Consiglio Nazionale delle Ricerche

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

F. Martinelli, I. Matteucci

IIT TR-07/2005

Technical report

Marzo 2005

Istituto di Informatica e Telematica

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

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

Abstract

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

1 Overview

Many approaches for the analysis of security properties have been successfully developed in the last two decades. One is based on the idea that potential attackers should be analyzed as if they were un-specified components of a system; thus reducing security analysis to the analysis of *open* systems [7, 8, 10].

More recently there has been also interest on developing techniques to study how to enforce security properties. One notable example is the security automata in [14] and some extensions proposed in [5].

The paradigm of analysis of security as analysis of open systems has been extended to cope with security protocols [10], fault tolerance [4] and recently access control based on trust management [11]. In this paper we enrich this theory with a method for (automatically) enforcing security properties.

Basically, we define a set of process algebra operators that act as programable controllers of a component that must be managed in order to guarantee that the overall system satisfies a given security policy. Also, we developed a technique to automatically synthesize the appropriate controllers. This represent a significant contribution w.r.t. to the previous work in [14, 5] where this issue was not addressed. The synthesis is based on a satisfiability procedure for the mu-calculus.

Moreover, under certain hypothesis on the observation power of the enforcing controllers, we are able to enforce some non-interference properties (for finite-state systems) that were not intentionally addressed in [14], due to the specific assumptions they had on the enforcing mechanisms.

Our logical approach is also able to cope with composition problems, that have been considered as an interesting issue in [2], however, not addressed in the journal version [5].

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

*Work partially supported by CNR project "Trusted e-services for dynamic coalitions" and by a CREATE-NET grant for the project "Quality of Protection (QoP)".

2 Background

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

2.1 A language for describing concurrent and distributed systems

We now describe the syntax for the *Security process algebra* (SPA) [3] used to describe concurrent and distributed systems that is derived from CCS process algebra of R. Milner [12]:

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

where α is an action in Act, $L \subseteq \mathcal{L}$ and Z is a process constant that must be associated with a definition $Z \doteq E$. As usual, we assume that constants are *guarded* [12], i.e. they must be in the scope of some prefix operator $\alpha.E'$. The set of SPA *processes*, i.e., of terms with guarded constants, is denoted with $\mathcal{E}$, ranged over by $E, F, P, Q \ldots$. We will often use some usual syntactic simplifications, e.g., omission of trailing $\mathbf{0}$'s as well as omission of brackets on restriction on a single action. We use $Sort(E)$ to denote the set of actions that occurs in the term E.

We give an informal overview of SPA operators:

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

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

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

In the appendix the interested reader may find the formal definition of the semantics.

2.2 Strong and weak bisimulations

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

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

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

Now, we can define *strong bisimulation*:

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

We define another kind of bisimulation: *weak bisimulation*. This relation is used when there is the necessity to understand if systems with different internal structure, and hence different internal behavior, have the same external behavior and thus may be considered observationally equivalent.

Before, we give the notion of *observational relations*: $E \stackrel{\tau}{\Rightarrow} E'$ (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 $E \stackrel{\tau}{\Rightarrow}\stackrel{a}{\rightarrow}\stackrel{\tau}{\Rightarrow} E'$.

We, now, give the definition of weak simulation. This definition is very similar to the definition of strong simulation, but reference to experiments $\stackrel{e}{\Rightarrow}$ instead of arbitrary actions $\stackrel{a}{\rightarrow}$.

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

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

We, now, give a definition similar to the 2.3 for the weak bisimulation.

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

Every strong simulation is also a weak one (see [12])

2.3 Equational μ-calculus

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

Equational μ-calculus is based on fixpoint equations that substitute recursion operators, so it permits to define recursively the properties of a given systems.

We use the equational μ-calculus instead of modal μ-calculus because the first one is very suitable for partial model checking (see [1]). Let a be in Act and X be a variable ranging over a finite set of variables $Vars$.

We give the grammar:

$$A ::= X \mid \mathbf{T} \mid \mathbf{F} \mid X_1 \wedge X_2 \mid X_1 \vee X_2 \mid \langle a \rangle X \mid [a]X$$

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

$X =_\nu A$ is a minimal fixpoint equation, where A is an assertion, i.e. a simple modal formula without recursion operator, and $X =_\mu A$ is a maximal fixpoint equation. Roughly, the semantic $[\![D]\!]$ of the list of equations D is the solution of the system of equations corresponding to D. According to this notation, $[\![D]\!](X)$ is the value of the variable X, and we write $E \models D \downarrow X$ as a notation for $E \in [\![D]\!](X)$.

The following result can be proved by putting together standard results for decision procedures for μ-calculus (see [15]).

