formulae that is denoted by $Fr_\mu$. This class consists in equational $\mu$-calculus formulae without $\langle \_ \rangle$. It is easy to prove that this set of formulae is close for partial model checking function. The following result holds.

[5]Here the set $Act$ must be consider enriched by control actions

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

At this point in order to check the equation (3) it is sufficient to check:

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

To further reduce the previous equation, 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:

$$\exists Y \quad Y \models \phi''_{\mathbf{K}} \tag{4}$$

In this way we reduce ourselves to a satisfiability problem in $\mu$-calculus that can be solved by Theorem 1.

## 5 Automated synthesis of Schneider's controller operator

In this section we synthesize a maximal program controller $Y$ for the operator $Y \triangleright_T X$ by exploiting the theory developed by Walukiewicz in [13, 21].

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 $\phi$* iff $E \models \phi$ and $\forall E'$ s.t. $E' \models \phi$, $E' \preceq E$.

Informally, the maximal program controller $Y$ is the process that restricts as less as possible the activity of the target $X$.

Usually the discovered model is a non-deterministic process. In order to find a deterministic model we consider a subset of formulae of $Fr_\mu$ without $\vee$. This set of formulae is called the *universal conjunctive $\mu$-calculus formulae*, $\forall_\wedge \mu C$ (see [6]).

**Definition 5** *The set $\forall_\wedge \mu C$ of* universal conjunctive $\mu$-calculus formulae *is the largest subset of equational $\mu$-calculus formulae that can be written without either the $\vee$ operator and the $\langle \_ \rangle$ modality.*

**Proposition 7** *Given a formula $\phi \in \forall_\wedge \mu C$, a maximal deterministic model $E$ of this formula exists.*

Due to the fact that Schneider in his article [18] is interested in *trace of executions*[6], we assume that the process with a good behavior is deterministic, i.e., we are interested in properties of the form $(E \| X)\backslash L \preceq E\backslash L$ where $E\backslash L$ a deterministic process. Hence the characteristic formula of $E$, $X_{E'} =_\nu \bigwedge_{a \in Act}([a](\bigvee_{E'':E' \overset{\hat{a}}{\Rightarrow} E''} X_{E''})))$, becomes simpler because $\bigvee_{E'':E' \overset{\hat{a}}{\Rightarrow} E''} X_{E''}$ is reduced either to $X_{E''}$ or to false. So it is in $\forall_\wedge \mu C$.

In order to apply our logical approach based on partial model checking we also need to ensure that after the partial model checking phase for the characteristic formula

[6]For any $E \in \mathcal{E}$ the set $Tr(E)$ of *traces associated with* $E$ is $Tr(E) = \{\gamma \in (Act\backslash\{\tau\})^* | \exists E' : E \overset{\gamma}{\Longrightarrow} E'\}$.

we still get a formula in $\forall_\wedge\mu C$ whose satisfiability procedure returns a deterministic process. This actually holds.

**Proposition 8 ([6])** $\forall_\wedge\mu C$ *is closed under the partial model checking function.*

Thus, by using the result in proposition 7, it is possible to find a maximal deterministic model that synthesizes a controller operator to force a security policy, i.e. the synthesis of a truncation automaton for a component that will allow the whole system to enjoy the desired security property.

## 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 $c$. 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 $\mathcal{D}$:

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

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 $\mathbf{T}$. We obtain that $\mathcal{D}_{//f_S}$ is:

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

We can note 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 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:

$$(a.b.\mathbf{0}\|(-c.\mathbf{0} \triangleright_S c.\mathbf{0})) \xrightarrow{\tau} (a.b.\mathbf{0}\|\mathbf{0} \triangleright_S \mathbf{0})$$

[7] We define $[[c]]\phi$ as $\neg\langle\langle c\rangle\rangle\neg\phi$ where $\langle\langle c\rangle\rangle$ is defined as follows: $\langle\langle\epsilon\rangle\rangle\phi \overset{def}{=} \mu X.\phi \vee \langle\tau\rangle X$, $\langle\langle c\rangle\rangle\phi \overset{def}{=} \langle\langle\epsilon\rangle\rangle\langle c\rangle\langle\langle\epsilon\rangle\rangle\phi$. (see [17]).

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

## 7 Discussion on enforcing techniques

### 7.1 Security automata

As we have already said, one way to enforce security properties is with a *monitor* that runs in parallel with the *target* program (see [18]). A program monitor can be formally modeled by a *security automaton*. A *security automaton* defined in [18] is a triple $(\mathcal{Q}, q_0, \delta)$ where $\mathcal{Q}$ is a set of states, $q_0$ is the initial one and $\delta : Act \times \mathcal{Q} \to \mathcal{Q}$, where $Act$ is a set of security-relevant actions, is the transition function. A security automata processes a sequence $a_1 a_2 \ldots$ of actions. At each step only one action is considered and for each action we calculate the *global state* $Q'$ that is the set of the possible states for the current action, i.e. if the automaton is checking the action $a_i$ then $Q' = \bigcup_{q \in Q'} \delta(a_i, q)$. If the automaton can make a transition on a given action, 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 execution is terminated. Thus, at every step, it verifies if the action is in the set of the possible actions or not. As we have already shown, in [7] four different kind of security automata are defined. To study the cost in term of time of these security automata, we must consider how much transition function costs. Since the security automata we consider are deterministic (thus $Q'$ would be either a singleton or empty), by using the standard graphical representation through matrix, it is easy to understand that the cost in time is $O(1)$. Thus, given a sequence of $n$ actions, we need $O(n)$ to check whether this sequence is acceptable or not.

#### 7.1.1 Enforceable properties with Security automata

Referring to [18], the truncation automata is able to enforce security property that corresponds to a safety property (according to [18], a property is a safety one, if whenever it does not hold in trace then it does not hold in any extension of this trace).

Refer to [7] we consider the *effectively enforcement*. This definition of enforcement uses a system-specific equivalence relation ($\cong$) on executions that is reflexive, symmetric and transitive. Moreover, any property that we might consider should not distinguish equivalent sequences:

$$\sigma \cong \sigma' \Rightarrow P(\sigma) \Leftrightarrow P(\sigma')$$

where $P$ is the property that we want enforce.

