Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 168 (2007) 29–43

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

An Approach for the Specification, Verification and Synthesis of Secure Systems[⋆]

Fabio Martinelli[1]

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

Ilaria Matteucci[2]

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

Abstract

In this paper we describe an approach based on *open system analysis* for the *specification*, *verification* and *synthesis* of secure systems. In particular, by using our framework, we are able to model a system with a possible intruder and verify whether the whole system is secure, i.e. whether the system satisfies a given temporal logic formula that describes its secure behavior. If necessary, we are also able to automatically synthesize a process that, by controlling the behavior of the possible intruder, enforces the desired secure behavior of the whole system.

Keywords: open system analysis, partial model checking, secure systems analysis, synthesis of controller operators

1 Overview

In the last few years, research on the definition of formal methods for the analysis and the verification of security properties of systems has increased greatly. This is mainly due to the practical relevance of these systems and moreover to preliminary encouraging results achieved by the application of formal methods to security analysis.

Here we describe a logical approach for *specification, verification* and *synthesis* of secure systems by summarizing some results of the works [14,16,17].

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

[1] Email: `Fabio.Martinelli@iit.cnr.it`

[2] Email: `Ilaria.Matteucci@iit.cnr.it`

doi:10.1016/j.entcs.2006.12.003

The *specification* is the first step of the analysis of a system. The language for the description of properties and the one for the description of systems must have a clear formal semantics. We consider for specification both declarative languages as temporal logic, in particular equational μ-calculus, and operational ones as process algebras, in particular CCS (see [20]).

We then specify the security of a system as the specification of a property of an *open system*, by following the approach given in [14,16]. As a matter of fact the analysis of security properties is based on the idea that potential attackers should be analyzed as if they were un-specified components of a system. In this way we reduce security analysis to the analysis of open systems. The behavior of an open system may be not completely specified and may present some uncertainty. The main idea underling this approach is the following: at the beginning we have a system S and a temporal logic formula ϕ that describes a security property. It is possible that an intruder X works in parallel with S or it is also possible that X is a malicious component of S. In each of these cases we require that S composed with X $(S\|X)$ satisfies ϕ whatever X is.

The *verification* phase requires to check for any X that $(S\|X)$ satisfies the property ϕ. In principle, this corresponds to an unbounded number of classical model checking problems in closed systems[3]. Indeed, the universal quantification on all possible intruders makes this problem difficult to manage. In order to solve it we use the *partial model checking* technique. It is introduced by Andersen in [1] in order to deal with compositional analysis of concurrent system. By using this technique, we may focus only on X and the previous problem becomes a validity checking problem. As a matter of fact by using the partial model checking technique, the property ϕ is projected on another one, says $\phi' = \phi_{//_S}$, depending only on S and ϕ, that only the component X must satisfy. Here, there is still the universal quantification, but the problem is now a validity checking one, that has been solved for many logics, including μ-calculus.

We consider the following *synthesis* problem. Assume to have a system S that is secure in isolation, but that in composition with a certain component X does not enjoy the desired security property, say ϕ. Then, we are able to synthesize a process Y that controlling the component X guarantees the whole system with S works correctly, i.e. it satisfies ϕ.

Hence we have extended the line of research of [14,16] with a method for automatically enforcing the desired security properties (see [17,18,19]). We define process algebra operators called *controller operators* and denoted by $Y \triangleright X$ where Y is the *controller program* i.e. the process that controls the un-specified component X. In particular we define controller operators that are able to model security automata described in [3,4,22] for enforcing safety properties as well as others able to force *Non-interference* properties (see [19]), under certain assumptions.

As before we start from a system S and a security property ϕ and we project

[3] Actually, there exists a verification problem, called module checking, i.e. model checking of open system, introduced in [12]. Such a problem can be solved using the technique we are going to present here, e.g. see [15].

ϕ on ϕ' by partial model checking. In this way we have to monitor only the necessary/untrusted part of the system, here X. Then we can force X to enjoy ϕ' by using an appropriate controller $Y \triangleright X$. Moreover, our approach permits us to automatically synthesize a controller program Y for a given controller operator $Y \triangleright X$ by exploiting satisfiability procedure on process algebra and temporal logic.

We also show a related specification framework called $GNDC$ (e.g., see [8]) that is able to describe security properties, e.g. *Non-interference, Agreement, Authentication, Non-Repudiation* and so on. By using this schema we are also able to uniformly model *dependable systems* and analyze *dependablity properties*. $GNDC$ was firstly introduced in [8] as a framework where family of security properties could be uniformly expressed and compared. Generally speaking a $GNDC$ property has the following form:

$$S \text{ satisfies } GNDC_{\triangleleft}^{\alpha(S)} \text{ iff } \forall\ X\ S\|X \triangleleft \alpha(S)$$

This means that a system S enjoys $GNDC_{\triangleleft}^{\alpha(S)}$ iff S shows w.r.t. a certain *behavioral relation* [4] $\triangleleft$, the same behavior of $\alpha(S)$. This is must to be true even if S is composed with a possible un-trusted component X, whatever it is. By using characteristic formulae (e.g. see [21]) for expressing the relation $\triangleleft$, we can reduce this problem to a usual open system analysis one.