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

2.4 Characteristic formulae

A characteristic formula is a formula in equational μ-calculus that completely characterizes the behavior of a (state in a) state-transition graph modulo a chosen notion of behavioral relation. It is possible to define the notion of characteristic formula for a given finite state process E w.r.t. weak bisimulation as follows (see [13]).

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

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

where $\langle\langle a \rangle\rangle$ of the modality $\langle a \rangle$ which can be introduce as abbreviation (see [13]):

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

$$\langle\langle a \rangle\rangle \phi \stackrel{def}{=} \langle\langle \epsilon \rangle\rangle \langle a \rangle \langle\langle \epsilon \rangle\rangle \phi$$

We have the following lemma that characterize the power of these formulas.

Lemma 2.1 *Let E_1 and E_2 be two different finite-state processes. If ϕ_{E_2} is characteristic for E_2 then:*

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

2.5 Partial model checking

We use partial model checking technique that was developed for compositional analysis of concurrent systems (processes) (see [1]). The idea is the following: we consider a system which is the parallel composition of two process, $E \| F$. We want to study if this system verifies a given formula ϕ or not. In formula:

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

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

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

Lemma 2.2 *Given a process $E \| F$ and a formula ϕ we have:*

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

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

In the appendix there is the partial evaluation function for parallel operator.

2.6 A logical approach for specifying and analyzing information flow properties

Information flow is a main topic in the theoretical study of computer security. We can find several formal definitions in the literature (see [6]). To describe this problem, we can consider two users, *High* and *Low* interacting with the same computer system. We ask if there is any flow of information from *High* to *Low*. The central property is the itshape Non Deducibility on composition (*NDC*, see [3]): the low level users cannot infer the behavior of the high level user from the system because for the the low level users the system is always the same. This idea can be represented as follow:

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

We study this property in term of *SPA* parallel composition operator and *bisimulation* equivalence.

We denote with $BNDC$ a security property called *Bisimulation Non Deducibility on Compositions*.

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

By using the characteristic formulae of a process we may express this information flow property in a logical way.

$$E \in BNDC \text{ iff } \forall \Pi \in S : (E \| \Pi) \backslash H \models \phi_{\approx, E \backslash H}$$

By using partial model checking we have a method for reducing the verification of the previous property to a validity checking problem in μ-calculus (see [7]). (We must remember that this is true only if we consider finite-state processes.)

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

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

3 Our approach for enforcing security properties

We wish to provide a framework where we are able to enforce specific security properties. We assume to have a system S that will cooperate with another high component X $(S \| X)$, possibly not known a priori. We would like that the visible low behavior of this system is always the same regardless of the high component behavior.

By using the theory in [8], one can study whether a potential enemy could exists. In particular, we know the necessary and sufficient conditions that an enemy should satisfy in order to alter the visible behavior of the system.

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

We consider three levels of observability on the process X:

1. we cannot inspect its code; if X performs an action we may detect and intercept it;
2. we know which are the possible next steps and whether the X cannot perform a given action;
3. we are able to access its whole code.

Depending on these three scenarios, different techniques may be applied to enforce security properties.

In particular, in this last case, we can applied partial model checking to the system $(S \| X)$, so we obtain $X \models \phi'$ from $S \| X \models \phi$ where $\phi' = \phi_{//S}$. In order to investigate if X satisfies the formula ϕ' or not, we can applied model checking technique. We can notice that, in this case, a controller is not necessary. Clearly this technique works as far as decidability issues are solved.

In the other two cases, the introduction of a controller operator helps us to guarantee a correct behavior of the entire system. The remaining two cases are similar but not equal. In fact, in the second case we can analyze the set of possible next step in order to understand if the following action that X is going to perform is correct or not and eventually we can modify the behavior of the system and force it to be correct. Again, the inspection of the next step could not be a decidable problem. We assume it is in the following.

We assume to have an operator, say $Y \triangleright^* X$, that can permit to control the behavior of the component X, given the behavior of a control program Y. Note that differently from other approaches the control target and the controller are expressed in a similar formalism.

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

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E \triangleright' F \xrightarrow{a} E' \triangleright' F'} \tag{3}$$

$$\frac{E \xrightarrow{a} E'}{E \triangleright' F \xrightarrow{a} E' \triangleright' F} \tag{4}$$

This controller operator can be helpful in the first scenario that we have illustrated before. In fact with this operator we force the system to make always the right action also if we don't know what action X is going to perform. A more clever controller may be defined if we are permitted to observe the possible next steps of F.