**Definition 6** *A automaton $A$ with starting state $q_0$* effectively enforces *a property $P$ on the system with action set $Act$ iff* $\forall \sigma \in \overrightarrow{Act}$[8] $\exists q'\ \exists \sigma' \in \overrightarrow{Act}$*:*

*1.* $(\sigma, q_0) \overset{\sigma'}{\Longrightarrow}_A (\cdot, q')$

[8] $\overrightarrow{Act}$ is the set of sequences of actions

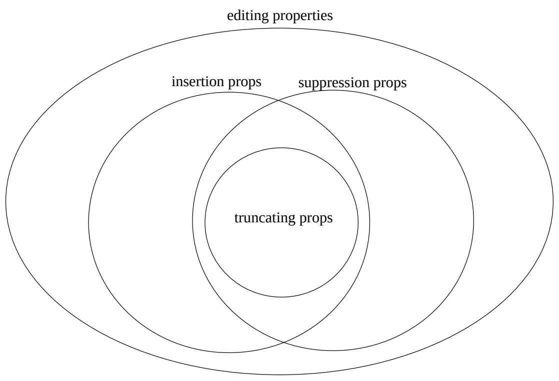


Figure 1: A taxonomy of *effectively* enforceable security properties

2. *$P(\sigma')$, and*

3. *$P(\sigma) \Rightarrow \sigma \cong \sigma'$.*

The power of the four different kinds of security automata is different as we show in Figure 1.

## 7.2 Model checking a path

A technique that is used in run-time verification is the *model checking a path*, i.e., solve the model checking problem on a single path instead on the whole model. It was introduced by Markey and Schnoebelen in [9].

In particular, this technique is developed for model checking of linear time logic. We recall some definitions, even if, we will not recall the syntax and semantics of $LTL$.

**Definition 7** *A* path *is a finite sequence of states $\pi = s_0, s1, \ldots$ where a* state *is a valuation $s \in 2^{AP}$ of the atomic proposition (namely $AP$). $|\pi| \in \mathbb{N} \cup \{\omega\}$ denotes the length of $\pi$.*

Let $L$ a linear logic.
**Path Model Checking for** $L$ (**PMC**($L$)):
**Input:** given a path $u$ and a temporal formula $\phi$ of $L$.
**Output:** yes iff $u \models \phi$, no otherwise.

Using standard dynamic programming methods a path can obviously be checked in bilinear time, $O(|path| \times |formula|)$. In particular, we recall here some important results on linear temporal logic.

**Theorem 2 ([9])** ***PMC**($LTL$) can be solved in time $O(|u| \times |\phi|)$ where $\phi$ is an $LTL$ formula.*

**Theorem 3 ([9])** ***PMC**($LTL + Past$) can be solved in time $O(|u| \times |\phi|)$ where $\phi$ is an $LTL + Past$ formula.*

| **Logic**: | **Relabeling**: |
|---|---|
| $FOMLO$ | PSPACE $-$ $complete$ |
| $LTL$ | PTIME $-$ $easy$ |
| $LTL + Past$ | PTIME $-$ $easy$ |
| $LTL + Past + Now$ | PTIME $-$ $complete$ |
| $LTL + Chop$ | PTIME $-$ $complete$ |

Table 1: Checking richly expressing logics on paths

In [9] article, authors study the complexity of the model checking a path on linear temporal logic and prove that for the first-order monadic logic of order the algorithm is PSPACE-complete. In [10], they prove that model checking a path of modal $\mu$-calculus formulae has the complexity of $O(|u| \times |\phi|^{ad})$ where $ad$ is the alternation depth (when dealing with safety properties it is 1).

We show in Table 1[9] all the complexity results.

### 7.2.1 Enforceable properties with model checking (paths)

We can use this approach in order to enforce security properties. This technique permit us to control if a target execution is correct or not. As we have already said, this techniques was developed in order to deal with run-time verification.

*The algorithm*: The behavior of the target is not known a priori. To every target action is associated a new target state. Thus every time an action is performed a new state is add to the path that have to be checked. Hence for every action we apply **PMC** on the new path. For example, let $\pi_t$ the sequence of states at time $t$. We can apply path model checking on $\pi_t$. If the output is ”yes” we allow that the target performs the next action $a$, otherwise we stop it. After an action $a$ the target goes in a new state $s_{t+1}$ and the sequences of state becomes $\pi_{t+1}$ and we repeat the same algorithm. Using standard dynamic programming, that works by memorizing outputs that are obtained at the previous step of the algorithm, the cost of this method is $O(|path| \times |\phi|)$ where $\phi$ is a $LTL$ formula.

We can note that this technique is developed o $LTL$ formulae and, in [9], the authors give the cost of the algorithm for $LTL$ formulae and for $LTL + Past$ formulae. These two logics are suitable for express safety properties.

## 7.3 Comparison

It is easy to note that using this technique it is easy to implement a controller with the same behavior of truncation automata. In fact, this technique permits to recognize a bad action but it does not give any advantage to repair it. Security automata instead allow us to modify the behavior of a target system to make it compliant with the policy.

[9]Take from [9]

This is a main difference from a declarative policy language as logic instead of an operational one as security automata.

## References

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

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

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

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

[5] B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *J.ACM*, 42(1), 1995.

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

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

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

[9] N. Markey and P. Schnoebelen. Model checking a path (preliminary report). In *Proc. Concurrency Theory (CONCUR'2003), Marseille, France*, volume 2761 of *Lect.Notes Comp. Sci*, pages 251–265. Springer, Aug. 2003.

[10] N. Markey and Ph. Schnoebelen. Mu-calculus path checking. *Information Processing Letters*, 97(6):225–230, Mar. 2006.

[11] F. Martinelli. 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] F. Martinelli 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.

[13] F. Martinelli and I. Matteucci. Model and synthesize security automata. 2006. www.iit.cnr.it/staff/fabio.martinelli/mssa.pdf.

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

[15] R. Milner. Synthesis of communicating behaviour. In *Proceedings of 7th MFCS*, Poland, 1978.

[16] R. Milner. *Communicating and mobile systems: the $\pi$-calculus*. Cambridge University Press, 1999.

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

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

[19] C. Stirling. Bisimulation, model checking, and other games, 1997. Mathfit instructional meeting on games and computation.

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

[21] I. Walukiewicz. *A Complete Deductive System for the $\mu$-Calculus*. PhD thesis, Institute of Informatics, Warsaw University, June 1993.

## A Tables

In this section there are all the table whose references are in the paper. The first table shows the operational semantics of $CCS$; the second one the equational $\mu$-calculus. The last two tables show how partial evaluation function works w.r.t. parallel, restriction and relabeling operators respectively.

