Formal Specification and Validation of Security Policies

Tony Bourdier, Horatiu Cirstea, Mathieu Jaume, Hélène Kirchner

Formal Specification and Validation of Security Policies

Tony Bourdier[1], Horatiu Cirstea[1], Mathieu Jaume[2], and Hélène Kirchner[3]

[1] INRIA Nancy - GRAND-EST Research Center & Nancy University & LORIA
[2] SPI LIP6, Université Paris 6
[3] INRIA Bordeaux - SUD-OUEST Research Center

Abstract. We propose a formal framework for the specification and validation of security policies. To model a secured system, the evolution of security information in the system is described by transitions triggered by authorization requests and the policy is given by a set of rules describing the way the corresponding decisions are taken. Policy rules are constrained rewrite rules whose constraints are first-order formulas on finite domains, which provides enhanced expressive power compared to classical security policy specification approaches like the ones using Datalog, for example. Our specifications have an operational semantics based on transition and rewriting systems and are thus executable. This framework also provides a common formalism to define, compare and compose security systems and policies. We define transformations over secured systems in order to perform validation of classical security properties.

1 Introduction

When addressing the field of security policies in computer science, we are faced to multiple definitions of this concept, most often based on their purpose rather than on their behavior. For instance, in a very generic way, one can say that the purpose of a security policy is to define what it means to be secure for a system, an organization or another entity. With this point of view, security policies can be seen as special procedures that deliver authorizations to perform specific actions: for instance, they decide whether or not an access is granted, whether or not a transaction may be approved, possibly taking into account the history of transactions (e.g., on a bank account, the total amount of cash withdrawal during the month should not exceed a fixed amount), or priority considerations (e.g., an emergency call is always given priority).

The additional specificity of security policies is their reactive behaviour with respect to their execution environment: on one hand, a target system may query the policy for an authorization before performing specific accesses or transactions; on the other hand, the answers of the policy not only determine the way the corresponding action is handled in the system but can also modify the (security) information of the system and consequently subsequent executions. For example, a negative authorization from an ATM machine security policy due to an incorrect PIN not only prevents immediate money withdrawal but can also induce a (bad PIN) counter incrementation and lead to a permanent blocking of the corresponding account after a certain number of unsuccessful attempts. So, the security information could be seen as part of the target system but it is also intrinsic to the corresponding policy whose decisions strongly depend on it.

In this paper, we formalise separately the security system that manipulates all the security information that are used for producing the authorization decisions and the policy rules that compute the decisions. This separation is relevant not only for a conceptually clear specification and design, but also for the verification, comparison and composition of policies. In particular, this allows one to analyse separately properties related to the management of the security information (expressed as invariants of the security system) and properties related to the policy rules (consistency or completeness for example).

A security system is formalised as a transition system whose states are first-order structures generated by syntactic environments, and whose transitions are described by transition rules on environments. Each transition is triggered by an event which corresponds to an authorization given by the security policy. The policy is given as a set of rules describing the way the decisions are taken. Policy rules are constrained rewrite rules, whose constraints are first-order formulas that are solved in the current state of the transition system. According to the authorization, the transition rule may or may not apply. So, conceptually, the security policy restricts the possible transitions of the security system. Such specifications of security systems and policies have a well-understood operational semantics based on transition and rewriting systems and are thus executable.

A security policy is often expected to fulfill a certain security property expressed on some entities, while it is dealing with a different set of entities. A typical example is given by access control policies designed for ensuring flow properties: such policies do not deal with information flow but only with objects containing information to be traced. Intuitively, a link is needed between "what you do" (the policy) and "what you want" (the goal for which the policy is designed). We formalize this link through a transformation of environments, whose aim is to translate an environment into another one dealing with the entities we are interested in. Such a transformation allows thus the validation of a property over a system even if the property is expressed in a different specification. In practice, this approach provides a way to reuse the same specification of a security property in order to analyse or to verify several policies and systems, thus showing the benefits of a library of generic security properties, dedicated to particular domains (like information flows) and that can be considered in several contexts.

We first introduce some useful notions and notations in Section 2. Section 3 presents the different components of our specification framework: security signatures, environments, transition rules as well as security systems, policy rules, and secured systems. Section 4 addresses the validation point of view by defining environment transformations and illustrating the verification of security properties. In Section 5, we compare our approach with other works. Conclusion and future work are presented in Section 6.

2 Preliminaries

We assume the reader familiar with the standard notions of term rewriting, first order logic and Datalog. This section briefly recalls basic notions used in this paper; more details can be found in [12] for logic considerations, in [1] for rewriting considerations and in [18] for Datalog related notions.

A many-sorted signature $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P})$ is given by a set of sorts $\mathcal{S}$, a set of function symbols $\mathcal{F}$ and a set of predicate symbols $\mathcal{P}$. A function symbol f with arity $\mathfrak{s}_1, \ldots, \mathfrak{s}_n \in \mathcal{S}^*$ and co-arity $\mathfrak{s}$ is written $f{:}\mathfrak{s}_1, \ldots, \mathfrak{s}_n \mapsto \mathfrak{s}$. A predicate symbol p with arity $\mathfrak{s}_1, \ldots, \mathfrak{s}_n \in \mathcal{S}^*$ is written $p{:}\mathfrak{s}_1, \ldots, \mathfrak{s}_n$. Variables are also sorted and the notation $x{:}\mathfrak{s}$ specifies that the variable x has sort $\mathfrak{s}$. We assume in this paper that all variables are ranging over finite sets. This condition can be relaxed under some conditions [18], especially for allowing built-in sorts such as integers. Given a set ζ extending a set of variables $\mathcal{X}$ (possibly empty) with constants sorted by $\mathcal{S}$, the set of Σ-terms over ζ denoted by $\mathcal{T}^{\mathfrak{s}}_{\Sigma,\zeta}$ is the smallest set containing elements of ζ of sort $\mathfrak{s}$ and all the $f(t_1, \ldots, t_n)$ such that $f{:}\mathfrak{s}_1, \ldots, \mathfrak{s}_n \mapsto \mathfrak{s} \in \Sigma$ and $t_i \in \mathcal{T}^{\mathfrak{s}_i}_{\Sigma,\zeta}$ for $i \in [1..n]$. We write $\mathcal{T}^{\mathfrak{s}}_{\Sigma}$ instead of $\mathcal{T}^{\mathfrak{s}}_{\Sigma,\varnothing}$ and the sort is omitted when not important in the context. We also consider a partial ordering $<$ on the set $\mathcal{S}$ of sorts of a signature Σ and we write $\mathfrak{s}_1 < \mathfrak{s}_2$ if $\mathcal{T}^{\mathfrak{s}_1}_{\Sigma,\zeta} \subseteq \mathcal{T}^{\mathfrak{s}_2}_{\Sigma,\zeta}$. $\mathcal{P}os(t)$ denotes the set of positions of a term t, $t_{|\omega}$ denotes the subterm of t at position ω, and $t\,[u]_\omega$ the term t with the subterm at position ω replaced by u. The set of variables occurring in a term t is denoted by $\mathcal{V}ar(t)$. If $\mathcal{V}ar(t)$ is empty, t is called a ground term. All the following definitions are given *w.r.t.* to a set ζ whose subset of variables is denoted by $\mathcal{X}$. A substitution is a mapping from $\mathcal{X}$ to $\mathcal{T}_{\Sigma,\zeta}$ which is the identity except over a finite set of variables called domain of σ and denoted by $\mathcal{D}om(\sigma)$. σ naturally extends to an endomorphism of $\mathcal{T}_{\Sigma,\zeta}$. If any variable in the domain is mapped to a ground term then, the corresponding substitution is called ground. A Σ-atom is of the form $p(t_1, \ldots, t_n)$ or $t_1 = t_2$ or $t{:}\mathfrak{s}$ where $t_1, \ldots, t_n, t \in \mathcal{T}_{\Sigma,\zeta}$, $p \in \mathcal{P}$ and $\mathfrak{s} \in \mathcal{S}$. A Σ-literal is either a Σ-atom or a negated (with $\neg$) Σ-atom and the set of Σ-formulae built out of Σ-literals is denoted by $\mathcal{F}or_{\Sigma,\zeta}$. The set of free variables of a formula ϕ (*i.e.* variables not in the scope of a quantifier) is denoted by $\mathcal{FV}ar(\phi)$. A Σ-constraint is either a formula φ or $\mathfrak{Count}(\varphi) \odot n$ where $n \in \mathbb{N}$ and $\odot$ a comparison symbol. A Σ-action is either a Σ-literal without equality symbol or an oriented equality $lhs \hookleftarrow rhs$ with $lhs, rhs \in \mathcal{T}_{\Sigma,\zeta}$. A logical rule over Σ, denoted by $l_1 \wedge \ldots \wedge l_n \Rrightarrow a$, consists of a conjunction of Σ-literals l_i called the body and a Σ-atom a called the goal.