3.1 Enforcing security properties

Let S be a system, and let X one of its components (e.g., a downloaded mobile agent). We say that the new system $S \| X$ enjoys a security property expressed iff for every behavior of the component X, the behavior of the system S enjoys that security property:

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

where $H = Sort(X)$ and ϕ expresses the correct behavior of the system. We want to prove that exists an agent Y such that:

$$\forall X (S \| Y \triangleright^* X) \backslash H \models \phi \tag{6}$$

After one step of partial model checking, we obtain the equivalent property:

$$\exists Y \, \forall X \, (Y \triangleright^* X) \models \phi' \tag{7}$$

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

While the equation 7 should be the property to manage, it might not be easy. However, we note that if the controller operator satisfies the following additional property

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

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

then the property 7 is equivalent to:

$$\exists Y \; Y \models \phi' \tag{8}$$

As a matter of fact, the previous assumption permits us to conclude that $Y \triangleright^* X$ and Y are strongly equivalent on so they satisfy the same formulas. The formulation 8 is easier to be managed.

We note that the operator $\triangleright'$ defined in the Example 3.1 enjoys Assumption 3.1.

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

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

3.2 Automated synthesis of controllers

We may exploit the satisfiability procedure of theorem 2.1 for achieving the automated synthesis of suitable controllers that enforce specific security properties, i.e. find a suitable Y for 8. Unfortunately, the satisfiability procedure has a complexity that is, in the worst case, exponential in the size of the formula. Eventually, this provide us with an automated mechanism to enforce security properties.

3.3 Composition of properties

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

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

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

$$\forall X(S \| X) \backslash H \models \bigwedge_{i=1,\cdots,n} \phi_i \tag{9}$$

If we assume $\bigwedge_{i=1,\cdots n} \phi_i = \phi$, we have the same situation that we have described by the formula 7.

4 Other controllers

We can define other controller operators as follows.

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

$$\frac{E \xrightarrow{a} E' \; F \xrightarrow{a} F'}{E \triangleright'' F \xrightarrow{a} E' \triangleright'' F'} \tag{10}$$

$$\frac{E \xrightarrow{a} E' \; F \overset{a}{\not\rightarrow} F'}{E \triangleright'' F \xrightarrow{a} E' \triangleright'' F} \tag{11}$$

This controller is the most complete: if F does not have a correct behavior, the process E correct the action of F, so the system maintains a honest behavior, else F make its action.

The following result holds.

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

Another interesting operator is described by the following rule:

$$\frac{E \xrightarrow{a} E' \; F \xrightarrow{a} F'}{E \triangleright''' F \xrightarrow{a} E' \triangleright''' F'} \tag{12}$$

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

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

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

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

As a matter of fact, for this operator we can ensure that the systems is secure only w.r.t. security properties that are safety properties. Such properties are preserved under weak simulation (e.g. see [4]). Thus, we can enforce safety properties through this controller.

4.1 Implementability issues for our controllers

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

For the first controller operator, $\triangleright'$, we can note that this operator may in any moment neglect the agent behavior because the behavior of the system may simply follow the behavior of the controller process. In particular, the controller may always choose to perform its correct action, rather than waiting for an action by the target.

Thus, it would be easily implementable in all the three scenarios.

The operator $\triangleright''$ cannot be implemented in the scenario 1: if we cannot decide a priori which are possible next steps that the external agent is able and not able to perform and so we cannot implement the second rule (11). In the scenario 2) such an operator would be implementable. It would be also possible in the scenario 2) to give priority to the first rule in order to allow always the correct actions of the target. Thus, controller $\triangleright''$ would be our favorite, if we could consider scenario 2).

The last controller operator, as we will say after, can be implemented in any scenarios. As a matter of fact, it coincides with the monitors defined in [14]. Below we discuss more deeply this point.

5 A simple example

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6 Discussion on related work

In [9], we presented preliminary work based on different techniques for automatically synthesizing systems enjoying a very strong security property, i.e. SBSNNI (e.g., see [3]). That work did not deal with controllers.

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

In [14], Schneider deals with enforceable security properties in a systematic way. He discusses whether a given property is enforceable and at what cost. To study those questions, Schneider uses the class of enforceable mechanisms (EM) that work by monitoring execution steps of some system, herein called the *target*, and terminating the target's execution if it is about to violate the security property being enforced. The author asserts there isn't any EM (Execution Monitoring) that can enforce information flow because it can't be formalized like a safety property. The security automata defined in [14] have the follow behavior:

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

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

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

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

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

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

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

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

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

Also in [5, 2] there is the idea that information flow can not be forced by an automaton. The reasoning is the same that is given by Schneider in [14]: information flow can only be specified as a condition on the set of possible execution of a program instead a property is defined exclusively in term of individual execution and may not specify a relationship between different executions of the program.