## B Technical proofs

**Lemma 3** *Let $E$ be a finite-state process and let $\phi_{E,\preceq}$ be its characteristic formula w.r.t. weak simulation then $F \preceq E \Leftrightarrow F \models \phi_{E,\preceq}$.*

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 2: *GSOS* system for CCS.

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

We use $\sqcup$ to represent union of disjoint environments. Let $\rho$ be the environment ( a function from variables to values) and $\sigma$ be in $\{\mu, \nu\}$, then $\sigma U.f(U)$ represents the $\sigma$ 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* $\sigma$ *fixpoint solution* $U'$ *of* $[\![A]\!]$ *where the solution to the rest of the lists of equations* $D$ *is used as environment.*

Table 3: Equational $\mu$-calculus

*Proof*: In order to prove the following proposition we give the following chain:

$$F \preceq E \Leftrightarrow \forall a \;\; F \xrightarrow{a} F' \;\; \exists E' \;\; E \overset{a}{\Rightarrow} E' \wedge F' \preceq E' \Leftrightarrow$$
$$\forall a \;\; F \xrightarrow{a} F' \;\; F' \models \bigvee X_{E'} \Leftrightarrow \forall a \;\; F \models [a](\bigvee X_{E'}) \Leftrightarrow$$
$$F \models \bigwedge([a](\bigvee X_{E'}))$$

□

Before starting to prove proposition 1, 2, 3, 4, we note that in our controller operator the halt condition is not roundly given because this occurs when there are not rule that could be applied, i.e., when premises of all rules are not verify. As we have already note, also in security automata described in section 2.5, the action $\tau$ in stop rule of each automata is an internal action that is not really performed. So in our proofs, without loss of validity, we can omit the stop case because the stop rule of each automata is equivalent to the halt condition of respectively operator.

**Proposition 1** *Each 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.*

*Proof:* To simplify the notation, $(\sigma, q)$ denotes a generic state of automata and

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

Table 4: Partial evaluation function for parallel operator $E\|_-$.

| Restriction: | Relabeling: |
|---|---|
| $X//\backslash L = X$ | $X//[f] = X$ |
| $\langle a\rangle A//\backslash L = \begin{cases} \langle a\rangle(A//\backslash L) & \text{if } a \notin L \cup \bar{L} \\ \mathbf{F} & \text{if } a \in L \end{cases}$ | $\langle a\rangle A//[f] = \bigvee_{b:f(a)=b}\langle b\rangle(A//[f])$ |
| $[a]A//\backslash L = \begin{cases} [a](A//\backslash L) & \text{if } a \notin L \cup \bar{L} \\ \mathbf{T} & \text{if } a \in L \end{cases}$ | $[a]A//[f] = \bigwedge_{b:f(a)=b}\langle b\rangle(A//[f])$ |
| $A_1 \wedge A_2//\backslash L = (A_1//\backslash L) \wedge (A_2//\backslash L)$ | $A_1 \wedge A_2//[f] = (A_1//[f]) \wedge (A_2//[f])$ |
| $A_1 \vee A_2//\backslash L = (A_1//\backslash L) \vee (A_2//\backslash L)$ | $A_1 \vee A_2//[f] = (A_1//[f]) \vee (A_2//[f])$ |
| $\mathbf{T}//\backslash L = \mathbf{T}$ | $\mathbf{T}//[f] = \mathbf{T}$ |
| $\mathbf{F}//\backslash L = \mathbf{F}$ | $\mathbf{F}//[f] = \mathbf{F}$ |

Table 5: Partial evaluation function for restriction and relabeling operator.

$E \triangleright_T F$ a generic state of the process. In order to define a relation of strong bisimulation $\mathcal{R}_T$, we underline that every couple $(\sigma, q)$ of the suppression automata depend on $\delta$. As the process $E$ is a constant, also it can depend on this function. So we denote $E$ with $E^q$. This process has the following definition:

$$E^q = a.E^{q'} \quad \text{iff } \delta(a,q) = q'$$

Now we can define $\mathcal{R}_T$ in the following way:

$$\mathcal{R}_T = \{((\sigma,q), E^q \triangleright_T F) : (\sigma,q) \in \overrightarrow{Act} \times \mathcal{Q}, F \overset{\sigma}{\mapsto}\}$$

Assume that $(\sigma,q) \xrightarrow{a}_T (\sigma',q')$. For the semantic rule of $\triangleright_T$, if $E^q \xrightarrow{a} E^{q'}$ and $F \xrightarrow{a} F'$ perform the action $a$ also $E^q \triangleright_T F \xrightarrow{a} E^{q'} \triangleright_T F'$ and $F' \overset{\sigma'}{\mapsto}$.

Now assume that $E^q \triangleright_T F \xrightarrow{a} E^{q'} \triangleright_T F'$ and $F' \overset{\sigma'}{\mapsto}$. This means that $\delta(a,q) = q'$ and $\sigma = a;\sigma'$. We should prove that exists a $(\sigma,q)'$ s.t. $(\sigma,q) \xrightarrow{a}_T (\sigma,q)'$ and $(E^{q'} \triangleright_T F', (\sigma,q)') \in \mathcal{R}_T$. For the rule T-Step, $(\sigma,q) \xrightarrow{a}_T (\sigma',q')$. So the couple that we are looking for is $(\sigma',q')$. $\square$

**Proposition 2** *Each 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.*

*Proof:* The scheme of the proof and the notation are the same of the previous one. Every couple $(\sigma,q)$ of the suppression automata depend on $\delta$ and $\omega$ hence we denote

$E$ with $E^{q,\omega}$ and define it as follows

$$\begin{aligned} E^{q,\omega} = & \quad a.E^{q',\omega} & \text{iff } \omega(a,q) = + \text{ and } \delta(a,q) = q' \\ = & \quad -a.E^{q',\omega} & \text{iff } \omega(a,q) = - \text{ and } \delta(a,q) = q' \end{aligned}$$

Now we can define $\mathcal{R}_S$ in the following way:

$$\mathcal{R}_S = \{((\sigma,q), E^{q,\omega} \triangleright_S F) : (\sigma,q) \in \overrightarrow{Act} \times \mathcal{Q}, F \overset{\sigma}{\mapsto}\}$$

We have two cases: the first one is similar of proposition 1 in fact, let $((\sigma,q), E^{q,\omega} \triangleright_S F)$ be in $\mathcal{R}_S$ and $(\sigma,q) \xrightarrow{a}_S (\sigma',q')$. We should prove that exists a $(E^{q,\omega} \triangleright_S F)'$ s.t. $E^{q,\omega} \triangleright_S F \xrightarrow{a} (E^{q,\omega} \triangleright_S F)'$ and $((\sigma',q'), (E^{q,\omega} \triangleright_S F)') \in \mathcal{R}_S$. By the first rule of $\triangleright_S$ and by definition of $E^{q,\omega}$, using a similar reason of the proof of proposition 1, we trivially have the thesis. On the other hand, let $(E^{q,\omega} \triangleright_S F, (\sigma,q))$ be in $\mathcal{R}_S$ and $E^{q,\omega} \triangleright_S F \xrightarrow{a} E^{q',\omega} \triangleright_S F'$. We should prove that exists a $(\sigma,q)'$ s.t. $(\sigma,q) \xrightarrow{a}_S (\sigma,q)'$ and $(E^{q',\omega} \triangleright_S F', (\sigma,q)') \in \mathcal{R}_S$. For the rule S-StepA we have that $(\sigma',q')$ is the solution we are looking for. The reasoning is similar to the previous one.