Summing up our aim is to present a logical approach based on *open system* analysis and *partial model checking* technique for the specification, verification and synthesis of secure systems.

This paper is organized as follows. Section 2 briefly recalls the basic theory about process algebra and temporal logic. Section 3 explains our approach for the specification and verification of secure systems. Section 4 presents how we are able to define and synthesize controller programs. Section 5 shows a related approach used also to deal with dependability properties. Eventually, Section 6 concludes the paper.

2 Background

2.1 Process Algebra: CCS

CCS (see [20]) is a calculus for describing the behavior of concurrent processes.

The CCS language assumes a set $Act = \mathcal{L} \cup \bar{\mathcal{L}}$ of (observable) *communication actions* built from a set $\mathcal{L}$ of names and a set $\bar{\mathcal{L}}$ of co-names. The purpose of putting a line, called complementation, over a names is to show that the corresponding action can synchronize with its complemented action. Complementation follows the rule that $\bar{\bar{a}} = a$, for any communication action $a \in Act$.

A special symbol, τ, is used to model any (unobservable) *internal action*; hence the full set of possible actions is $Act_\tau = Act \cup \{\tau\}$. We let $a, b, \ldots$ range over

[4] There are a lot of different behavioral relations that can be studied. In particular we are interested in simulation, bisimulation and trace equivalences.

Prefixing:

$$\frac{}{a.P \xrightarrow{a} P}$$

Choice:

$$\frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$$

Parallel:

$$\frac{P \xrightarrow{a} P'}{P \| Q \xrightarrow{a} P' \| Q} \qquad \frac{Q \xrightarrow{a} Q'}{P \| Q \xrightarrow{a} P \| Q'} \qquad \frac{P \xrightarrow{l} P' \quad Q \xrightarrow{\bar{l}} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'}$$

Restriction:

$$\frac{P \xrightarrow{a} P'}{P \backslash L \xrightarrow{a} P' \backslash L}$$

Relabeling:

$$\frac{P \xrightarrow{a} P'}{P[f] \xrightarrow{f(a)} P'[f]}$$

Constant:

$$\frac{P \xrightarrow{a} P'}{A \xrightarrow{a} P'}$$

Table 1
SOS system for *CCS*.

Act_τ. The following grammar specifies the syntax of the language defining all *CCS* processes:

$$P, Q ::= \mathbf{0} \mid a.P \mid P + Q \mid P \| Q \mid P \backslash L \mid P[f] \mid A$$

where $L \subseteq Act$ and the relabeling function $f : Act_\tau \mapsto Act_\tau$ must be such that $f(\tau) = \tau$.

Informally, $\mathbf{0}$ is the process that does not perform any action. $a.P$ is the process ready to perform the action a, then, it behaves as P. Process $P + Q$ can choose *non-deterministically* to behave either as P or as Q. $P \| Q$ is the *parallel operator* where P and Q evolve concurrently. In $P \backslash L$, actions $a \in L \cup \bar{L}$ are prevented from happening. $P[f]$ is the process obtained from P by changing each $a \in Act_\tau$ into $f(a)$. A process identifier A defines a process and it is assumed that each identifier A has a defining equation of the form $A \doteq P$.

The operational semantics of *CCS* terms (see [20]) is described by a *labeled transition system* that is a tuple $(\mathcal{E}, Act_\tau, \rightarrow)$, where $\mathcal{E}$ is the set of all *CCS* terms and $\rightarrow \subseteq \mathcal{E} \times Act_\tau \times \mathcal{E}$ is a *transition relation* defined by structural induction as the least relation generated by the set of *Structural Operational Semantics* (*SOS*) rules of Table 1. The transition relation $\rightarrow$ defines the usual concept of derivation in one step. As a matter of fact $P \xrightarrow{a} P'$ means that process P evolves in one step into process P' by executing action $a \in Act_\tau$. The transitive and reflexive closure of $\bigcup_{a \in Act_\tau} \xrightarrow{a}$ is written $\rightarrow^*$.

Given a *CCS* process P, $Der(P) = \{P' | P \rightarrow^* P'\}$, is the set of its derivatives.

A CCS process P is said *finite state* if $Der(P)$ is finite. $Sort(P)$ (called the *sort of* P) is the set of names of actions that syntactically appear in the process P.

2.2 Behavioral Equivalences

Several behavioral relations are defined in order to compare the behavior of different processes.