A constrained rewrite rule over a signature Σ is a 3-tuple $(l, \varphi, r) \in \mathcal{T}_{\Sigma,\mathcal{X}} \times \mathcal{F}or_{\Sigma,\mathcal{X}} \times \mathcal{T}_{\Sigma,\mathcal{X}}$, denoted by $l \xrightarrow{\varphi} r$, such that $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \mathcal{FV}ar(\varphi)$. A constrained term rewrite system (CTRS) $\mathcal{R}$ is a set of constrained rewrite rules. We say that $t \in \mathcal{T}_\Sigma$ rewrites into a term $t' \in \mathcal{T}_\Sigma$ with respect to $\mathcal{R}$ and a Σ-theory ϑ, which is denoted by $t \rightarrow^{\vartheta}_{\mathcal{R}} t'$ iff there exist a position $p \in \mathcal{P}os(t)$, a rewrite rule $l \xrightarrow{\varphi} r \in \mathcal{R}$, and a ground substitution σ with $\mathcal{D}om(\sigma) = \mathcal{V}ar(l) \cup \mathcal{FV}ar(\varphi)$ such that $\vartheta \models \{t_{|p} = \sigma(l)\ ;\ t' = t\,[\sigma(r)]_p\ ;\ \sigma(\varphi)\}$.

3 Secured systems

A security policy responds to the authorization requests of a system according to a certain number of rules and to the configuration of the system at the moment of the request. We consider thus that a system constrained by a security policy consists of two parts: on one hand, the set of rules describing the way the decisions are taken and on the other hand, the information used by the rules and the way these evolve in the system. We call the former the policy rules and the latter the security system. In our framework all objects manipulated by the security system and the policy rules are

described as first order terms over a common signature called the security signature. We define the security system using transition rules over environments and the policy rules as a constrained rewrite system.

3.1 Security signature

A transition of the security system is triggered when an authorization request occurs and the result of the respective transition depends on the corresponding decision. We thus call *events* the pairs consisting of an authorization request and the associated decision and the security signature always defines the sorts $\mathcal{Q}uery$ and $\mathcal{D}ecision$ corresponding to the sorts of the first and respectively second element of such a pair.

Definition 1 (Security signature). *A* **security signature** *is a signature $\Sigma_{Sys} \cup \Sigma_{Ev}$ such that Σ_{Ev} contains two sorts $\mathcal{Q}uery$ and $\mathcal{D}ecision$ with $\mathcal{D}ecision < \mathcal{Q}uery$ and a set of function symbols whose co-arity belongs to $\mathcal{S}_{Ev}$.*

Example 1. Along the lines of this paper, we consider an access control system on which we define a confidentiality policy (which can be viewed as a variant of the mandatory part of the Bell and LaPadula policy [5]). This policy constrains accesses done by subjects (S) over objects (O) according to access modes (A) by considering levels of security belonging to a finite lattice $(L, \prec)$ associated with subjects and objects. Hence, we introduce the security signature $\Sigma^{lbp} = \Sigma^{lbp}_{Sys} \cup \Sigma^{lbp}_{Ev}$ as follows. First, $\Sigma^{lbp}_{Sys} = (\mathcal{S}, \mathcal{F}, \mathcal{P})$ consists of $\mathcal{S}^{lbp}_{Sys} = \{S, O, A, L\}$ and

$$\mathcal{P}^{lbp}_{Sys} = \left\{ \begin{array}{l} \prec : L, L \\ m{:}S, O, A \\ sudo{:}S \end{array} \right\} \quad \mathcal{F}^{lbp}_{Sys} = \left\{ \begin{array}{ll} read{:} \mapsto A, & write{:} \mapsto A, \; erase{:} \mapsto A, \\ root{:} \mapsto S, & top{:} \mapsto L, \\ f_s{:}S \mapsto L, & f_o{:}O \mapsto L \end{array} \right\}$$

The functions f_s and f_o describe security levels associated with subjects and objects; $root$ (resp. top) is a particular subject (resp. security level). The predicate m describes current accesses over objects by subjects: $m(s, o, a)$ means that the subject s performs an access of type a over an object o. The predicate $sudo$ describes sudoers, *i.e.* users with root privileges. The signature Σ^{lbp}_{Ev} is based on the following function symbols:

$$\mathcal{F}^{lbp}_{Ev} = \left\{ \begin{array}{ll} ask{:}S, O, A \mapsto \mathcal{Q}uery, & release{:}S, O, A \mapsto \mathcal{Q}uery, deny{:} \mapsto \mathcal{D}ecision \\ create{:}S, L \mapsto \mathcal{Q}uery, & delete{:}S, L \mapsto \mathcal{Q}uery, permit{:} \mapsto \mathcal{D}ecision \end{array} \right\}$$

$ask(s, o, a)$ (resp. $release(s, o, a)$) means that the subject s asks to get (resp. to release) an access over an object o according to the access mode a; $create(s, l)$ means that the subject s asks to create an object o whose security level is l; $delete(s, l)$ means that the subject s asks to delete all objects whose security levels are smaller than l.

3.2 Environments and transition rules

A security system is a transition system that describes the way security information evolve. A state of this system is defined by a set of kernel information, called environment, that can be modified by the transition rules of the system, and an immutable set of closure rules used to compute the complete security information. The result of

such a computation, called in what follows the semantics of the environment, represents an extensional description of the state defined intensionally by the corresponding environment and is obtained by saturation of the environment using the closure rules.