Now, let $((\sigma,q), E^{q,\omega} \triangleright_S F)$ be in $\mathcal{R}_S$ and $(\sigma,q) \xrightarrow{\tau}_S (\sigma',q')$. We should prove that exists a $(E^{q,\omega} \triangleright_S F)'$ s.t. $E^{q,\omega} \triangleright_S F \xrightarrow{\tau} (E^{q,\omega} \triangleright_S F)'$ and $((\sigma',q'), (E^{q,\omega} \triangleright_S F)') \in \mathcal{R}_S$. We have, by the second rule of $\triangleright_S$ and by the definition of $E^{q,\omega}$, that if $E^{q,\omega} \xrightarrow{-a} E^{q',\omega}$ and $F \xrightarrow{a} F'$ then $E^{q,\omega} \triangleright_S F \xrightarrow{\tau} E^{q',\omega} \triangleright_S F'$. We have also $F' \overset{\sigma'}{\mapsto}$.So $((\sigma',q'), E^{q',\omega} \triangleright_S F') \in \mathcal{R}_S$ trivially.

Now assume that $(E^{q,\omega} \triangleright_S F, (\sigma,q))$ be in $\mathcal{R}_S$ and $E^{q,\omega} \triangleright_S F \xrightarrow{\tau} E^{q',\omega} \triangleright_S F'$. Remembering that neither $E$ nor $F$ can perform the action $\tau$, this transection means that $\delta(a,q) = q'$ and $\omega(a,q) = -$. We should prove that exists a $(\sigma,q)'$ s.t. $(\sigma,q) \xrightarrow{\tau}_S (\sigma,q)'$ and $(E^{q',\omega} \triangleright_S F', (\sigma,q)') \in \mathcal{R}_S$. For the rule S-StepS we have that $(\sigma',q')$ is the solution we are looking for. The reasoning is similar to the previous one. □

**Proposition 3** *Each 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.*

*Proof:* The scheme of the proof and the notation are the same of the previous one. Every couple $(\sigma,q)$ of the suppression automata depend on $\delta$ and $\gamma$ hence we denote $E$ with $E^{q,\gamma}$ and define it as follows.

$$\begin{aligned} E^{q,\gamma} = & \quad a.E^{q',\gamma} & \text{iff } \delta(a,q) = q' \\ = & \quad +a.b.E^{q',\gamma} & \text{iff } \gamma(a,q) = (b,q') \end{aligned}$$

Now we can define $\mathcal{R}_I$ in the following way:

$$\mathcal{R}_I = \{((\sigma,q), E^{q,\gamma} \triangleright_I F) : (\sigma,q) \in \overrightarrow{Act} \times \mathcal{Q}, F \overset{\sigma}{\mapsto}\}$$

We have two cases: the first one is similar of proposition 1 in fact, let $((\sigma,q), E^{q,\gamma} \triangleright_I F)$ be in $\mathcal{R}_I$ and $(\sigma,q) \xrightarrow{a}_I (\sigma',q')$. We should prove that exists a $(E^{q,\gamma} \triangleright_I F)'$ s.t. $E^{q,\gamma} \triangleright_I F \xrightarrow{a} (E^{q,\gamma} \triangleright_I F)'$ and $((\sigma',q'), (E^{q,\gamma} \triangleright_I F)') \in \mathcal{R}_I$. By the first rule of $\triangleright_I$ and by definition of $E^{q,\gamma}$,using a similar reasoning of the proof of proposition 1, we trivially have the thesis. On the other hand, let $(E^{q,\gamma} \triangleright_I F, (\sigma,q))$ be in $\mathcal{R}_I$ and

$E^{q,\gamma} \triangleright_I F \xrightarrow{a} E^{q',\gamma} \triangleright_I F'$. We should prove that exists a $(\sigma, q)'$ s.t. $(\sigma, q) \xrightarrow{a}_I (\sigma, q)'$ and $(E^{q',\gamma} \triangleright_I F', (\sigma, q)') \in \mathcal{R}_I$. For the rule I-Step we have that $(\sigma', q')$ is the solution we are looking for. The reasoning is similar to the previous one.

Now let $((\sigma, q), E^{q,\gamma} \triangleright_I F)$ be in $\mathcal{R}_I$ and $(\sigma, q) \xrightarrow{b}_I (\sigma, q')$. We should prove that exists a $(E^{q,\gamma} \triangleright_I F)'$ s.t. $E^{q,\gamma} \triangleright_I F \xrightarrow{b} (E^{q,\gamma} \triangleright_I F)'$ and $((\sigma, q'), (E^{q,\gamma} \triangleright_I F)') \in \mathcal{R}_I$. We have, by second rule of $\triangleright_I$ and by to the definition of $E^{q,\gamma}$, that if $E^{q,\gamma} \not\xrightarrow{a} E^{q',\gamma}$, $E^{q,\gamma} \xrightarrow{+a.b} E^{q',\gamma}$ and $F \xrightarrow{a} F'$ then $E^{q,\gamma} \triangleright_I F \xrightarrow{b} E^{q',\gamma} \triangleright_I F$. So $(E^{q,\gamma} \triangleright_I F)'$ is $E^{q',\gamma} \triangleright_I F$ and $((\sigma, q'), E^{q',\gamma} \triangleright_I F) \in \mathcal{R}_I$ trivially.