2.2.1 Simulation and Bisimulation Equivalences

Definition 2.1 Let $(\mathcal{E}, Act_\tau, \rightarrow)$ 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 $(\mathcal{E}, Act_\tau, \rightarrow)$ if and only if, whenever $(E, F) \in \mathcal{R}$ we have:

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

A *strong bisimulation* is a relation $\mathcal{R}$ s.t. both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong simulations. We represent with $\sim$ the union of all the strong bisimulations.

We give the notion of *observational relations* as follows: $E \overset{\tau}{\Rightarrow} E'$ (or $E \Rightarrow E'$) if $E \overset{\tau}{\rightarrow}^{*} E'$; for $a \neq \tau$, $E \overset{a}{\Rightarrow} E'$ if $E \overset{\tau}{\Rightarrow}\overset{a}{\rightarrow}\overset{\tau}{\Rightarrow} E'$[5]. Let $\gamma \in Act^*$ be a sequence of actions, i.e. $\gamma = a_1, \ldots, a_n$, then $E \overset{\gamma}{\Longrightarrow} E'$ iff there exist $E = E_0, E_1, \ldots, E_n = E'$ s.t. $E_0 \overset{a_1}{\Longrightarrow} E_1 \ldots E_{n-1} \overset{a_n}{\Longrightarrow} E_n$.

The *weak bisimulation* relation (see [20]) permits to abstract to some extent from the internal behavior of the systems, represented by the internal τ action.

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

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

A weak bisimulation is a relation $\mathcal{R}$ s.t. both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are weak simulations. We represent with $\approx$ the union of all the weak bisimulations.

Every strong simulation is also a weak one (see [20]).

2.2.2 Trace Equivalence

Most of the security properties are based on the simple notion of *traces*: two processes are equivalent if they exactly show the same execution sequences (called *traces*). In order to formally define traces, we define *trace preorder* ($\leq_{trace}$) and *trace equivalence* ($\approx_{trace}$) as follows.

Definition 2.3 For any $E \in \mathcal{E}$ the set $T(E)$ of *traces associated with* E is $T(E) = \{\gamma \in Act^* \mid \exists E' : E \overset{\gamma}{\Longrightarrow} E'\}$. F can execute all traces of E (notation $E \leq_{trace} F$) iff $T(E) \subseteq T(F)$. E and F are *trace equivalent* (notation $E \approx_{trace} F$) iff $E \leq_{trace} F$ and $F \leq_{trace} E$, i.e. iff $T(E) = T(F)$.

[5] Note that it is a short notation for $E \overset{\tau}{\Rightarrow} E_\tau \overset{a}{\rightarrow} E'_\tau \overset{\tau}{\Rightarrow} E'$ where E_τ and E'_τ denote intermediate states that is not important for this framework.

$$[\![\mathbf{T}]\!]'_\rho = S \quad [\![\mathbf{F}]\!]'_\rho = \emptyset \quad [\![Z]\!]'_\rho = \rho(Z) \quad [\![\phi_1 \wedge \phi_2]\!]'_\rho = [\![\phi_1]\!]'_\rho \cap [\![\phi_2]\!]'_\rho$$
$$[\![\phi_1 \vee \phi_2]\!]'_\rho = [\![\phi_1]\!]'_\rho \cup [\![\phi_2]\!]'_\rho \quad [\![\langle a\rangle\phi]\!]'_\rho = \{s \mid \exists s' : s \xrightarrow{a} s' \text{ and } s' \in [\![\phi]\!]'_\rho\}$$
$$[\![[a]\phi]\!]'_\rho = \{s \mid \forall s' : s \xrightarrow{a} s' \text{ implies } s' \in [\![\phi]\!]'_\rho\}$$

Table 2
Equational μ-calculus

2.3 Equational μ-Calculus

The equational μ-calculus is a modal logic (see [5]) based on fix-point equations. Let Z be a variable ranging over a set V of variables, a least (greatest) fix-point equation is $Z =_\mu \phi$ ($Z =_\nu \phi$), where ϕ is an *assertion*. The syntax of assertions (ϕ) and of lists of equations ($\mathcal{D}$) is defined as follows:

$$\begin{array}{ll} \textit{assertion} & \phi ::= \mathbf{T} \mid \mathbf{F} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle a\rangle\phi \mid [a]\phi \\ \textit{equations list} & \mathcal{D} ::= (Z =_\mu \phi)\mathcal{D} \mid (Z =_\nu \phi)\mathcal{D} \mid \epsilon \end{array}$$

where the symbol $\mathbf{T}$ means *true* and $\mathbf{F}$ means *false*; $\wedge$ is the symbol for the conjunction of formulae, i.e. $\phi_1 \wedge \phi_2$ holds iff both of the formulae ϕ_1 and ϕ_2 hold, and $\vee$ is the disjunction of formulae and $\phi_1 \vee \phi_2$ holds when either ϕ_1 or ϕ_2 holds. The *possibility operator* $\langle a\rangle\phi$ means that "there exists a transition labeled by a after that ϕ holds". The *necessity operator* $[a]\phi$ means "for all a-actions performed ϕ holds".