Definition 2 (Environment). *An* **environment** η **over a signature** $\Sigma = \langle \mathcal{S}, \mathcal{F}, \mathcal{P} \rangle$ *consists of:*

(i) *a* **domain***: a finite set* $|\,\eta\,|$ *of sorted constants such that* $\mathcal{C}_\Sigma \subseteq |\,\eta\,|$*;*

(ii) *a* **base of facts***: a finite set* $\mathcal{B}_\eta$ *of atoms of the form* $p(t_1, \ldots, t_n)$ *with* $p \in \mathcal{P}$*,* $n > 0$ *and* $t_1, \ldots, t_n \in |\,\eta\,|$*.*

(iii) *a* **base of equalities***: a finite set* $\mathcal{E}_\eta$ *of equalities of the form* $f(t_1, \ldots, t_n) = t$ *with* $f \in \mathcal{F}$*,* $n > 0$ *and* $t_1, \ldots, t_n, t \in |\,\eta\,|$ *which does not contain two equalities with the same left-hand side;*

(iv) **closure rules***: a set* $\mathcal{R}_\eta$ *of safe and stratified logical rules over* Σ*.*

Safety and stratification of logical rules are well-known notions whose technical definition can be found in [18] ensuring that (i) any variable occurring in a rule has a bounded domain and (ii) negations wrapped inside recursion are forbidden.

The base of equalities gives the interpretation into the domain of the environment for any term of the signature. We denote by $t{\downarrow_\eta}$ the interpretation of the term t in $|\,\eta\,|$, *i.e.* $f(t_1, \ldots, t_n){\downarrow_\eta} = u$ iff $f(u_1, \ldots, u_n) = u \in \mathcal{E}_\eta$ and $u_i = t_i{\downarrow_\eta}$ for all $i \in [1, n]$. If $t = t{\downarrow_\eta}$ we say that t is η-normalized.

Example 2. If we consider the security signature Σ^{lbp} introduced in Example 1, we can define the environment η^{lbp} defined as follows. The domain $\left|\,\eta^{lbp}\,\right|$ contains the constants Bob, $Alice$ and $Charlie$ of sort S, and the constants $Secret$, $Confidential$, L_1, L_2, $Public$, $Sanitized$ of sort L. The base of facts $\mathcal{B}_{\eta^{lbp}}$ (partially) defines the partial order $\prec$ and states that $Charlie$ is a "sudoer":

$$\mathcal{B}_{\eta^{lbp}} = \left\{ \begin{array}{l} Confidential \prec Secret, L_1 \prec Confidential, Public \prec L_2, \\ Sanitized \prec L_1, L_2 \prec Confidential, Sanitized \prec Public, \\ sudo(Charlie) \end{array} \right\}$$

The base of equalities $\mathcal{E}_{\eta^{lbp}}$ provides a definition for the security levels associated with the subjects defined in the domain:

$$\mathcal{E}_{\eta^{lbp}} = \{f_s(root) = top, f_s(Bob) = Secret, f_s(Alice) = L_2, f_s(Charlie) = Public\}$$

The set of closure rules completes the definition of $\prec$:

$$\mathcal{R}_{\eta^{lbp}} = \left\{ \; \Mapsto x \prec x; \;\; x \prec y \wedge y \prec z \Mapsto x \prec z \right\}$$

Due to the restrictions imposed on the domain and on the formulas, we can state:

Proposition 1 (Semantics of an environment). *For any environment* η *over* Σ*, it exists a unique and computable least fixpoint of the logic program consisting of* $\mathcal{B}_\eta$*,* $\mathcal{E}_\eta$ *and* $\mathcal{R}_\eta$*. This fixpoint is denoted by* $[\![\eta]\!]$ *and is called the* **semantics** *of* η*.*

Since $|\,\eta\,|$ is finite, the validity of any first-order formula in $[\![\eta]\!]$ is decidable.

The transition rules of the security system describing the evolution of the states are labelled by the events that trigger them.

Definition 3 (Transition rule). *Given a security signature* $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$*, a* Σ**-transition rule** *consists of a pair of terms* $event \in \mathcal{T}^{\mathcal{Q}uery}_{\Sigma,\mathcal{X}} \times \mathcal{T}^{\mathcal{D}ecision}_{\Sigma,\mathcal{X}}$ *whose subterms*

are constants or variables of Σ_{Sys} together with a sequence of pairs $\langle condition, statement\rangle$ where:

- *$condition$ is a Σ_{Sys}-constraint and*
- *$statement$ is a conjunction of Σ_{Sys}-actions and operations of the form **new** k:$\mathfrak{s}$ or **delete** t with k a constant symbol, $\mathfrak{s} \in \mathcal{S}_{Sys}$ and $t \in \mathcal{T}_{\Sigma_{Sys},\mathcal{X}}$.*

A transition rule is usually written:

$$(event)\ \frac{condition_1}{statement_1};\ \frac{condition_2}{statement_2};\ \ldots;\ \frac{condition_m}{statement_m}$$

The transition rules in a set $\left\{r_i \mid r_i = \left\langle event_i, \langle cond_j, stat_j\rangle_{1\leq j\leq n_i}\right\rangle, 1 \leq i \leq n\right\}$ are **disjoint** *iff for any $i \neq i'$ there exists no substitution σ s.t. $\sigma(event_i) = \sigma(event_{i'})$.*

Intuitively, an operation **new** k:$\mathfrak{s}$ creates a fresh constant of sort $\mathfrak{s}$ called k in the corresponding statement. An operation **delete** t removes t (appropriately instantiated) from the domain of the environment. A $statement$ is "executed" when an $event$ is triggered and the corresponding $condition$ is satisfied. We require the conditions occurring in the same transition rule to be mutually exclusive, *i.e.* in each environment, at most one of them is satisfied. In practice, this is not a strong restriction, since it is always possible to add to the i-th condition the negation of the previous conditions. When executing a statement on an environment, we can add/remove entities to/from its domain, add/remove atoms to/from its base of facts and change the base of equalities in some specific points. The precise semantics of a transition rule is given in Definition 4.

Example 3. If we consider the security signature Σ^{lbp} introduced in Example 1, we can define the following set δ^{lbp} of Σ^{lbp}-transition rules:

$$(ask(s,o,a), permit)\frac{}{m(s,o,a)} \qquad (create(s,l), permit)\frac{\mathfrak{Count}(o{:}O) \leq 10}{\textbf{new}\ k{:}O \wedge f_o(k) \hookleftarrow l}$$

$$(release(s,o,a), permit)\frac{m(s,o,a)}{\neg m(s,o,a)} \quad (delete(s,l), permit)\frac{l' \prec l \wedge f_o(o) = l'}{\textbf{delete}\ o}$$

Intuitively, the rules indicate that when an access request is permitted, the corresponding fact is added and when the respective access is released, the fact is removed. An object can be created only if the number of existing objects has not reached a certain threshold (10 in our case) even if the corresponding request is permitted (note that $o{:}O$ is a formula and $\mathfrak{Count}(o{:}O)$ counts the number of valuations making $o{:}O$ true, that is the number of constants sorted by O which belong to the domain). When a request $delete(s,l)$ is permitted, all objects of a lower level than l are deleted.

Allowing a sequence of pairs $\langle condition, statement\rangle$ in transition rules leads to a more concise description of the operations to be performed for a given event. Notice that using only one pair is strictly less expressive. For example, if we suppose a security signature containing a function symbol $createS : S, L \mapsto Query$ then, specifying that the operations performed at the creation of a subject are different for $root$ needs the use of two separate conditions:

$$(createS(s,l), permit)\ \frac{s = root}{\textbf{new}\ k{:}S \wedge sudo(k) \wedge f_s(k) \hookleftarrow l};\ \frac{}{\textbf{new}\ k{:}S \wedge f_s(k) \hookleftarrow l}$$

Alternatively, several rules with a similar event can be used but our approach allows the factorization with respect to events having the same shape and thus, we get a simpler definition for the application of a set of transition rules, given below.