Now, let $(E^{q,\gamma} \triangleright_I F, (\sigma, q))$ be in $\mathcal{R}_I$ and $E^{q,\gamma} \triangleright_I F \xrightarrow{b} E^{q',\gamma} \triangleright_I F$. this means that $\sigma = a; \sigma'$ and $\gamma(a, q) = (b, q')$. We should prove that exists a $(\sigma, q)'$ s.t. $(\sigma, q) \xrightarrow{b} (\sigma, q)'$ and $(E^{q',\gamma} \triangleright_I F, (\sigma, q)') \in \mathcal{R}_I$. For the rule I-Ins we have that $(\sigma, q')$ is the solution we are looking for. The reasoning is similar to the previous one. $\square$

**Proposition 4** *Each 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.*

*Proof:* In order to prove this lemma, we give the relation of bisimulation $\mathcal{R}_E$ which exists between edit automata and the controller operator $\triangleright_E$.

$$\begin{aligned} E^{q,\gamma,\omega} &= & a.E^{q',\gamma,\omega} && \text{iff } \delta(a,q) = q' \text{ and } \omega(a,q) = + \\ &= & -a.E^{q',\gamma,\omega} && \text{iff } \delta(a,q) = q' \text{ and } \omega(a,q) = - \\ &= & +a.b.E^{q',\gamma,\omega} && \text{iff } \gamma(a,q) = (b,q') \end{aligned}$$

We define $\mathcal{R}_E$ in the following way:

$$\mathcal{R}_E = \{((\sigma, q), E^{q,\gamma,\omega} \triangleright_E F) : (\sigma, q) \in \overrightarrow{Act} \times \mathcal{Q}, E^{q,\gamma,\omega} \triangleright_E F \in \mathcal{P}, F \overset{\sigma}{\mapsto}\}$$

We have three cases ad their proof following the reasoning made in the proof of lemma 2 and lemma 3. In fact:

- – Let $((\sigma, q), E^{q,\gamma,\omega} \triangleright_E F)$ be in $\mathcal{R}_E$ and $(\sigma, q) \xrightarrow{a}_E (\sigma', q')$. We should prove that exists a $(E^{q,\gamma,\omega} \triangleright_E F)'$ s.t. $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{a}_E (E^{q,\gamma,\omega} \triangleright_E F)'$ and $((\sigma', q'), (E^{q,\gamma,\omega} \triangleright_E F)') \in \mathcal{R}_E$. We have, by the first rule of $\triangleright_E$ and by definition of $E^{q,\gamma,\omega}$, that if $E^{q,\gamma,\omega} \xrightarrow{a}_E E^{q',\gamma,\omega}$ and $F \xrightarrow{a} F'$ then $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{a} E^{q',\gamma,\omega} \triangleright_E F'$. Now $F' \overset{\sigma'}{\mapsto}$. So $(E^{q,\gamma,\omega} \triangleright_E F)'$ is $E^{q',\gamma,\omega} \triangleright_E F'$ and $((\sigma', q'), E^{q',\gamma,\omega} \triangleright_E F') \in \mathcal{R}_E$ trivially.
  - Let $(E^{q,\gamma,\omega} \triangleright_E F, (\sigma, q))$ be in $\mathcal{R}_E$ and $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{a} E^{q',\gamma,\omega} \triangleright_E F'$. We should prove that exists a $(\sigma, q)'$ s.t. $(\sigma, q) \xrightarrow{a} (\sigma, q)'$ and $(E^{q',\gamma,\omega} \triangleright_E F', (\sigma, q)') \in \mathcal{R}_E$. For the rule E-StepA we have that $(\sigma', q')$ is the solution we are looking for. The reasoning is similar to the previous one.

- – Let $((\sigma, q), E^{q,\gamma,\omega} \triangleright_E F)$ be in $\mathcal{R}_E$ and $(\sigma, q) \xrightarrow{\tau}_E (\sigma', q')$. We should prove that exists a $(E^{q,\gamma,\omega} \triangleright_E F)'$ s.t. $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{\tau} (E^{q,\gamma,\omega} \triangleright_E F)'$ and $((\sigma', q'), (E^{q,\gamma,\omega} \triangleright_E F)') \in \mathcal{R}_E$. We have, by second rule of $\triangleright_E$ and by the definition of $E^{q,\gamma,\omega}$, that if $E^{q,\gamma,\omega} \xrightarrow{-a} E^{q',\gamma,\omega}$ and $F \xrightarrow{a} F'$ then $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{\tau} E^{q',\gamma,\omega} \triangleright_E F'$. Now $F' \overset{\sigma'}{\mapsto}$. So $(E^{q,\gamma,\omega} \triangleright_E F)'$ is $E^{q',\gamma,\omega} \triangleright_E F'$ and $((\sigma', q'), E^{q',\gamma,\omega} \triangleright_E F') \in \mathcal{R}_E$ trivially.

  - Let $(E^{q,\gamma,\omega} \triangleright_E F, (\sigma, q))$ be in $\mathcal{R}_E$ and $E^{q,\omega} \triangleright_E F \xrightarrow{\tau} E^{q',\gamma,\omega} \triangleright_E F'$. We should prove that exists a $(\sigma, q)'$ s.t. $(\sigma, q) \xrightarrow{\tau}_e (\sigma, q)'$ and $(E^{q,\gamma,\omega} \triangleright_E F', (\sigma, q)') \in \mathcal{R}_E$ For the rule E-StepS we have that $(\sigma', q')$ is the solution we are looking for. The reasoning is similar to the previous one.

- - Let $((\sigma, q), E^{q,\gamma,\omega} \triangleright_E F)$ be in $\mathcal{R}_E$ and $(\sigma, q) \xrightarrow{b}_E (\sigma, q')$. We should prove that exists a $(E^{q,\gamma,\omega} \triangleright_E F)'$ s.t. $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{b} (E^{q,\gamma,\omega} \triangleright_E F)'$ and $((\sigma, q'), (E^{q,\gamma,\omega} \triangleright_E F)') \in \mathcal{R}_E$. We have, by third rule of $\triangleright_E$ and by the definition of $E^{q,\gamma,\omega}$ that if $E^{q,\gamma,\omega} \not\xrightarrow{a} E^{q',\gamma,\omega}$, $E^{q,\gamma,\omega} \xrightarrow{+a.b} E^{q',\gamma,\omega}$ and $F \xrightarrow{a} F'$ then $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{b} E^{q',\gamma,\omega} \triangleright_E F$. So $(E^{q,\gamma,\omega} \triangleright_E F)'$ is $E^{q',\gamma,\omega} \triangleright_E F$ and $((\sigma, q'), E^{q',\gamma,\omega} \triangleright_E F) \in \mathcal{R}_E$ trivially.