The semantics of the equational μ-calculus is defined over labeled transition systems. In order to give the semantics of an equation list we show our notation: let $\mathcal{M}$ be a labeled transition system and ρ be a function, called environment, from variables to a subset of the set of states of $\mathcal{M}$, $\sqcup$ represents the union of disjoint environments, and $[]$ denotes the empty environment. Let σ be in $\{\mu, \nu\}$, $\sigma U.f(U)$ represents the σ fix-point of the function f in one variable U. The semantics, $[\![\mathcal{D}]\!]_\rho$ is defined by the following equations:

$$[\![\epsilon]\!]_\rho = [] \quad [\![(Z =_\sigma \phi)\mathcal{D}]\!]_\rho = [\![\mathcal{D}]\!]_{(\rho\sqcup[U'/Z])} \sqcup [U'/Z]$$

where $U' = \sigma U.[\![\phi]\!]_{(\rho\sqcup[U/Z]\sqcup\rho'(U))}$ and $\rho'(U) = [\![D]\!]_{(\rho\sqcup[U/Z])}$

Informally $[\![(Z =_\sigma \phi)\mathcal{D}]\!]_\rho$ says that the solution to $(Z =_\sigma \phi)\mathcal{D}$ is the σ fixed point solution U' of $[\![\phi]\!]_\rho$ where the solution to the rest of the list of equations $\mathcal{D}$ is used as environment. A labeled transition system $\mathcal{M}$ satisfies an equation list $\mathcal{D}$, written $\mathcal{M} \models_\rho \mathcal{D} \downarrow Z$ if the initial state of $\mathcal{M}$ is in $[\![\mathcal{D}]\!]_\rho Z$, where Z is the first variable in the list $\mathcal{D}$. We omit ρ out when it is evident from the context or when $\mathcal{D}$ is closed.

The semantics, $[\![\phi]\!]_\rho$, of an assertion ϕ is defined in Table 2.

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

Theorem 2.4 ([23]) *Given a formula ϕ it is possible to decide in exponential time in the length of ϕ if there exists a model of ϕ and it is also possible to give an* *example of such model.*

2.4 Partial Model Checking

Partial model checking is a technique that relies upon compositional methods for proving properties of concurrent systems [1,2].

The intuitive idea underlying the partial model checking is the following: proving that $E \| F$ satisfies ϕ is equivalent to prove that F satisfies a modified specification $\phi = \phi_{//_E}$, where $//_E$ is the partial evaluation function for the parallel composition operator (see Table A.1 in Appendix)[6].

Hence, the behavior of a component has been partially evaluated and the requirements are changed in order to respect this evaluation.

We give the following main result (see [2]).

Lemma 2.5 *Given a process $E \| F$ and an equational specification $\mathcal{D} \downarrow Z$ we have:*

$$E \| F \models \mathcal{D} \downarrow Z \quad \textit{iff} \quad F \models \mathcal{D} \downarrow Z_{//_E}$$

A lemma similar to the previous one holds for each CCS operator.

2.4.1 Characteristic Formulae

A *characteristic formula* is a formula in equational μ-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 finite state process E w.r.t. several behavioral relations (see [21]). Here we present the definition of characteristic formula w.r.t. (weak) simulation as follows.

Definition 2.6 Given a finite state process, its characteristic formula (w.r.t. weak simulation) $D_E \downarrow Z_E$ is defined by the following equations: for every $E' \in Der(E)$, $Z_{E'} =_\nu \bigwedge_{a \in Act}([a](\bigvee_{E'':E' \overset{a}{\Rightarrow} E''} Z_{E''}))$.

Following the reasoning used in [21], the following proposition holds.

Lemma 2.7 *Let E be a finite-state process and let $\phi_{E,\preceq}$ be its characteristic formula w.r.t. simulation, then $F \preceq E \Leftrightarrow F \models \phi_{E,\preceq}$.*

3 Specification and Verification of Secure Systems

Following the approach proposed in [14,16], we describe here a methodology for the formal analysis of secure systems based on the concept of open systems and partial model checking technique.

[6] We present the partial model checking technique w.r.t. parallel operator because its application w.r.t this operators is more intuitive than w.r.t. another CCS operator.

3.1 Open Systems Analysis for Security

A system is *open* if it has some unspecified components. We want to make sure that the system with an unspecified component works properly, e.g. fulfills a certain property. Thus, the intuitive idea underlying the verification of an open system is the following:

> *An open system satisfies a property if and only if, whatever component is substituted to the unspecified one, the whole system satisfies this property.*

In the context of formal languages for the description of system behavior, an open system may be simply regarded as a term of this language which may contain "holes" (or placeholders). These are unspecified components. For instance $A\|(\_)$ and $A\|B\|(\_)$ may be considered as open systems.