Definition 4 (Relation induced by a transition rule *w.r.t.* an event). *The transition rule $r = \langle event, \langle condition_i, statement_i\rangle_{1\leq i\leq n}\rangle$ is triggered in a given environment η by the ground (η-normalized) event $evt \in \mathcal{T}^{\mathcal{Q}uery}_{\Sigma,|\,\eta\,|} \times \mathcal{T}^{\mathcal{D}ecision}_{\Sigma,|\,\eta\,|}$ if there exists a ground substitution σ of the variables of $event$ into $|\,\eta\,|$ such that $\sigma(event) = evt$. It induces a relation $\xrightarrow{evt}_r$ over environments such that $\eta \xrightarrow{evt}_r \eta'$ iff, when denoting by $\langle condition, statement\rangle$ the first pair such that $Sol_\eta(\sigma(condition)) = \{\mu \mid [\![\eta]\!] \models \mu(\sigma(condition))\} \neq \varnothing$, η' is obtained by taking initially $\eta' = \eta$ and executing the operations in the statement as follows:*

- *for each **new** k:$\mathfrak{s}$ add to $|\,\eta'\,|$ a fresh constant of sort $\mathfrak{s}$ (i.e. which is not in $|\,\eta'\,|$);*
- *for each $p(t_1, \ldots, t_m)$ add to $\mathcal{B}_{\eta'}$ the fact $\mu(\sigma(p(t_1, \ldots, t_m)))\downarrow_\eta$ for each $\mu \in \mathcal{S}ol_\eta(\sigma(condition))$;*
- *for each $\neg p(t_1, \ldots, t_m)$ remove the fact $\mu(\sigma(p(t_1, \ldots, t_m)))\downarrow_\eta$ from $\mathcal{B}_{\eta'}$ for each $\mu \in \mathcal{S}ol_\eta(\sigma(condition))$;*
- *for each $f(t_1, \ldots, t_m) \hookleftarrow t$ remove $f(\mu(\sigma(t_1))\downarrow_\eta, \ldots, \mu(\sigma(t_m))\downarrow_\eta) = u$ from $\mathcal{E}_{\eta'}$ if it exists and add $f(\mu(\sigma(t_1))\downarrow_\eta, \ldots, \mu(\sigma(t_m))\downarrow_\eta) = \mu(\sigma(t))\downarrow_\eta$ to $\mathcal{E}_{\eta'}$, for each $\mu \in \mathcal{S}ol_\eta(\sigma(condition))$;*
- *for each **delete** t remove $\mu(\sigma(t))\downarrow_\eta$ from $|\,\eta'\,|$ and all facts (from $\mathcal{B}_{\eta'}$) and equalities (from $\mathcal{E}_{\eta'}$) in which occur $\mu(\sigma(t))\downarrow_\eta$, for each $\mu \in \mathcal{S}ol_\eta(\sigma(condition))$.*

We say that η is transformed by r w.r.t. evt into η'. Notice that if there exists no σ s.t. $\sigma(event) = evt$ or no i s.t. $\mathcal{S}ol_\eta(\sigma(condition_i)) \neq \varnothing$ then, there is no η' in relation with η.

Given a set δ of Σ-transition rules, $\xrightarrow{evt}_\delta$ is the relation such that $\eta \xrightarrow{evt}_\delta \eta'$ iff there exists $r \in \delta$ such that $\eta \xrightarrow{evt}_r \eta'$ or $\eta = \eta'$ otherwise.

Although the application of a transition rule yields a unique environment when the **new** operations are performed before any other operation, the result might depend on the strategy used for computing it and, in particular, on the order the different solutions μ are selected and on the order the operations are performed for each solution. If we want to obtain the same result independently of the application strategy, we should guarantee that during the execution of the operations corresponding to the rule: (1) a fact cannot be added and removed; (2) the interpretation of a term does not change; (3) constants of the signature and constants potentially involved in the actions of the rule are not deleted.

There are strong syntactic restrictions constraining the syntax of transition rules that would enforce these conditions. For instance, we can impose that (1) a statement cannot contain p and its negation; (2) a statement cannot contain two oriented equalities whose left-hand sides can be unified and the right-hand sides of all oriented equalities in a statement are necessarily constants or variables of the event of the rule; (3) an operation **delete** t occurs alone in a statement and should not affect the constants of the signature. Indeed, these conditions are too strong, since they limit significantly the expressive power of the formalism.

For example, let us suppose the existence of the symbols $chown{:}S, S, O, A \mapsto \mathcal{Q}uery$, $create{:} \mapsto \mathcal{Q}uery$, and $remove{:}S \mapsto \mathcal{Q}uery$. If the above conditions are imposed on the formation of transition rules then, we could not use the following rule to specify what happens when the subject accessing an object is replaced by another one:

$$(chown(s_1, s_2, o, a), permit) \ \frac{m(s_1, o, a) \wedge s_1 \neq s_2}{\neg m(s_1, o, a) \wedge m(s_2, o, a)}$$

Similarly, it would not be possible to specify that the level of any new created object is the greatest lower bound of the levels of the other existing objects, using the rule

$$(create, permit) \ \frac{\forall o.y \preceq f_o(o) \ \wedge \ \forall l.(\forall o.l \preceq f_o(o) \Rightarrow l \preceq y)}{\textbf{new}\ k{:}O \wedge f_o(k) \hookleftarrow y}$$

where $x \preceq y \stackrel{\triangle}{=} x \prec y \vee x = y$. Moreover, we could not specify that the accesses of removed subjects should be deleted:

$$(remove(s), permit) \ \frac{m(s, o, a)}{\textbf{delete}\ s \wedge \neg m(s, o, a)}$$

We replace thus the above restrictions by the following weaker and decidable assumptions $\mathcal{H}$ on each pair $\langle condition_i, statement_i \rangle$ of a transition rule $r = \langle event, \langle condition_i, statement_i \rangle_{1 \leq i \leq n} \rangle$:

1. a fact cannot be added and removed: for any pair $p(t_1, \ldots, t_n)$, $\neg p(t'_1, \ldots, t'_n)$ in $statement_i$, $t_1, \ldots, t_n, t'_1, \ldots, t'_n$ contain no free variable from $condition_i$ and either t_j and t'_j are two different ground terms or $condition_i$ is a conjunction containing $t_j \neq t'_j$ for some $j \in [1, n]$.
2. the interpretation of a term does not change: $statement_i$ cannot contain two oriented equalities $l \hookleftarrow r$ and $l' \hookleftarrow r'$ such that there exists a substitution σ with $\sigma(l) = \sigma(l')$; the variables of the rhs of oriented equalities in $statement_i$ must be instantiated in a unique way, *i.e.* for any $lhs \hookleftarrow rhs$ in $statement_i$, $condition_i$ is of the form $(\forall \overrightarrow{x} \exists ! \overrightarrow{y} \forall \overrightarrow{z}.\varphi) \wedge \varphi$ with $\overrightarrow{x} = \mathcal{V}ar(event) \cup \mathcal{V}ar(lhs)$, $\overrightarrow{y} = \mathcal{V}ar(rhs) \setminus \overrightarrow{x}$ and $\overrightarrow{z} = (\mathcal{F}\mathcal{V}ar(\varphi) \cup \mathcal{V}ar(statement_i)) \setminus (\overrightarrow{x} \cup \overrightarrow{y})$.
3. constants of the signature are not deleted: for any **delete** t in $statement_i$, $condition_i$ should contain, for every $u \in \mathcal{C}_\Sigma$, the atom $u \neq t$; constants potentially involved in other actions are not deleted: for every **delete** t with t of sort $\mathfrak{s}$ in $statement_i$, $condition_i$ should contain, for every term u of sort $\mathfrak{s}$ occurring in an action of $statement_i$, the atom $u \neq t$.