In both of these articles, many types of automata are illustrated. All of them are in the scenario 1). The automata waits for an action of the target. In particular, in [5] there are four different automata:

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

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

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

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

Now we should give a definition of these automata by our formalism. Since that truncation automata is the same automata is described in [14], we already define a controller operator which have the same outputs.

Now we should prove the same properties for the other automata. For this reason, we introduce the following controller operators: $\triangleright_S$, $\triangleright_I$ and $\triangleright_E$.

To be able to compare these automata with controllers definable in our framework, it is crucial to have a rigorous definition of semantic rules that describe the behavior of each operator:

$\triangleright_S$

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

$$\frac{E \overset{a}{\not\rightarrow} E' \quad E \xrightarrow{-a} E' \quad F \xrightarrow{a} F'}{E \triangleright_S F \xrightarrow{\tau} E' \triangleright_S F'} \tag{14}$$

where $-a$ is an action not in Act, so it doesn't admit a complementary action, which is made by the process E in order to verify that the process F performs the action a and τ is an internal action that permit to suppress the action a which is made by F.

$\triangleright_I$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E \triangleright_I F \xrightarrow{a} E' \triangleright_I F'} \tag{15}$$

$$\frac{E \overset{a}{\not\rightarrow} E'' \quad E \xrightarrow{+a.b_1 \cdots b_n} E' \quad F \xrightarrow{a} F'}{E \triangleright_I F \xrightarrow{b_1 \cdots b_n} E' \triangleright_I F} \tag{16}$$

where $+a$ is an action not in Act, so it doesn't admit a complementary action, which is perform by the process E in order to verify if the process F is going to perform the action a. In this case the controller process insert a sequence of action $b_1 \cdots b_n$ and the entire system performs this sequence of actions[1].

$\triangleright_E$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{E \triangleright_E F \xrightarrow{a} E' \triangleright_E F'} \tag{17}$$

$$\frac{E \overset{a}{\not\rightarrow} E' \quad E \xrightarrow{-a} E' \quad F \xrightarrow{a} F'}{E \triangleright_E F \xrightarrow{\tau} E' \triangleright_E F'} \tag{18}$$

$$\frac{E \overset{a}{\not\rightarrow} E'' \quad E \xrightarrow{+a.b_1 \cdots b_n} E' \quad F \xrightarrow{a} F'}{E \triangleright_E F \xrightarrow{b_1 \cdots b_n} E' \triangleright_E F} \tag{19}$$

These three rules derive from the union of the two previous controller operators.

We should prove the following general result:

Proposition 6.1 *Let $\mathcal{K}$ be a set of different kind of automata, $\mathcal{K} = \{suppression, insertion, edit\}$. If an $\mathcal{K}$-automata outputs a sequence of actions σ then the controller operator $\triangleright_{\mathcal{K}}$ is able to infer the same sequence of actions. Also vice-versa holds.*

The proofs are given in the appendix.

[1]Here, we consider a multiple step relation in this rule; it could be possible to define a one step rule by inserting additional control actions. A more precise correspondence is thus worth of investigation but out of the scope of this paper.

7 Conclusion and future work

We illustrated some preliminary results towards a uniform theory for enforcing security properties based on a process calculi and logical formalization of security properties. With respect to prior work in the area we also add the possibility to automatically build enforcing mechanisms. Much work need to be done in order to make our approach more feasible in practice. We argue that there are many security properties whose corresponding controller may be built more efficiently. Moreover, the comparison among our framework and others should be better developed.

With this work, we contribute to extend a framework based on process calculi that have been shown to be very suitable to model and verify security properties. We argue that extending our approach to consider timed security properties should be possible and worth of investigation.

References

[1] 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.

[2] 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.

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

[4] 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.

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

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

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

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

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

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

[11] F. Martinelli. A uniform approach for the analysis of security protocols and access control systems. *FMOODS 2005*, LNCS, 2005.

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

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

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

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

A Partial evaluation function for parallel operator

In this section we give some rules for partial evaluation function for parallel operator given by Andersen in [1]:

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

B Semantics of equational μ-calculus

We give the formal semantics of equational μ-calculus (see [1]). It is assumed that variables appear only once on the left-hand sides of the equations of the list. We denote the set of this variable with $DefD$. Let $\langle S, A, \{\xrightarrow{a}\}_{a\in A}\rangle$ be an LTS extended with ρ an enviroment that assigns subsets of S to the variables that appear in the assertions of D, but which are not in $Def(D)$. The semantics $[\![A]\!]'_\rho$ of an assertion A is the following:

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

The semantics of a list of equations D, $[\![D]\!]_\rho$ is an enviroment that assigns subsets of S to variables in $Def(D)$. A list of equations is closed if every variable that appears in the assertions of the list is in $Def(D)$. 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.

$$\begin{aligned}
[\![\epsilon]\!]_\rho &= [] \\
[\![X =_\sigma AD']\!] &= [\![D']\!]_{(\rho\sqcup[U'/X])} \sqcup [U'/X]
\end{aligned}$$

where

$$U = \sigma U.[\![A]\!]'_{(\rho\sqcup[U/X]\sqcup\rho'(U))} \text{ and } \rho'(U) = [\![D']\!]_{(\rho\sqcup[U/X])}$$

It informally says that *the solution to* $(X =_\sigma A)D$ *is the* σ *fixpointsolution* U' *of* $[\![A]\!]$ *where the solution to the rest of the lists of equations* D *is used as environment*.

C Operational Semantics of SPA

Prefixing:

$$\frac{}{\alpha.E \xrightarrow{\alpha} E}$$

Choice:

$$\frac{E_1 \xrightarrow{a} E_1'}{E_1 + E_2 \xrightarrow{a} E_1'} \quad \frac{E_2 \xrightarrow{a} E_2'}{E_1 + E_2 \xrightarrow{a} E_2'}$$

Parallel:

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

Constant:

$$\frac{Z \doteq E \quad E \xrightarrow{\alpha} E'}{Z \xrightarrow{\alpha} E'}$$

Restriction:

$$\frac{E_1 \xrightarrow{\alpha} E_1'}{E_1 \backslash L \xrightarrow{\alpha} E_1' \backslash L} (\alpha \notin L \cup \overline{L})$$

Figure 1: Operational semantics for SPA.

D Technical proofs

Proposition 3.1 For every X exists an Y such that:

$$Y \triangleright X \sim Y$$

Proof : We show that the following relation is a strong bisimulation:

$$\mathcal{R} = \{(E \triangleright F, E) \mid E \in Proc(Y), F \in Proc(X)\}$$

where $Proc(Y)$ is the set of process that Y can do, $Proc(X)$ is the same for X.

$(E \triangleright F, E) \in \mathcal{R}$: Assume that $(E \triangleright F, E) \in \mathcal{R}$ and $(E \triangleright F, E) \xrightarrow{a} (E \triangleright F, E)'$. According to given semantic rules, $(E \triangleright f)'$ can be $E' \triangleright F$ or $E' \triangleright F'$. For both of this cases, we have that exists E' s.t. $E \xrightarrow{a} E'$. We have also $(E' \triangleright F, E') \in \mathcal{R}$ or $(E' \triangleright F', E') \in \mathcal{R}$. It's depend on which rule we have applied.

$(E, E \triangleright F) \in \mathcal{R}$: Assume that is true the converse of the relation $\mathcal{R}$ and we have $E \to E'$. Using one of the two rules that we have for the monitoring operator $\triangleright$, we can have two different options for $(E \triangleright F)'$. In both cases exists $(E \triangleright F)'$ s.t. $(E \triangleright F) \xrightarrow{a} (E \triangleright F)'$ and $(E', (E \triangleright F)') \in \mathcal{R}^{-1}$.

□

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

Proof: With the same argument of the previous demonstration, we can proof that $Y \triangleright'' X \sim Y$ and $Y \triangleright''' X \sim Y$.

□

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

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

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

Proof: We have only the following rule:

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

We can notice behavior of $E \triangleright'''' F$ depend on behavior of E and on behavior of F. This two processes is not bisimilar because we can have $E \xrightarrow{a} E'$ and $F \overset{a}{\not\rightarrow} F'$. In this case we can not conclude $E \triangleright''' F \xrightarrow{a} E' \triangleright''' F'$.

$E \triangleright''' F$ and E are strongly similar: we assume that $E \triangleright''' F \xrightarrow{a} E' \triangleright''' F'$ and $F \xrightarrow{a} F'$ then, by the rule 12 we have that $E \xrightarrow{a} E'$ and, obviously, $(E' \triangleright''' F', E') \in \mathcal{R}$. With a similar reasoning we prove also that $Y \triangleright''' X \prec X$.

□

E Technical proofs about related work

In order to prove the theorem 6.1, we note that 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 and in our approach, we consider only finite state processes. We also have to note that in both of case we must consider only one action for single step. With this additional hypothesis we are able to effectively prove theorem 6.1.