The main idea is that, when analyzing security-sensitive systems, neither the enemy's behavior nor the malicious users' behavior should be fixed beforehand. A system should be secure regardless of the behavior the malicious users or intruders may have, which is exactly a *verification* problem of open systems. According to [14,16], for defining security properties as open systems properties we study the following problem:

$$\text{For every component } X \quad S\|X \models \phi \tag{1}$$

where X stands for a possible enemy, S is the system under examination and ϕ is a (temporal) logic formula expressing the security property. It roughly states that the property ϕ holds for the system S, regardless of the component (i.e. intruder, malicious user, hostile environment, *etc.*) which may possibly interact with it.

Our aim is to reduce such a verification problem as in Formula (1) to a validity checking problem. To obtain this, we apply the partial model checking techniques as follows:

$$\forall X \quad S\|X \models \phi \quad \text{iff} \quad X \models \phi_{//S} \tag{2}$$

In this way we find the sufficient and necessary condition on X, expressed by the logical formula $\phi_{//S}$, so the whole system $S\|X$ satisfies ϕ if and only if X satisfies $\phi_{//S}$.

Several results exist about the decidability of such problems for temporal logic and, for several interesting properties, like several *safety properties* ("nothing bad happens"), the validity problem expressed by Formula (2) may be efficiently solved.

4 Synthesis of Controller Programs

In previous sections we have presented our approach for analyzing secure systems as open systems. As we have already said, the universal quantification over all possible intruders in Formula (1) it is not easy to manage.

Our aim in this section is to present our method to enforce a system to behave correctly whatever the behavior of the target is. To do this we define several *process algebra controller operators* that permit to control possible un-trusted behaviors of a target. We denote them by $Y \triangleright X$, where X is the target and Y is a *controller* *program* i.e. the process that controls X in order to guarantee that a given security property is satisfied.

By using a controller operator the specification of the system changes from Formula (1) to:

$$\exists Y\, \forall X \quad \text{s.t.} \quad S \| (Y \triangleright X) \models \phi \tag{3}$$

By partially evaluating ϕ w.r.t. S the Formula (3) is reduced as follows:

$$\exists Y\, \forall X \quad Y \triangleright X \models \phi' \tag{4}$$

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

It is important to note that, by using partial model checking we need to control only the possible un-trusted component of the system. This is an advantage of our approach because often not all the system needs to be checked or it is simply not convenient to check it as a whole or also it is not possible to do. Some components could be trusted and one would like to have a method to constrain only the untrusted ones (e.g. downloaded applets). Our method allows one to monitor only the necessary/untrusted part of the system, here X. Sometimes it could be possible that not the whole system can be checked but only some of its components.

Moreover, for some security properties, we are able to automatically synthesize a controller program for a controller operator.

4.1 Controller Operators

We can define several kinds of controller operators. Each of them has different capabilities. For instance, in [17,18,19] we have dealt with security automata (*truncation, suppression, insertion, edit*) defined in [3,4] by modeling them by process algebra controller operators $Y \triangleright_K X$, where $K \in \{T, S, I, E\}$ [7].

Referring to [3,4], we recall the informal definition of security automata as follows:

Truncation automata: The truncation automaton (similar to Schneider's ones (see [22])) can recognize bad sequences of actions and halt the program execution before security property is violated, but cannot otherwise modify the program behavior.

Suppression automata: The suppression automaton can halt program execution and suppress individual program actions without terminating the program outright.

Insertion automata: The insertion automaton can insert a sequence of actions into the program actions stream as well as terminate the program.

Edit automata: The edit automaton combines the power of suppression and insertion automata. It can truncate actions sequences and insert or suppress security-relevant actions at will.

[7] T stays for *Truncation*, S for *Suppression*, I for *Insertion* and E for *Edit*.

According to [3,4], these operators are applied in order to enforce safety properties. As a matter of fact for this class of formulae it is possible to prove that if E and F are two processes, s.t. $F \preceq E$ then $E \models \phi \Rightarrow F \models \phi$. In [18] we have proven that $Y \triangleright_T X$ is weakly similar to Y. Hence, in order to satisfy the Formula (4) it is sufficient to prove the following one:

$$\exists Y \quad Y \models \phi' \tag{5}$$

In this case we obtain a satisfiability problem in μ-calculus, that can be solved by Theorem 2.4. Hence we are able to find a process Y that halts the execution of the target whenever it is unsafe.

It is important to note that a similar result can be proven also for the other controller operators. As a matter of fact, by defining appropriate relabeling function f_K, we have proven that $Y \triangleright_K X$ is weakly similar to $Y[f_K]$ for $K \in \{S, I, E\}$. So, by partial model checking w.r.t. relabeling operator (see Table A.1 in Appendix) we are able to calculate $\phi'' = \phi_{//_{[f_K]}}$ to reduce Formula (4) as follows:

$$\exists Y \quad Y \models \phi''\,[8] \tag{6}$$

Also in this case we can solve the problem by Theorem 2.4.