These assumptions can be easily checked and, if necessary, enforced by a syntactic process suggested by the requirements imposed on the conditions of the rules.

Proposition 2. *For any set δ of disjoint transition rules satisfying $\mathcal{H}$, the relation $\stackrel{evt}{\longrightarrow}_\delta$ is deterministic.*

Proof. We prove that no contradictory operations are performed when a rule is triggered. Assumption (1) guarantees that, if two actions $p(t_1, \ldots, t_n)$ and $\neg p(t'_1, \ldots, t'_n)$

occur in the statement, then there is only one possible instantiation (given by the event) for their variables (which are not free in the condition) and $p(t_1, \ldots, t_n)$ is different from $p(t'_1, \ldots, t'_n)$, since there exists at least one j such that $t_j \neq t'_j$. Assumption (2) guarantees that there is only one possible instantiation for the variables in the right-hand side of an oriented equality and since, additionally, it imposes that the left-hand sides of the different oriented equalities are not unifiable, we always obtain the same interpretation. Assumption (3) obviously guarantees that only constants not affected by the rule can be deleted. The result follows immediately for disjoint rules since at most one rule can be triggered in this case.

3.3 Security systems, policy rules and secured systems

A security system is defined by a set of transition rules and by an initial environment.

Definition 5 (Security system). *Given a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$, a* **system presentation** *over Σ consists of an initial environment η_{init} over Σ_{Sys}, and a set δ of disjoint Σ-transition rules. The semantics of a system presentation $\mathfrak{S} = (\eta_{init}, \delta)$, called a* **security system***, is the labelled transition system $[\![\mathfrak{S}]\!]$ whose states are environments over Σ_{Sys}, whose initial state is η_{init} and whose transitions are $\eta \xrightarrow{evt}_{\delta} \eta'$ for some $evt \in \mathcal{T}^{\mathcal{Q}uery}_{\Sigma,|\,\eta\,|} \times \mathcal{T}^{\mathcal{D}ecision}_{\Sigma,|\,\eta\,|}$.*

Example 4. The security system $\mathfrak{S}^{lbp}$ over the security signature Σ^{lbp} defined in Example 1 consists of the initial environment η^{lbp} defined in Example 2 and the set δ^{lbp} of transition rules defined in Example 3.

Definition 6 (Policy rules). *A set of* **policy rules** *over a security signature Σ is an ordered constrained term rewrite system $\Re$ over $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$ with all the rules of the form $l \xrightarrow{\varphi} r$ with l, r whose sorts are in $\mathcal{S}_{Ev}$ and φ a Σ_{Sys}-constraint.*

We write $q \rightarrow^{\eta}_{\Re} d$ when q is rewritten in one step *w.r.t.* the set of policy rules $\Re$ and the environment η into d and we write $q \xrightarrow{*}{}^{\eta}_{\Re} d$ for multiple steps rewriting.

Example 5. If we consider our example, the following ordered CTRS defines a policy:

$$\left\{\begin{array}{rcl}
ask(s,o,a) & \xrightarrow{sudo(s) \,\wedge\, s \neq root} & ask(root,o,a) \\
ask(s,o,read) & \xrightarrow{f_o(o) \prec f_s(s) \,\wedge\, (\forall o'.m(s,o',write) \Rightarrow f_o(o) \prec f_o(o'))} & permit \\
ask(s,o,write) & \xrightarrow{\forall o'.m(s,o',read) \Rightarrow f_o(o') \prec f_o(o)} & permit \\
ask(s,o,erase) & \xrightarrow{f_o(o) \prec f_s(s)} & permit \\
ask(s,o,a) & \rightarrow & deny \\
release(s,o,a) & \rightarrow & permit \\
create(s,l) & \xrightarrow{s = root \,\vee\, sudo(s) \,\vee\, f_s(s) \prec l} & permit \\
create(s,l) & \rightarrow & deny \\
delete(s,l) & \xrightarrow{sudo(s) \,\wedge\, s \neq root} & delete(root,l) \\
delete(s,l) & \xrightarrow{l \prec f_s(s)} & permit \\
delete(s,l) & \rightarrow & deny
\end{array}\right\}$$

These rules specify that:

- a $sudo$ subject has the same access and deletion rights as $root$;
- a subject can read an object whose level of security is smaller than its level of security if it does not write into an object of a lower security level;
- a subject can write into an object if it does not read an object of a higher level;
- a subject can erase an object whose security level is smaller than its security level;
- a subject can release any of its accesses;
- except for root and sudoers which are authorized to create objects of any security level, a subject can only create objects of security levels higher than its level;
- a subject can delete objects of security levels smaller than its security level;
- in all other cases, the request is denied.

Notice that since the rules are ordered, the constraints do not need to impose explicitly the negation of the constraints of previous overlapping rules and, in particular, no constraint is needed for the "default" rules.

Definition 7 (Consistent and complete policy rules). *A set of policy rules $\Re$ over a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$ is η-***consistent** *(resp. η-***complete***) for an environment η over Σ_{Sys} iff for any query $q \in \mathcal{T}^{\mathcal{Q}uery}_{\Sigma,|\,\eta\,|}$, there exists at most (resp. at least) one decision $d \in \mathcal{T}^{\mathcal{D}ecision}_{\Sigma,|\,\eta\,|}$ such that $q \xrightarrow{*}{}^{\eta}_{\Re} d$.*

These properties can be proved for a large class of policy rules.

Proposition 3. *A set $\Re$ of policy rules is η-consistent if (1) for each rule, its left-hand side contains only one occurrence of each variable and its constraint does not involve terms of sort $\mathcal{Q}uery$; (2) for any two rules $l \xrightarrow{\varphi} r$ and $l' \xrightarrow{\varphi'} r'$ there exists no position ω and no substitution σ such that $[\![\eta]\!] \models \{\sigma(l_{|\omega}) = \sigma(l') \wedge \sigma(\varphi) \wedge \sigma(\varphi')\}$.*

Proof. The proof is obtained by adapting the confluence proof for orthogonal TRS [1].

Proposition 4. *A set $\Re$ of policy rules over a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$ is η-complete if (1) the reduction $\xrightarrow{*}{}^{\eta}_{\Re}$ terminates, (2) for any $f{:}\mathfrak{s}_1, \ldots, \mathfrak{s}_n \mapsto \mathfrak{s} \in \mathcal{F}_{Ev}$ with $\mathfrak{s} > \mathcal{D}ecision$ there exists a default rule for f, i.e. a rule containing no constraint and whose left-hand side is a term $f(x_1, \ldots, x_n)$ with $x_i \in \mathcal{X}$, (3) each rule of $\Re$ is sort-preserving or sort-decreasing (i.e. the sort of its left-hand side is equal or greater than the sort of its right-hand side).*

Proof. We prove that $q \in \mathcal{T}^{\mathcal{Q}uery}_{\Sigma,|\,\eta\,|}$ has a normal form of sort $\mathcal{D}ecision$ by induction on the well-founded rewrite relation induced by $\Re$. If q is of sort $\mathcal{D}ecision$ then its normal form is necessarily of the same sort by (3). If not, then q is reducible by $\Re$ (because of the default rule) into a ground term q' which, by induction hypothesis, has a normal form of sort $\mathcal{D}ecision$.