  - Let $(E^{q,\gamma,\omega} \triangleright_E F, (\sigma, q))$ be in $\mathcal{R}_E$ and $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{b} E^{q',\gamma,\omega} \triangleright_E F$. We should prove that exists a $(\sigma, q)'$ s.t. $(\sigma, q) \xrightarrow{b} (\sigma, q)'$ and $(E^{q',\gamma} \triangleright_E F, (\sigma, q)') \in \mathcal{R}_E$. For the rule E-Ins we have that $(\sigma, q')$ is the solution we are looking for. The reasoning is similar to the previous one.

□

**Proposition 5** *For every $\mathcal{K} \in \{$truncation, suppression, insertion, edit$\}$ the following relation holds*

$$Y \triangleright_{\mathcal{K}} X \preceq Y[f_{\mathcal{K}}]$$

*where $f_{\mathcal{K}}$ is a relabeling function definition of which depend on $\mathcal{K}$.*

In order to prove this proposition we prove the following four lemmas. The proof of the proposition comes from these.

**Lemma 4** *The following relation holds*

$$Y \triangleright_T X \preceq Y[f_T] \tag{5}$$

*where $f_T$ is the identity function.*

*Proof:* We prove that the following relation is a weak simulation.

$$\mathcal{S}_T = \{(E \triangleright_T F, E[f_T]) | E, F \in \mathcal{E}\}$$

Note that being $f_T$ the identity function we could omit it without loss of generality.

Assume that $E \triangleright_T F \xrightarrow{a} E' \triangleright_T F'$ with the additional hypothesis that $F \xrightarrow{a} F'$ then, by the rule of $\triangleright_T$ we have that $E \overset{a}{\Rightarrow} E'$ and, obviously, $(E' \triangleright_T F', E') \in \mathcal{S}_T$. □

**Lemma 5** *The following relation holds*

$$Y \triangleright_S X \preceq Y[f_S] \tag{6}$$

*where*

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

*Proof:* We prove that the following relation is a weak simulation.

$$\mathcal{S}_S = \{(E \triangleright_S F, E[f_S]) | E, F \in \mathcal{E}\}$$

There are two possible cases: the first one is when $E \triangleright_S F$ performs the action $a$. The proof of this case is the same of the proof of lemma 4. If $E \triangleright_S F \xrightarrow{\tau} E' \triangleright_S F'$ means that $E \xrightarrow{-a} E'$ and $F$ perform an action $a$ that $E$ should not perform. Applying the relabeling function $f_S$ to $E$ we obtain $E_1 = E[f_S]$ s.t. $E_1 \overset{\tau}{\Longrightarrow} E_1'$. where $E_1'$ is $E'[f_S]$. Hence $(E' \triangleright_S F', E_1') \in \mathcal{S}_S$. □

**Lemma 6** *The following relation holds*

$$Y \triangleright_I X \preceq Y[f_I] \tag{7}$$

*where*

$$f_I(a) = \begin{cases} a & \textit{if } a \in Act \\ \tau & \textit{if } a = +a \end{cases}$$

*Proof:* We prove that the following relation is a weak simulation.

$$\mathcal{S}_I = \{(E \triangleright_I F, E[f_I]) | E, F \in \mathcal{E}\}$$

There are two possible cases: the first one is when $E \triangleright_I F$ performs the action $a$. The proof of this case is the same of the proof of lemma 4. If $E \triangleright_I F \xrightarrow{b} E' \triangleright_I F$ means that $E \xrightarrow{+a.b} E'$ and $F$ perform an action $a$ that $E$ should not perform in order to go in the state $E'$. Applying the relabeling function $f_I$ to $E$ we obtain $E_1 = E[f_I]$ s.t. $E_1 \overset{b}{\Longrightarrow} E_1'$. where $E_1'$ is $E'[f_I]$. Hence $(E' \triangleright_I F', E'1) \in \mathcal{S}_I$. □

**Lemma 7** *The following relation holds*

$$Y \triangleright_E X \preceq Y[f_E] \tag{8}$$

*where*

$$f_E(a) = \begin{cases} a & \textit{if } a \in Act \\ \tau & \textit{if } a \in \{-a, +a\} \end{cases}$$

*Proof:* We prove that the following relation is a weak simulation.

$$\mathcal{S}_E = \{(E \triangleright_E F, E[f_E]) | E, F \in \mathcal{E}\}$$

There are three possible cases: the first one is when $E \triangleright_E F$ performs the action $a$. The proof of this case is the same of the proof of lemma 4. the other two case is the following:

- $E \triangleright_E F \xrightarrow{\tau} E' \triangleright_E F'$ we want to find a $E'[f_E]$ s.t. $E[f_E] \xrightarrow{\tau} E[f_E]'$. Referring to the second rule of the edit automata we see that $E \triangleright_E F \xrightarrow{\tau} E' \triangleright_E F'$ when $E \xrightarrow{-a} E'$. Through the relabeling function $f_E$ we have $E[f_E] \xrightarrow{\tau} E'[f_E]$ and $(E' \triangleright_E F', E'[f_E]) \in \mathcal{S}_E$.

- $E \triangleright_E F \xrightarrow{b} E' \triangleright_E F$ we want to find a $E'[f_E]$ s.t. $E[f_E] \overset{b}{\Longrightarrow} E[f_E]'$. Referring to the last rule of edit automata we see that $E \triangleright_E F \xrightarrow{b} E' \triangleright_E F$ when $E \xrightarrow{+a.b} E'$. Through the relabeling function $f_E$ we have $E[f_E] \overset{b}{\Longrightarrow} E'[f_E]$ and $(E' \triangleright_E F, E'[f_E]) \in \mathcal{S}_E$

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

*Proof*: A translation from equational $\mu$-calculus to modal $\mu$-calculus is possible [1]. So first of all we consider the modal formula associated with the given formula $\phi$ then the proof may be divided in two part. Former we prove the proposition holds for the formulae of modal $\mu$-calculus without recursion operator, latter we extended the results also to $\mu X.\phi$ and $\nu X.\phi$.