Other controller operators can be defined in order, for example, to enforce not only safety properties but also *information flow properties* (see [19]).

5 A General Schema for Security and Dependability Properties: The *GNDC* Schema

Referring to the open system approach defined before, here we present a general schema to specify several security properties. It is called *Generalized NDC*, $GNDC^{\alpha}_{\lhd}$ for short, where $\lhd$ and α are two parameters that express a behavioral equivalence and a property respectively. (It is a generalization of *Non Deducibility on Compositions*,NDC for short, (see [6]).) This general schema permits to study relationships among different security properties in a fairly simple way. Indeed, their comparison can be carried out by simply studying the relations among the relative α's and $\lhd$'s. It is worth noticing that some of the properties we consider have been proposed for completely different aims. For instance, NDC has been introduce for studying *non-interference properties* in non-deterministic systems while *Agreement* has been proposed for the analysis of entity authentication in protocols.

The idea is similar to the one of the analysis as open systems, but it considers as correct specification of the behavior another process rather than a formula.

The main idea is that a system E is $GNDC^{\alpha}_{\lhd}$ iff for every process X the composition of the system with such a X satisfies a specification $\alpha(E)$. Essentially, $GNDC^{\alpha}_{\lhd}$ guarantees that the property identified by α is satisfied, w.r.t. $\lhd$ relation even when the system is composed with any possibly hostile process.

[8] The interested reader can find more details in [18].

Definition 5.1 E is $GNDC^{\alpha}_{\lhd}$ *iff*

$$\forall X \quad E\|X\backslash H \lhd \alpha(E)$$

The property is parametric with respect to $\alpha(E)$ and $\lhd$ that can be instantiated in order to obtain different security properties. In particular $\alpha : \mathcal{E} \to \mathcal{E}$ is a function between processes and $\alpha(E)$, w.r.t. a given E, specifies which should be the "correct" (intended) behavior of E; $\lhd \subset \mathcal{E} \times \mathcal{E}$ is a relation between processes that represents our notion of "observation". The idea is that just by studying different α functions, one could compare different properties. This has been useful to formally show, as the intuition suggests, that non-interference properties are usually the strongest ones. (The interested reader may check [7] for a deeper discussion.)

As a matter of fact, we can instantiate the $GNDC$ schema to obtain several properties. For instance, we can define NDC, $BNDC$, *Agreement*, *authentication* and *non-repudiation* by choosing particular instances of $\lhd$ and $\alpha(E)$.

Example 5.2 As we have already said, *Non Deducibility on Compositions*, NDC for short, (see [6]) has been proposed as a generalization of the classical idea of *Non-Interference* (see [10]) to non-deterministic systems.

Since $GNDC^{\alpha}_{\lhd}$ is a generalization of NDC, it can be instantiated in order to obtain NDC and also the *bisimulation* based NDC, called $BNDC$. We first redefine in our extended language the original definition as follows:

$$E \text{ is } NDC \text{ iff } \forall\, \Pi \in \textit{ High} \text{ users }, E\|\Pi\backslash H \approx_{trace} E\backslash H \text{ w.r.t. } \textit{Low} \text{ users}$$

where H is a set of high actions. NDC requires that high level processes Π are not able to change the low level behavior of the system represented by $E\backslash H$. As a matter of fact $E \setminus H$ is the system where no high level activity is allowed. If it is equivalent to $E\|\Pi\backslash H$ this clearly means that Π is not able to modify in any way the execution of E.

We can obtain a bisimulation based NDC by simply substituting $\approx_{trace}$ with $\approx$.

$$E \text{ is } BNDC \text{ iff } \forall\, \Pi \in \textit{ High} \text{ users }, E \setminus H \approx E\|\Pi \setminus H.$$

Note that NDC and $BNDC$ correspond to $GNDC^{E\backslash H}_{\approx_{trace}}$ and $GNDC^{E\backslash H}_{\approx}$, respectively.

Example 5.3 The approach proposed in [13] for the analysis of authentication properties, inside the framework of CSP [11] process algebra, can be rephrased in terms of our specification schema. The basic idea of the *Agreement* property is the following:

> "A protocol guarantees to an initiator A *agreement* with a responder B on a set of data items ds if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the variables in ds, and each such run of A corresponds to a unique run of B".

What is technically done in the *agreement* property is to have for each party an action representing the running of the protocol and another one representing the completion of it. Hence

$$E \text{ satisfies } Agreement \text{ iff } E \text{ is } GNDC^{\alpha_{Agree}(E)}_{\leq_{trace}}.$$

where α_{Agree} says that even in the presence of an hostile process, E does not execute wrong traces.

The universal quantification over all possible intruders is yet problematic when trying to check a property, since, in principle, we have to verify it over infinitely many processes, one for each intruder.