The classical methods for proving termination of TRS can be adapted for CTRS. For example, the policy rules introduced in Example 5 can be shown terminating using an obvious polynomial interpretation [1] connected to the corresponding constraints. There is also a default rule for each symbol of sort $\mathcal{Q}uery$ and the rules are sort-preserving or sort-decreasing. Consequently, the corresponding normal forms are clearly in this

case $permit$ or $deny$. The policy rules obviously satisfy condition (1) of Proposition 3 and, because of the order, condition (2) as well. The policy rules of Example 5 are thus η-complete and η-consistent for any environment η.

Definition 8 (Secured system). *Given a security signature $\Sigma = \Sigma_{Sys} \cup \Sigma_{Ev}$, a* **presentation of a secured system** *over Σ consists of a system presentation $\mathfrak{S} = (\eta_{init}, \delta)$ over Σ and a set $\Re$ of η-complete and η-consistent policy rules over Σ for any environment η over Σ_{Sys}. The semantics of a secured system presentation $\wp = \langle \mathfrak{S}, \Re \rangle$, called a* **secured system***, is the labelled transition system $[\![\wp]\!]$ whose states are environments over Σ_{Sys}, whose initial state is the initial state of $\mathfrak{S}$ and whose transitions are $\eta \xrightarrow{\langle q,d \rangle}_{\wp} \eta'$ for some $\langle q, d \rangle \in \mathcal{T}^{\mathcal{Q}uery}_{\Sigma,|\eta|} \times \mathcal{T}^{\mathcal{D}ecision}_{\Sigma,|\eta|}$ such that $\eta \xrightarrow{\langle q,d \rangle}_{\delta} \eta'$ and $q \xrightarrow{*}^{\eta}_{\Re} d$.*

Since the set of security rules of a secured system is η-complete and η-consistent for any environment η, the corresponding relation is computable. Moreover, the underlying relation of the security system is computable by construction and thus, so is the underlying relation of the corresponding secured system.

Proposition 5. *The transition relation $\rightarrow_{\wp}$ is computable for any presentation of a secured system $\wp$.*

4 Checking security properties

In this section we propose a methodology based on environment transformations for the validation of security properties enforced by a policy over a system. This is particularly relevant when the security property is expressed on some entities like, for instance, when considering information flow properties, while the policy is an access control policy that can only manipulate objects containing the information to be traced. To solve this problem, we introduce a transformation whose aim is to translate an environment into another one dealing with the entities we are interested in. Such a transformation is defined below by a signature morphism that allows the translation of the domain and of the base of equalities and by a set of transformation rules on the base of facts.

Definition 9 (Signature morphism). *A signature morphism θ from $\Sigma_1 = (\mathcal{S}_1, \mathcal{F}_1, \mathcal{P}_1)$ to $\Sigma_2 = (\mathcal{S}_2, \mathcal{F}_2, \mathcal{P}_2)$ is a pair $(\theta_{\mathcal{S}}, \theta_{\mathcal{F}})$ such that $\theta_{\mathcal{S}}$:$\mathcal{S}_1 \rightarrow \mathcal{S}_2$ and $\theta_{\mathcal{F}}$:$\mathcal{F}_1 \rightarrow \mathcal{F}_2$ are (partial or total) functions such that $\forall f$:$\mathfrak{s}_1, \ldots, \mathfrak{s}_n \mapsto \mathfrak{s} \in \mathcal{D}om(\theta_{\mathcal{F}})$ where $\mathfrak{s}_1, \ldots, \mathfrak{s}_n$, $\mathfrak{s} \in \mathcal{D}om(\theta_{\mathcal{S}})$, $\theta_{\mathcal{F}}(f)$:$\theta_{\mathcal{S}}(\mathfrak{s}_1), \ldots, \theta_{\mathcal{S}}(\mathfrak{s}_n) \mapsto \theta_{\mathcal{S}}(\mathfrak{s}) \in \mathcal{F}_2$. We extend θ to a morphism $\hat{\theta}$ (which is simply denoted by θ) over terms as follows:*

- $\forall x$:$\mathfrak{s} \in \mathcal{X},\ \hat{\theta}(x$:$\mathfrak{s}) = x$:$\theta_{\mathcal{S}}(\mathfrak{s})$
- $\forall f \in \mathcal{D}om(\theta_{\mathcal{F}}),\ \ \hat{\theta}(f(t_1, \ldots, t_n)$:$\mathfrak{s}) = \theta_{\mathcal{F}}(f)(\hat{\theta}(t_1), \ldots, \hat{\theta}(t_n))$

Definition 10 (Environment tranformation). *Given two signatures $\Sigma_1 = (\mathcal{S}_1, \mathcal{F}_1, \mathcal{P}_1)$ and $\Sigma_2 = (\mathcal{S}_2, \mathcal{F}_2, \mathcal{P}_2)$, an environment transformation Θ is a tuple $(\theta, \delta_\Theta, \mathcal{R})$ where:*

- *θ is a signature morphism from Σ_1 to Σ_2;*
- *δ_Θ is a set of pairs $\langle condition, conclusion \rangle$ where $condition$ is a Σ_1-constraint and $conclusion$ is a conjunction of Σ_2-atoms such that x:$\mathfrak{s} \in \mathcal{FV}ar(condition)$ iff $\theta(x$:$\mathfrak{s}) \in \mathcal{FV}ar(conclusion)$ and thus variables of $\mathcal{FV}ar(conclusion)$ (resp. $\mathcal{FV}ar(conclusion)$) are sorted by $\mathcal{D}om(\theta_{\mathcal{S}})$ (resp. $\mathcal{I}m(\theta_{\mathcal{S}})$);*

- *$\mathcal{R}$ is a set of safe and stratified logical rules over Σ_2.*

Due to their similarities with the transitions rules of a security system, the pairs in δ_Θ are sometimes called rules. As one can see in the following definition, the similarities with the transition rules are situated not only at the syntactic level but also at the semantic one.

Definition 11 (Application of an environment transformation). *Let $\Theta = (\theta, \delta_\Theta, \mathcal{R})$ be an environment transformation from Σ_1 to Σ_2, and η be an environment over Σ_1. Applying Θ on η produces an environment $\Theta(\eta)$ over Σ_2 defined as follows:*

- *$|\,\Theta(\eta)\,| = \{c{:}\theta(\mathfrak{s}) \mid c{:}\mathfrak{s} \in |\,\eta\,| \wedge \mathfrak{s} \in Dom(\theta)\}$;*
- *$\mathcal{E}_{\Theta(\eta)}$ contains an equality $\theta(f(t_1, \cdots, t_n)) = \theta(t)$ for each $f(t_1, \cdots, t_n) = t$ in $\mathcal{E}_\eta$ whose image by θ is defined;*
- *$\mathcal{B}_{\Theta(\eta)}$ contains all the Σ_2-atoms $p(\mu(t_1), \ldots, \mu(t_m))\downarrow_{\Theta(\eta)}$ for which there exists a pair $\langle condition, conclusion\rangle \in \delta_\Theta$ where $p(t_1, \cdots, t_n)$ occurs in $conclusion$, and a mapping μ from $\mathcal{V}ar(condition)$ to $|\,\eta\,|$ such that $[\![\eta]\!] \models \mu(condition)$;*
- *$\mathcal{R}_{\Theta(\eta)} = \mathcal{R}$.*

We say that η is transformed by Θ into $\Theta(\eta)$.