We now report all the definition of automata that are given in [2]. We start giving some notation: with σ we denote a sequences of actions, $\cdot$ is the empty sequence, δ is a partial function $\delta : Act \times \mathcal{Q} \rightarrow \mathcal{Q}$, it specifies the transition function for the automata and indicates that the automata should accept the current input and move in a new state.

truncation automata The operational semantic of truncation automata is:

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

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

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

otherwise

suppression automata It is define as $(\mathcal{Q}, q_0, \delta, \omega)$ where $\omega : Act \times \mathcal{Q} \rightarrow \{-, +\}$ indicates whethere or not the action in question is to be suppressed (-) or emitted (+).

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

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

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

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

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

otherwise

isertion automata It is define as $(\mathcal{Q}, q_0, \delta, \gamma)$ where $\gamma : Act \times \mathcal{Q} \to \overrightarrow{Act} \times \mathcal{Q}$ that specifies the insertion of a finite sequence of actions into the program's action sequence.

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

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

$$(\sigma, q) \xrightarrow{b_1 \cdots b_n}_I (\sigma, q') \qquad \text{(I-Ins)}$$

if $\sigma = a;\sigma'$
and $\gamma(a,q) = b_1 \cdots b_n, q'$

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

otherwise

edit automata 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 wheter or not the action i question is to be suppressed (-) or emitted (+).

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

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 $\omega(a,q) = -$

$$(\sigma, q) \xrightarrow{b_1 \cdots b_n}_e (\sigma, q') \qquad \text{(E-Ins)}$$

if $\sigma = a;\sigma'$
and $\gamma(a,q) = b_1 \cdots b_n, q'$

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

otherwise

In order to guarantee homogeneity of notation, we assume to work with an LTS, since that both automata and sequential process are LTS ([12]). We should give a proof of theorem 6.1 proving that a bisimulation exists between automata and controller operator. First of all, we prove the following lemmas. From these results come, immediately the prove of the previous theorem.

Lemma E.1 *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.*

Proof: To simplify the notation, we chose to denote with the form (σ, q) a generic state of automata and with $E \triangleright_S F$ a generic state of the process. In order to define a relation of strong bisimulation $\mathcal{R}_S$, we underline that every couple (σ, q) of the suppression automata depend on δ and ω. Since that the process E is a constant, also it can depend on these two function. So we denote E with $E^{q,\omega}$. This process has the following definition:

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

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

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

We have two cases:

- – *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 P' s.t. $E^{q,\omega} \triangleright_S F \xrightarrow{a} P'$ and $((\sigma', q'), P') \in \mathcal{R}_S$. We have, by rule (13) and by definition of $E^{q,\omega}$, that if $E^{q,\omega} \xrightarrow{a} E^{q',\omega}$ and $F \xrightarrow{a} F'$ then $E^{q,\omega} \triangleright_S F \xrightarrow{a} E^{q',\omega} \triangleright_S F'$. Now $F' \overset{\sigma'}{\mapsto}$. So $((\sigma', q'), E^{q',\omega} \triangleright_S F') \in \mathcal{R}_S$ follows immediately.*
  - – *Let $(E^{q,\omega} \triangleright_S F, (\sigma, q))$ be in $\mathcal{R}_S^{-1}$ 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 $(E^{q',\omega} \triangleright_S F', (\sigma, q)') \in \mathcal{R}_S^{-1}$. For the rule S-StepA we have that (σ', q') is the solution we search for a similar reasoning to that we have illustrated in the previous point.*
- – *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 P' s.t. $E^{q,\omega} \triangleright_S F \xrightarrow{\tau} (E^{q,\omega} \triangleright_S F)'$ and $((\sigma', q'), P') \in \mathcal{R}_S$. We have, by rule (14) and by the definition of $E^{q,\omega}$, that if $E^{q,\omega} \not\xrightarrow{a} E^{q',\omega}$, $E^{q,\omega} \xrightarrow{-a} E^{q',\omega}$ where the action $-a$ is made by $E^{q,\omega}$ in order to verify that F performs the action a and then suppress it, 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 $((\sigma', q'), E^{q',\omega} \triangleright_S F') \in \mathcal{R}_S$ follows immediately.*
  - – *Let $(E^{q,\omega} \triangleright_S F, (\sigma, q))$ be in $\mathcal{R}_S^{-1}$ and $E^{q,\omega} \triangleright_S F \xrightarrow{\tau} E^{q',\omega} \triangleright_S F'$. We should prove that exists a $(\sigma, q)'$ s.t. $(\sigma, q) \xrightarrow{\tau}_S (\sigma, q)'$ and $(E^{q,\omega} \triangleright_S F', (\sigma, q)') \in \mathcal{R}_S^{-1}$ For the rule S-StepS we have that (σ', q') is the solution we search for a similar reasoning to that we have illustrated in the previous point.*