The *GNDC* schema has a favorite verification method based on the so-called most-general intruder idea. For several kind of relations, it is possible to avoid the universal quantification and just consider one possible intruder. Thus, standard techniques and tools may be applied. Unfortunately, for several interesting properties, e.g. *BNDC*, such approach does not work and the one based on logic presented in the previous sections should be applied.

5.1 Related Problems: GNDC in Dependability

It is possible to show that also *dependable systems* can be uniformly modeled in our framework and also *dependablity properties* can be analyzed within *GNDC* (see [9]). A system must be modeled by *CCS*, where both the *failing behavior* of the system and the related *fault-recovering procedures* are explicitly described. The environment acts as a *fault-injector* and it is the unspecified component of our framework. We call it *faulty environment*. We may note that the neat separation between the system and its environment given by *GNDC* is very useful.

The *GNDC* schema can be exploited for expressing properties peculiar of dependability analysis as *fail stop, fail safe, fail silent* (see [9] for the details). We briefly recall some definitions:

A failing system model is a *CCS* process $P_{\mathcal{F}}$ obtained by extending the process P with the possibility of executing particular external actions from a set $\mathcal{F}$ of possible *fault actions*. After each fault action, the relative *failure mode* is also specified in $P_{\mathcal{F}}$.

A fault tolerant system model is a *CCS* process $P^{\#}_{\mathcal{F}}$ obtained by adding to $P_{\mathcal{F}}$ some processes realizing error-recover strategies in accordance to some fault tolerant design strategy.

Occurrences of faults are induces by a *fault-injector* process $F_{\mathcal{F}}$ that causes faults to happen. It interacts with $P^{\#}_{\mathcal{F}}$ exactly through $f \in \mathcal{F}$ fault actions.

Now we are able to give the characterization of fault tolerance as *GNDC* property as follows.

Definition 5.4 A process P satisfies $GNDC^{\alpha(P)}_{\triangleleft}$ iff $\forall F_{\mathcal{F}} \in \mathcal{E}_{\mathcal{F}}\ (P^{\#}_{\mathcal{F}} \| F_{\mathcal{F}}) \backslash \mathcal{F} \triangleleft \alpha(P^{\#}_{\mathcal{F}})$ where $\mathcal{E}_{\mathcal{F}} = \{X | Sort(X) \subseteq \mathcal{F} \cup \{\tau\}\}$.

It is important to observe that the clear separation between the system model and the environment allows us to leave $F_{\mathcal{F}}$ unspecified and to range it over $\mathcal{E}_{\mathcal{F}}$.

As we have already said, different definitions of $\alpha(P)$ characterize different fault tolerance properties, e.g. fail stop, fail safe, fail silent and fault tolerance.

6 Conclusion

In this paper we have shown how security properties can be conveniently *specified* as properties of open systems. Moreover, these properties can be *verified* in a uniform way by using a few concepts of concurrency and temporal logic theory, as, for instance, partial model checking. Logic provides rigorous methods to reason about the uncertainty of the execution environment of security systems. The *synthesis* of secure systems is also possible by means of logical techniques. More generally, we aim at providing a uniform approach for the specification\analysis\synthesis for several application areas, e.g. fault-tolerance, non-interference, network and system security, open systems analysis and so on.

References

[1] Andersen, H., "Verification of Temporal Properties of Concurrent Systems," Ph.D. thesis, Department of Computer Science, Aarhus University, Denmark (1993).

[2] Andersen, H. R., *Partial Model Checking (Extended Abstract)*, in: *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press, 1995, pp. 398–407.

[3] Bauer, L., J. Ligatti and D. Walker, *More Enforceable Security Policies*, in: I. Cervesato, editor, *Foundations of Computer Security: Proceedings of the FLoC'02 workshop on Foundations of Computer Security* (2002), pp. 95–104.

[4] Bauer, L., J. Ligatti and D. Walker, *Edit Automata: Enforcement Mechanisms for Run-time Security Policies*, International Journal of Information Security **4** (2005), pp. 2–16.

[5] Bradfield, J. and C. Stirling, "Modal logics and mu-calculi: an introduction," Handbook of Process Algebra. Elsevier, 2001.

[6] Focardi, R. and R. Gorrieri, *A Classification of Security Properties for Process Algebras*, Journal of Computer Security **3** (1994/1995), pp. 5–33.

[7] Focardi, R., R. Gorrieri and F. Martinelli, *Classification of Security Properties – Part II: Network Security*, in: *Tutorial Lectures of the International School on Foundations of Security Analysis and Design (FOSAD'02)*, Lecture Notes in Computer Science **2946**, 2004, pp. 139–185.

[8] Focardi, R. and F. Martinelli, *A Uniform Approach for the Definition of Security Properties*, in: *FM'99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems–Volume I*, Lecture Notes in Computer Science **1708** (1999), pp. 794–813.

[9] Gnesi, S., G. Lenzini and F. Martinelli, *Applying Generalized Non Deducibility on Compositions (GNDC) Approach in Dependability*, Electronic Notes in Theoretical Computer Science **99** (2004), pp. 111–126.