Notice that, due to the partial definition of θ, to the conditions to be satisfied by the translated equalities, and to the satisfiability of the condition in η, some of the components (base of equalities, base of facts, closure rules) of $\Theta(\eta)$ may be empty.

Since the rules defining the environment transformation Θ are syntactically much simpler than those of a security system in Section 3, we easily get:

Proposition 6. *Any environment transformation $\Theta = (\theta, \delta_\Theta, \mathcal{R})$ from Σ_1 to Σ_2 induces a total mapping $\eta \mapsto \Theta(\eta)$ from Σ_1-environments into Σ_2-environments.*

This operational view justifies to call Θ a **transformation operator**. Thanks to the notion of environment transformations, it becomes possible to check a security property expressed as a Σ_2-formula ψ over reachable environments of a secured system $[\![\wp]\!]$ over Σ_1. Indeed, this amounts to check that for every reachable environment η of $[\![\wp]\!]$, we have $[\![\Theta(\eta)]\!] \models \psi$, which is decidable, thanks to Proposition 1 and Proposition 6, for any ψ.

Example 6. We consider now environment transformations that can be used to deal with information flow properties of access control policies. First, we introduce the "generic" signature $\Sigma_{FLOW} = (\{Actor, Information\}, \mathcal{F}_{FLOW}, \mathcal{P}_{FLOW})$ where

$$\mathcal{P}_{FLOW} = \left\{ \begin{array}{ll} Get : Actor, Information; & MoveTo : Information, Information; \\ Put : Actor, Information; & Trustworthy : Actor, Information; \\ Eligible : Actor, Information; & Gflow : Information, Information \end{array} \right\}$$

and where $\mathcal{F}_{FLOW}$ is an arbitrary set of function symbols. $Get(a, i)$ means that the actor a knows the information i, $Put(a, i)$ means that the actor a modifies the information i (by using the information it knows), $MoveTo(i_1, i_2)$ means that the information i_2 is enriched with information i_1, $Eligible(a, i)$ means that the actor a is granted to know the information i, $Trustworthy(a, i)$ means that the actor a is granted to modify the information i and $Gflow(i_1, i_2)$ means that the information i_1 is authorized to flow into i_2. The predicates Get, Put and $MoveTo$ are useful for describing existing flows

while the predicates $\mathit{Eligible}$, $\mathit{Trustworthy}$, and Gflow are used to specify flow policies (respectively a confidentiality policy, an integrity policy and a confinement policy). Now, it is possible to define, in a generic way, confidentiality, integrity and confinement security properties as follows:

$$\begin{array}{lll} \text{Confidentiality} & \psi_{conf} & \forall a, i.\, Get(a,i) \Rightarrow \mathit{Eligible}(a,i) \\ \text{Integrity} & \psi_{int} & \forall a, i.\, Put(a,i) \Rightarrow \mathit{Trustworthy}(a,i) \\ \text{Confinement} & \psi_{info} & \forall i, i'.\, MoveTo(i,i') \Rightarrow \mathit{Gflow}(i,i') \end{array}$$

Let us consider the environment transformation defined from the signature Σ^{lbp} of Example 1 and the signature Σ_{FLOW} and consisting of the partial function $\theta_{\mathcal{S}} : \mathcal{S} \rightarrow \mathcal{S}_{FLOW}$ such that $\mathcal{D}om(\theta_{\mathcal{S}}) = \{S, O\}$ with $\theta_{\mathcal{S}}(S) = \mathit{Actor}$ and $\theta_{\mathcal{S}}(O) = Information$ together with the identity function $\theta_{\mathcal{F}}$, the following logical rules over Σ_{FLOW}

$$\mathcal{R}_{FLOW} = \left\{ \begin{array}{l} \Mapsto MoveTo(i,i) \\ Get(a,i) \wedge Put(a,i') \Mapsto MoveTo(i,i') \\ MoveTo(i,i') \wedge Get(a,i') \Mapsto Get(a,i) \\ MoveTo(i,i') \wedge Put(a,i) \Mapsto Put(a,i') \\ MoveTo(i,i') \wedge MoveTo(i',i'') \Mapsto MoveTo(i,i'') \end{array} \right\}$$

and δ_{Θ} defined by

$$\frac{m(x,y,read) \wedge \forall x'.\neg m(x',y,erase)}{Get(x,y)} \qquad \frac{f_o(y) \prec f_s(x)}{\mathit{Eligible}(x,y)}$$

$$\frac{m(x,y,write) \wedge \forall x'.\neg m(x',y,erase)}{Put(x,y)} \qquad \frac{f_o(y) \prec f_o(y')}{\mathit{Gflow}(y,y')}$$

The rules introducing Get and Put allow the translation of the accesses expressed in the source environment using the predicate m into accesses expressed in the target environment using the predicates Get and Put. The rules introducing $\mathit{Eligible}$ and Gflow can be viewed as the definition of the flow interpretation of a (security level-based) access control policy.

Note that this transformation allows the handling of transitive information flows generated by accesses performed simultaneously in a given environment, but does not take into account the past (accesses) of the system. Indirect flows can be nevertheless dealt with, by adding a new predicate which keeps track. in the source environments, of the origins of the information contained into each object of the system.

The above environment transformation provides the means for checking that our policy ensures confinement. This can be done by checking that each reachable environment η of the secured system $[\![\wp]\!]$ is such that $[\![\Theta(\eta)]\!] \models \psi_{info}$. However, the existence of "sudoers" may generate reachable environments that do not satisfy the confidentiality property *w.r.t.* Θ, *i.e.* it is possible to obtain a reachable environment in $[\![\wp]\!]$ which is transformed into an environment which does not satisfy ψ_{conf}. This is for example the case for the environment obtained by considering the initial environment introduced in Example 2 and a transition labelled by the event $create(root, L1)$ (leading to the creation of an object o_1) followed by a transition labelled by the event $ask(Charlie, o_1, read)$. Indeed, since $Charlie$ is a "sudoer", the policy defined in Example 5 allows $Charlie$ to have a read access over o_1 even if its security level is not greater than L_1 and hence, we have $Get(Charlie, o_1)$. However, the meaning

of $Eligible$ specified by δ_Θ does not take into account "sudoers" and we have thus $\neg Eligible(Charlie, o_1)$. Of course, when adding to δ_Θ the rule $\dfrac{sudo(s) \wedge o : O}{Eligible(s, o)}$ which gives a different semantics to the confidentiality property, one can check that any reachable environment η of the system $[\![\wp]\!]$ is such that $[\![\Theta(\eta)]\!] \models \psi_{info} \wedge \psi_{conf}$.