The first part is very similar to the proof proposed by Stirling in [19] that is made by induction on the structure of the formula $\phi$. The base case is clear. For the inductive step first suppose $\phi = \phi_1 \wedge \phi_2$ and that the result holds for the components $\phi_1$ and $\phi_2$. By the definition of satisfaction relation $E \models \phi$ iff $E \models \phi_1$ and $E \models \phi_2$. By inductive hypothesis $F \models \phi_1$ and $F \models \phi_2$ then $F \models \phi$. A similar argument justifies the case $\phi = \phi_1 \vee \phi_2$. Next suppose $\phi = [a]\phi_1$ and $E \models \phi$. Therefore for any $E'$ s.t. $E \overset{a}{\Rightarrow} E'$ it follows that $E' \models \phi_1$. Let $F \overset{a}{\rightarrow} F'$ we know that for some $E'$ there is the transition $E \overset{a}{\Rightarrow} E'$ and $F' \preceq E'$, so by inductive hypothesis $F' \models \phi_1$ and so $F \models \phi$. Now we have to prove that if $\phi = \mu X.\phi_1$ or $\phi = \nu X.\phi_1$ the proposition holds. Referring to the definition of minimum and maximum fixed point we can consider these as inductive limit (the union) of formulae like $\mu X^\alpha.\phi_1$, where $\mu X^0.\phi_1 = \mathbf{F}$ and $\mu X^{\alpha+1}.\phi_1 = \phi_1[\mu X^\alpha.\phi_1/X]$, and $\nu X^\alpha.\phi_1$ where $\nu X^0.\phi_1 = \mathbf{T}$ and $\nu X^{\alpha+1}.\phi_1 = \phi_1[\nu X^\alpha.\phi_1/X]$. In this way $E \models \mu X.\phi_1$ iff $E \models \mu X^\alpha.\phi_1$ for some $\alpha$ iff $E \models \bigvee_\alpha(\mu X^\alpha.\phi_1)$ and $E \models \nu X.\phi_1$ iff $E \models \nu X^\alpha.\phi_1$ for all $\alpha$ iff $E \models \bigwedge_\alpha(\nu X^\alpha.\phi_1)$. In the former case we have a sequence of disjunction and in the latter we have a sequence of conjunction. We can apply again the argument of the first part of the proof. $\square$

**Proposition 7**: *Given a formula $\phi \in \forall_\wedge \mu C$, a maximal deterministic model $E$ of this formula exists.*

In order to prove this proposition we introduce the following notions.

## B.1 Canonical structure

The vocabulary of the $\mu$-calculus is extended by a countable set *DCons* of fresh symbols that will be referred to as *definition constant* and usually denoted $U$, $V$, $\cdots$ (see [21]). These new symbols are now allowed to appear positively in formulae, like propositional variables. A *definition list* is a finite sequence of equations: $\mathcal{D} = ((U_1 = \sigma_1 X.\alpha_1(X)), \cdots, (U_n = \sigma_n X.\alpha_n(X))$ where $U_1, \cdots, U_n \in DCons$ and $\sigma_i X.\alpha_i(X)$ is a formula such that all definition constants appearing in $\alpha_i$ are among $U_1, \cdots, U_{i-1}$. We assume that $U_i \neq U_j$ and $\alpha_i \neq \alpha_j$ for $i \neq j$. If $i < j$ then $U_i$ is said to be older than $U_j$.

A *tableau sequent* is a pair $(\Gamma, \mathcal{D})$ where $\mathcal{D}$ is a definition list and $\Gamma$ is a finite set of formulae such that the only constants that occur in them are those from $\mathcal{D}$. We will denote $(\Gamma, \mathcal{D})$ by $\Gamma \vdash_{\mathcal{D}}$.

A *tableau axiom* is a sequent $\Gamma \vdash_{\mathcal{D}}$ such that some formula and its negation occurs in $\Gamma$.

Below we present the set of rules for constructing *tableau*. Let $\mathcal{S}$ be the following set of tableau rules:

$$\begin{array}{l}
\text{(and)}\ \dfrac{\alpha \wedge \beta, \Gamma \vdash_{\mathcal{D}}}{\alpha, \beta, \Gamma \vdash_{\mathcal{D}}} \\
\text{(or)}\ \dfrac{\alpha \vee \beta, \Gamma \vdash_{\mathcal{D}}}{\alpha, \Gamma \vdash_{\mathcal{D}} \quad \beta, \Gamma \vdash_{\mathcal{D}}} \\
\text{(cons)}\ \dfrac{U, \Gamma \vdash_{\mathcal{D}}}{\alpha U, \Gamma \vdash_{\mathcal{D}}}\ \text{whenever } (U = \sigma X.\alpha(X)) \in \mathcal{D} \\
(\mu)\ \dfrac{\mu X.\alpha(X), \Gamma \vdash_{\mathcal{D}}}{U, \Gamma \vdash_{\mathcal{D}}}\ \text{whenever } (U = \mu X.\alpha(X)) \in \mathcal{D} \\
(\nu)\ \dfrac{\nu X.\alpha(X), \Gamma \vdash_{\mathcal{D}}}{U, \Gamma \vdash_{\mathcal{D}}}\ \text{whenever } (U = \nu X.\alpha(X)) \in \mathcal{D} \\
\text{(all } \langle\rangle)\ \dfrac{\Gamma \vdash_{\mathcal{D}}}{\{\alpha, \{\beta : [a]\beta \in \Gamma\} \vdash_{\mathcal{D}} \quad : \langle a \rangle \alpha \in \Gamma\}}
\end{array}$$

where in the last rule each formula in $\Gamma$ is a propositional constant, a variable, a negation of one of them or a formula of the form $\langle b \rangle \beta$ or $[b]\beta$ for some action $b$ and a formula $\beta$.

Observe that each rule, except *(or)* or *(all $\langle\rangle$)*, has exactly one premise.

The system $\mathcal{S}_{mod}$ is obtained from $\mathcal{S}$ by replacing the rule *(or)* by two rules $(or_{left})$ and $(or_{right})$ defined in the obvious way.

The system $\mathcal{S}_{ref}$ is obtained from $\mathcal{S}$ by replacing the rule $(all\langle\rangle)$ by the rule $(\langle\rangle)\quad \dfrac{\langle a \rangle \alpha, \Gamma \vdash_{\mathcal{D}}}{\alpha, \{\beta : [a]\beta \in \Gamma\} \vdash_{\mathcal{D}}}$ with the same restrictions on formulae in $\Gamma$ as in the case of $(all\langle\rangle)$ rule.