□

Lemma E.2 *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.*

Proof: To simplify the notation, we chose to denote with the form (σ, q) a generic state of automata and with $E \triangleright_I F$ a generic state of the process. In order to define a relation of strong bisimulation $\mathcal{R}_I$, we underline that every couple (σ, q) of the suppression automata depend on δ and γ. Since that the process E is a constant, also it can depend on these two function. So we denote E with $E^{q,\gamma}$. This process has the following definition:

$$\begin{aligned} E^{q,\gamma} &= a.E^{q',\gamma} && \text{if } \delta(a,q) = q' \\ &= +a.b_1 \cdots b_n.E^{q',\gamma} && \text{if } \gamma(a,q) = b_1 \cdots b_n, q' \end{aligned}$$

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

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

We have two cases:

- – *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* P' *s.t.* $E^{q,\gamma} \triangleright_I F \xrightarrow{a} P'$ *and* $((\sigma', q'), P') \in \mathcal{R}_I$*. We have, by rule (15) and by definition of* $E^{q,\gamma}$*, that if* $E^{q,\gamma} \xrightarrow{a} E^{q',\gamma}$ *and* $F \xrightarrow{a} F'$ *then* $E^{q,\gamma} \triangleright_I F \xrightarrow{a} E^{q',\gamma} \triangleright_I F'$*. Now* $F' \overset{\sigma'}{\mapsto}$*. So* $((\sigma', q'), E^{q',\gamma} \triangleright_I F') \in \mathcal{R}_I$ *follows immediately.*
  - – *Let* $(E^{q,\gamma} \triangleright_I F, (\sigma, q))$ *be in* $\mathcal{R}_I^{-1}$ *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* $(E^{q',\gamma} \triangleright_I F', (\sigma, q)') \in \mathcal{R}_I^{-1}$*. For the rule I-Step we have that* (σ', q') *is the solution we search for a similar reasoning to that we have illustrated in the previous point.*

- – *Let* $((\sigma, q), E^{q,\gamma} \triangleright_I F)$ *be in* $\mathcal{R}_I$ *and* $(\sigma, q) \overset{b_1 \cdots b_n}{\longrightarrow}_I (\sigma, q')$*. We should prove that exists a* P' *s.t.* $E^{q,\gamma} \triangleright_I F \overset{b_1 \cdots b_n}{\longrightarrow} P'$ *and* $((\sigma, q'), P') \in \mathcal{R}_I$*. We have, by rule (16) and by to the definition of* $E^{q,\gamma}$*, that if* $E^{q,\gamma} \not\xrightarrow{a} E^{q'',\gamma}$*,* $E^{q,\gamma} \overset{+a.b_1 \cdots b_n}{\longrightarrow} E^{q',\gamma}$*, where* $+a$ *is an action made by the process* $E^{q,\gamma}$ *to verify that* F *performs the action* a *and then the process* $E^{q,\gamma}$ *performs the sequence of actions* $b_1 \cdots b_n$*, and* $F \xrightarrow{a} F'$ *then* $E^{q,\gamma} \triangleright_I F \overset{b_1 \cdots b_n}{\longrightarrow} E^{q',\gamma} \triangleright_I F$*. So* $((\sigma, q'), E^{q',\gamma} \triangleright_I F) \in \mathcal{R}_I$ *follows immediately.*
  - – *Let* $(E^{q,\gamma} \triangleright_I F, (\sigma, q))$ *be in* $\mathcal{R}_I^{-1}$ *and* $E^{q,\gamma} \triangleright_I F \overset{b_1 \cdots b_n}{\longrightarrow} E^{q',\gamma} \triangleright_I F$*. We should prove that exists a* $(\sigma, q)'$ *s.t.* $(\sigma, q) \overset{b_1 \cdots b_n}{\longrightarrow} (\sigma, q)'$ *and* $(E^{q',\gamma} \triangleright_I F, (\sigma, q)') \in \mathcal{R}_I^{-1}$*. For the rule I-Ins we have that* (σ, q') *is the solution we search for a similar reasoning to that we have illustrated in the previous point.*

□

Lemma E.3 *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.*

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* $\triangleright_E$*.*

$$\begin{aligned} E^{q,\gamma,\omega} &= a.E^{q',\gamma,\omega} && \text{if } \delta(a,q) = q' \text{ and } \omega(a,q) = + \\ &= -a.E^{q',\gamma,\omega} && \text{if } \delta(a,q) = q' \text{ and } \omega(a,q) = - \\ &= +a.b_1 \cdots b_n.E^{q',\gamma,\omega} && \text{if } \gamma(a,q) = b_1 \cdots b_n, q' \end{aligned}$$