The transformation approach can be also useful when one wants to enforce policies by directly using the desired security properties to constrain the transitions of a security system. Indeed, suppose we want to constrain a security system $\mathfrak{S} = (\eta_{init}, \delta)$ over a signature $\Sigma_1 = \Sigma_{Sys} \cup \Sigma_{Ev}$ in order to ensure a security property expressed as a formula φ over a different signature Σ_2. The corresponding secured system can be obtained by using an environment transformation $\Theta = (\theta, \delta_\Theta, \mathcal{R}_2)$ from Σ_1 to Σ_2 and by considering the transition relation $\rightarrow_{\delta_\varphi}$ such that $\eta \xrightarrow{evt}_{\delta_\varphi} \eta'$ iff $\eta \xrightarrow{evt}_{\delta} \eta' \wedge [\![\Theta(\eta'[\![\models \varphi)$. Of course, such an approach leads to a system whose reachable states satisfy φ iff $[\![\Theta(\eta_{init})]\!] \models \varphi$. The notion of environment transformation makes thus possible the application of a security policy expressed as a (required) property to several systems.

5 Related work

Among a rich literature on security policies (see for instance [9] for policy specification languages), our approach is in the line of logic-based languages providing a well-understood formalism, which is amenable to verification, proof and analysis.

Our formalism borrows inspiration from various sources. Horn clause logic has been used extensively for RBAC models [16]. Since negation and recursion are often needed, the concept of stratified theories has been used for instance in the authorization specification language ASL [13] for access control. Integrity rules specify application dependent conditions that limit the range of acceptable policies. Stratified logic for RBAC policies is also developed in [2]. In our work, we use similar concepts but do not restrict to RBAC models.

Constraint logic programming for designing RBAC and temporal RBAC policies is considered in [3]. Their constraints are conjunctions of equational constraints over sets of constants, and arithmetic constraints over non-negative integers. While keeping a declarative approach, CLP adds the expressive power and efficiency of constraint solving and database querying. A security administrator has then analysis capability thanks to the computation of sets of constraints as answers. Formalisation of security analysis in an abstract setting is done in [15] and exemplified for RBAC. In comparison, we allow a different class of constraints that we keep decidable by restricting to safe theories, and we use constraints in a rewriting context. Note that it is also possible to apply constraint narrowing to get analysis power as in [14].

Whereas most existing works on reasoning about security policies model the environment only lightly, if at all, there are some exceptions. One of the closest works is [10] who represents the behavior of access control policies in a dynamic environment. Policies are written in Datalog and can refer to facts in the authorization state. Events, such as access requests, can change the authorization state, and the changes are specified as

a state machine whose transition labels are guarded by the policy. Security properties can then be analyzed by model checking formulas in first-order temporal logic. In [4], the authors introduce a logic for specifying policies where access requests can have effects on the authorization state. The effects are explicitly specified in the language, an extension of Datalog backed on transaction logic. They also propose a proof system for reasoning about sequences of user actions. In comparison, thanks to constraint rewriting, we provide a more expressive formalism, while keeping it operational and decidable. The full expressive power of constraint rewriting is explored in [7].

Comparing the expressive power of access control models is a fundamental problem in computer security, already addressed in several works. In [6], different access control models are represented in C-Datalog (an object-oriented extension of Datalog) and compared using results from logic programming. In [17], the authors express access control systems as state transitions systems as we do and introduce security-preserving mappings, called reductions, to compare security analysis based on accessibility relations in two different models. In [8, 11], the comparison mechanism is based on a notion of simulation. Thanks to the notion of environment transformation, we address this problem with an operational transition rules based approach.

6 Conclusion and future work

We proposed a framework which provides a common formalism for defining security signatures, environments, systems, and policy rules. We have shown that secured systems specified in this formalism have an operational semantics based on transition and rewriting systems and are thus executable. Our framework allows also the definition of transformations of security signatures and environments and consequently, of secured systems. We defined a transformation operator and showed how it can be used to check security properties over the reachable environments of a secured system. This approach based on a transformation operator allows us to check some properties over a system even if these properties are expressed on a different signature (and/or specification). Our framework facilitates thus the reusability since the same specification of a security property can be used to check several policies and systems. It encourages also the specification of generic security properties dedicated to particular domains like, for example, information flows, and that can be used in different contexts.

As future work, we aim to focus on the extension of the proposed transformation in order to define policies and systems in a completely independent way and to provide thus an enhanced modularity in formal developments. We also want to study how the transformation operators could be used for comparing and composing security policies and systems. Indeed, the comparison between two policies expressed as policy rules $\Re_1$ and $\Re_2$, respectively based on the signatures Σ_1 and Σ_2, is often based on an embedding of Σ_1-formulas into Σ_2-formulas. Such an approach can also be considered for systems, using transformations between environments to define a comparison mechanism. Similarly, for composition, transformation operators could be used to translate policies and systems into policies and systems sharing the same security signature and specification, thus easing the definition of a composition relation.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. S. Barker. Access control for deductive databases by logic programming. In P. J. Stuckey, editor, *Logic Programming, 18th International Conference (ICLP) 2002*, volume 2401 of *LNCS*, pages 54–69. Springer, 2002.
3. S. Barker and P. J. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Transactions on Information and System Security*, 6(4):501–546, 2003.
4. M. Y. Becker and S. Nanz. A logic for state-modifying authorization policies. In *ESORICS 2007*, volume 4734 of *LNCS*, pages 203–218. Springer, 2007.
5. D. Bell and L. LaPadula. Secure Computer Systems: a Mathematical Model. Technical Report MTR-2547 (Vol. II), MITRE Corp., Bedford, MA, May 1973.
6. E. Bertino, B. Catania, E. Ferrari, and P. P. A logical framework for reasoning about access control model. In *ACM Transactions on Information and System Security*, volume 6, pages 71–127, 2003.
7. T. Bourdier and H. Cirstea. Constrained rewriting in recognizable theories. Technical report, INRIA, 2010. http://hal.archives-ouvertes.fr/inria-00456848/en/.
8. A. Chander, J. Mitchell, and D. Dean. A state-transition model of trust management and access control. In *Proceedings of the 14th IEEE Computer Security Foundation Workshop CSFW*, pages 27–43. IEEE Comp. Society Press, 2001.
9. N. C. Damianou, A. K. Bandara, M. S. Sloman, and E. C. Lupu. A survey of policy specification approaches. Technical report, Department of Computing, Imperial College of Scsinece Technology and Medicine, London, UK, 2002. http://www.doc.ic.ac.uk/ mss/Papers/PolicySurvey.pdf.
10. D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *3rd International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *LNCS*, pages 632–646, 2006.
11. L. Habib, M. Jaume, and C. Morisset. Formal definition and comparison of access control models. *Journal of Information Assurance and Security*, 4(4):372–381, 2009.
12. P. Hinman. *Fundamentals of mathematical logic*. A.K. Peters, Ltd., 2005.
13. S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 474–485, New York, NY, USA, 1997. ACM.
14. C. Kirchner, H. Kirchner, and A. Santana de Oliveira. Analysis of rewrite-based access control policies. In D. Dougherty and S. Escobar, editors, *Proceedings of the Third International Workshop on Security and Rewriting Techniques (SecReT 2008), Pittsburgh, PA, USA, 22 June 2008*, volume 234 of *Electronic Notes in Theoretical Computer Science*, pages 55–75. Elsevier, 2009.
15. N. Li and M. V. Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 9(4):391–420, 2006.
16. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29:38–47, 1996.
17. M. V. Tripunitara and N. Li. A theory for comparing the expressive power of access control models. *Journal of Computer Security*, 15(2):231–272, 2007.
18. J. Ullman. *Database and Knowledge - Base Systems*, volume 1: Classical Database Systems. Computer Science Press, 1988.