**Definition 8** *Given a positive guarded formula $\phi$, a* tableau *for $\phi$ is any labeled tree $\langle K, L \rangle$, where $K$ is a tree and $L$ a labeling function, such that*

1. *the root of $K$ is labeled with $\phi \vdash_{\mathcal{D}}$ where $\mathcal{D}$ is the definition list of $\phi$;*
2. *if $L$ is a tableau axiom then $n$ is a leaf of $K$;*
3. *if $L(n)$ is not an axiom then the sons of $n$ in $K$ are created and labeled according to the rules of the system $\mathcal{S}$.*

A *quasi-model* of $\phi$ is defined in a similar way to tableau, except the system $\mathcal{S}_{mod}$ is used instead of $\mathcal{S}$ and we impose the additional requirement that no leaf is labeled by a tableau axiom. A *quasi-refutation* of $\phi$ is defined in a similar way to tableau, except the system $\mathcal{S}_{ref}$ is used instead of $\mathcal{S}$ and we impose the additional requirement that every leaf is labeled by a tableau axiom.

Let $\mathcal{P} = (v_1, v_2, \cdots)$ be a path in the tree $K$. A *trace* $\mathcal{T}r$ on the path $\mathcal{P}$ is any sequence of formulas $\{\alpha_i\}_{i \in I}$ such that $\alpha_i \in L(v_i)$ and $\alpha_{i+1}$ is either $\alpha_i$, if formula $\alpha_i$ was not reduced by the rule applied in $v_i$, or otherwise $\alpha_{i+1}$ is one of the formulae obtained by applying the rule to $\alpha_i$.

A constant $U$ *regenerates* on the trace $\mathcal{T}r$ if for some $i$, $a_i = U$ and $a_{i+1} = \alpha(U)$ where $(U = \sigma X.\alpha(X)) \in \mathcal{D}$. The trace $\mathcal{T}r$ is called a $\nu$-trace iff it is finite and does not end with a tableau axiom, or if the oldest constant in the definition list $\mathcal{D}$ which is regenerated infinitely often on $\mathcal{T}r$ is a $\nu$-constant. Otherwise the trace is called a $\mu$-trace.

**Definition 9** *A quasi model $\mathcal{PM}$ is called* pre-model *iff any trace on any path of $\mathcal{PM}$ is a $\nu$-trace.*

*A quasi-refutation of $\phi$ is called a* refutation *of $\phi$ iff on every path of there exists a $\mu$-trace.*

**Definition 10** *Given a pre-model $\mathcal{PM}$, the* canonical structure *for $\mathcal{PM}$ is a structure $\mathcal{M} = \langle S^{\mathcal{M}}, R^{\mathcal{M}}, \rho^{\mathcal{M}} \rangle$ such that*

1. *$S^{\mathcal{M}}$ is the set of all nodes of $\mathcal{PM}$ which are either leaves or to which $(all\langle\rangle)$ rule was applied. For any node $n$ of $\mathcal{PM}$ we will denote by $s_n$ the closest descendant of $n$ belonging to $S^{\mathcal{M}}$.*
2. *$(s, s') \in R^{\mathcal{M}}(a)$ iff there is a son $n$ of $s$ with $s_n = s'$, such that $L(n)$ was obtained from $L(s)$ by reducing a formula of the form $\langle a \rangle \alpha$.*
3. *$\rho^{\mathcal{M}}(p) = \{s : p$ occurs in the sequent $L(s)\}$.*

In the following we assume pre-models (and so canonical models) that are built using quasi-models where the $(or_{left})$ is applied only if the $(or_{right})$ fails to provide a pre-model. With this assumption, since we will apply the canonical model only to one kind of formula with disjunction, we may control which branch will be followed and so the kind of canonical model generated.

**Proposition 9 ([21])** *If there exists a pre-model $\mathcal{PM}$ for a positive guarded sentence $\phi$ then $\phi$ is satisfiable in the canonical structure for $\mathcal{PM}$.*

## B.2 Proof of proposition 7

**Lemma 8** *Let $\phi \in \forall_\wedge \mu C$ and $\psi = X$ where $X =_\nu \bigwedge_{\alpha \in Act}([\alpha]\mathbf{F} \vee (\langle \alpha \rangle X \wedge [\alpha]X))$. If $\phi$ is satisfiable then $\phi \wedge \psi$ is satisfiable.*

*Proof*: The formula $X =_\nu \bigwedge_{\alpha \in Act}([\alpha]\mathbf{F} \vee (\langle \alpha \rangle X \wedge [\alpha]X))$ or its equivalent formulation in modal $\mu-$calculus $\nu X. \bigwedge_{\alpha \in Act}([\alpha]\mathbf{F} \vee (\langle \alpha \rangle X \wedge [\alpha]X))$ holds in every state (i.e. it is a tautology) and so if $E$ is a model of $\phi$ then $E$ is also a model for $\psi$. To prove that $\psi$ is a tautology one can build a refutation for its negation using the $\mathcal{S}_{ref}$ tableaux. □

**Lemma 9** *Let $E' \models \phi$ with $\phi \in \forall_\wedge \mu C$. Then the canonical model $E$ of $\phi \wedge \psi$, is such that $E' \preceq E$.*

*Proof*: We define the following relation:

$$\mathcal{R} = \{(E', E) | \exists \phi, E' \models \phi \in \forall_\wedge \mu C \text{ and } E \models \phi \wedge \psi$$

$$\text{and } E \text{ is the canonical structure for } \phi \wedge \psi\}$$

and prove that $\mathcal{R}$ is a simulation.

Suppose that $E' \xrightarrow{\alpha} E'_1$, $E' \models \phi$ and $E'_1 \models \phi'$ for some $\phi' \in \forall_{\wedge}\mu C$. As a matter of fact, due to the specific assumptions we have on $\phi$, we may rewrite it in an equivalent form $\phi^*$ as $\wedge_{\alpha \in Act}[\alpha]\phi_\alpha$. Thus $\phi'$ would be equivalent to $\phi_\alpha$ and $\phi_\alpha$ is not equivalent to **F** (since it has a model $E'$).

In the canonical model (that must exist since $\phi \wedge \psi$ is satisfiable) the possibility is to choose the $(or_{right})$ and so $E$ will do an $\alpha$ action reaching another state that is a model for $\psi$ and is also a model for $\phi_\alpha$, see rule **all** in the tableaux construction. As a matter of fact the initial tableaux contruction exactly puts $\phi$ in the desired format before applying the reduction. $\square$

*Proof of proposition 7*: It is necessary to prove that such $E$ is a model for $\phi$, that it is a maximal model and that it is a deterministic process. By Lemma A.1 it follows that $E$ is a model for $\phi$. Being $E$ the canonical structure, it is easy to note that it is deterministic because it performs only one action $\langle \_ \rangle$ and so every rule that permits to construct it has only a premise (see rule **all**). The maximality follows from Lemma A.2.

$\square$