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

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

We have three cases:

- – *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* P' *s.t.* $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{a}_e P'$ *and* $((\sigma', q'), P') \in \mathcal{R}_E$*. We have, by rule (17) 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* $F' \overset{\sigma'}{\mapsto}_e$*. So* $((\sigma', q'), E^{q',\gamma,\omega} \triangleright_E F') \in \mathcal{R}_E$ *follows immediately.*
  - – *Let* $(E^{q,\gamma,\omega} \triangleright_E F, (\sigma, q))$ *be in* $\mathcal{R}_E^{-1}$ *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* $(E^{q',\gamma,\omega} \triangleright_E F', (\sigma, q)') \in \mathcal{R}_E^{-1}$*. For the rule E-StepA we have that* (σ', q') *is the solution we search for a similar reasoning to that we have illustrated in the previous point.*

- – *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* P' *s.t.* $E^{q,\gamma,\omega} \triangleright_E F \xrightarrow{\tau} P'$ *and* $((\sigma', q'), P') \in \mathcal{R}_E$*. We have, by rule 18 and by the definition of* $E^{q,\gamma,\omega}$*, that*

if $E^{q,\gamma,\omega} \not\xrightarrow{a} E^{q',\gamma,\omega}$, $E^{q,\gamma,\omega} \xrightarrow{-a} E^{q',\gamma,\omega}$ *where the action* $-a$ *is made by* $E^{q,\gamma,\omega}$ *in order to verify that* F *performs the action* a *and then suppress it, 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* $((\sigma', q'), E^{q',\gamma,\omega} \triangleright_E F') \in \mathcal{R}_E$ *follows immediately.*

– *Let* $(E^{q,\gamma,\omega} \triangleright_E F, (\sigma, q))$ *be in* $\mathcal{R}_S^{-1}$ *and* $E^{q,\omega} \triangleright_S F \xrightarrow{\tau} E^{q',\gamma,\omega} \triangleright_E F'$. *We should prove that exists a* $(\sigma, q)'$ *s.t.* $(\sigma, q) \xrightarrow{\tau}_e (\sigma, q)'$ *and* $(E^{q,\gamma,\omega} \triangleright_E F', (\sigma, q)') \in \mathcal{R}_E^{-1}$ *For the rule E-StepS we have that* (σ', q') *is the solution we search for a similar reasoning to that we have illustrated in the previous point.*

- – *Let* $((\sigma, q), E^{q,\gamma,\omega} \triangleright_E F)$ *be in* $\mathcal{R}_E$ *and* $(\sigma, q) \overset{b_1 \cdots b_n}{\Longrightarrow}_e (\sigma, q')$. *We should prove that exists a* P' *s.t.* $E^{q,\gamma,\omega} \triangleright_E F \overset{b_1 \cdots b_n}{\Longrightarrow} P'$ *and* $((\sigma, q'), P') \in \mathcal{R}_E$. *We have, by rule 19 and by the definition of* $E^{q,\gamma,\omega}$ *that if* $E^{q,\gamma,\omega} \not\xrightarrow{a} E^{q'',\gamma,\omega}$, $E^{q,\gamma,\omega} \xrightarrow{+a.b_1 \cdots b_n} E^{q',\gamma,\omega}$, *where* $+a$ *is an action made by the process* $E^{q,\gamma,\omega}$ *to verify that* F *performs the action* a *and then the process* $E^{q,\gamma,\omega}$ *performs the sequence of actions* $b_1 \cdots b_n$, *and* $F \xrightarrow{a} F'$ *then* $E^{q,\gamma,\omega} \triangleright_E F \overset{b_1 \cdots b_n}{\Longrightarrow} E^{q',\gamma,\omega} \triangleright_E F$. *So* $((\sigma, q'), E^{q',\gamma,\omega} \triangleright_E F) \in \mathcal{R}_E$ *follows immediately.*

– *Let* $(E^{q,\gamma,\omega} \triangleright_E F, (\sigma, q))$ *be in* $\mathcal{R}_E^{-1}$ *and* $E^{q,\gamma,\omega} \triangleright_E F \overset{b_1 \cdots b_n}{\Longrightarrow} E^{q',\gamma,\omega} \triangleright_E F$. *We should prove that exists a* $(\sigma, q)'$ *s.t.* $(\sigma, q) \overset{b_1 \cdots b_n}{\Longrightarrow} (\sigma, q)'$ *and* $(E^{q',\gamma} \triangleright_E F, (\sigma, q)') \in \mathcal{R}_E^{-1}$. *For the rule E-Ins we have that* (σ, q') *is the solution we search for a similar reasoning to that we have illustrated in the previous point.*

□