[10] Goguen, J. A. and J. Meseguer, *Security Policy and Security Models*, in: *Proceedings of the 1982 Symposium on Security and Privacy* (1982), pp. 11–20.

[11] Hoare, C. A. R., "Communicating Sequential Processes," Prentice-Hall, 1985.

[12] Kupferman, O. and M. Y. Vardi, *Module checking*, in: Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification*, Lecture Notes in Computer Science **1102** (1996), pp. 75–86.

[13] Lowe, G., *A Hierarchy of Authentication Specification*, in: *Proceedings of the 10th Computer Security Foundation Workshop* (1997), pp. 31–43.

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

[15] Martinelli, F., *Module Checking through Partial Model Checking*, Technical Report 2002-TR-06, IIT–CNR (2002).

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

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

[18] Martinelli, F. and I. Matteucci, *Through modeling to Synthesis of Security Automata*, in: *Proceedings of the 2nd International Workshop on Security and Trust Management (STM'06)*, Electronic Notes in Theoretical Computer Science, 2006.

[19] Matteucci, I., *Automated synthesis of enforcing mechanisms for security properties in a timed setting*, in: *Proceedings of the Workshop on Information and Computer Security (ICS'06)*, Electronic Notes in Theoretical Computer Science, 2006.

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

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

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

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

A Tables of Partial Model Checking Function

Parallel:

$$(D{\downarrow} Z)//t = (D//t){\downarrow} Z_t \quad \epsilon//t = \epsilon$$

$$(Z =_\sigma \phi D)//t = ((Z_s =_\sigma \phi//s)_{s\in Der(E)})(D)//t \quad Z//t = Z_t$$

$$[a]\phi//s = [a](\phi//\text{s}) \wedge \bigwedge_{s\xrightarrow{a}s'} \phi//s', \text{ if } a \neq \tau \quad \phi_1 \wedge \phi_2//s = (\phi_1//s) \wedge (\phi_2//s)$$

$$\langle a\rangle\phi//s = \langle a\rangle(\phi//s) \vee \bigvee_{s\xrightarrow{a}s'} \phi//s', \text{ if } a \neq \tau \quad \phi_1 \vee \phi_2//s = (\phi_1//s) \vee (\phi_2//s)$$

$$[\tau]\phi//s = [\tau](\phi//s) \wedge \bigwedge_{s\xrightarrow{\tau}s'} \phi//s' \wedge \bigwedge_{s\xrightarrow{a}s'} [\overline{a}](\phi\ //s')$$

$$\langle\tau\rangle\phi//s = \langle\tau\rangle(\phi//s) \vee \bigvee_{s\xrightarrow{\tau}s'} \phi//s' \vee \bigvee_{s\xrightarrow{a}s'} \langle\bar{a}\rangle(\phi//_{s'}) \quad \mathbf{T}//s = \mathbf{T} \quad \mathbf{F}//s = \mathbf{F}$$

Restriction:

$$Z//\backslash L = Z \qquad (Z =_\sigma \phi D)//\backslash L = (Z =_\sigma \phi//\backslash L(D)//\backslash L)$$

$$\langle a\rangle\phi//\backslash L = \begin{cases} \langle a\rangle(\phi//\backslash L) & \text{if } a \notin L \cup \bar{L} \\ \mathbf{F} & \text{if } a \in L \end{cases} \qquad [a]\phi//\backslash L = \begin{cases} [a](\phi//\backslash L) & \text{if } a \notin L \cup \bar{L} \\ \mathbf{T} & \text{if } a \in L \end{cases}$$

$$\phi_1 \wedge \phi_2//\backslash L = (\phi_1//\backslash L) \wedge (\phi_2//\backslash L) \qquad \phi_1 \vee \phi_2//\backslash L = (\phi_1//\backslash L) \vee (\phi_2//\backslash L)$$

$$\mathbf{T}//\backslash L = \mathbf{T} \qquad \mathbf{F}//\backslash L = \mathbf{F}$$

Relabeling:

$$Z//[f] = Z \qquad (Z =_\sigma \phi D)//[f] = (Z =_\sigma \phi//[f](D)//[f])$$

$$\langle a\rangle\phi//[f] = \bigvee_{b:f(b)=a}\langle b\rangle(\phi//[f]) \qquad [a]\phi//[f] = \bigwedge_{b:f(b)=a}[b](\phi//[f])$$

$$\phi_1 \wedge \phi_2//[f] = (\phi_1//[f]) \wedge (\phi_2//[f]) \qquad \phi_1 \vee \phi_2//[f] = (\phi_1//[f]) \vee (\phi_2//[f])$$

$$\mathbf{T}//[f] = \mathbf{T} \qquad \mathbf{F}//[f] = \mathbf{F}$$

Table A.1
Partial evaluation function for parallel operator and relabeling operator.