Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 6 (2011) pp. 357–368

Specification, Analysis and Transformation of Security Policies via Rewriting Techniques

Tony Bourdier

INRIA Nancy & Université Henri Poincaré & LORIA – PAREO Team
BP 101, 54602 Villers-lès-Nancy Cedex, France
Tony.Bourdier@inria.fr

Abstract: **Formal methods for the specification and analysis of security policies have drawn many attention recently. It is now well known that security policies can be represented using rewriting systems. These systems constitute an interesting formalism to prove properties while provides an operational way to evaluate authorization requests. In this paper, we propose to split the expression of security policies in two distinct elements: a *security model* and a *configuration*. The security model (expressed as an equational problem) describes how authorization requests must be evaluated depending on security information. The configuration (expressed as a rewriting system) assigns values to security information. This separation eases the formal analysis of security policies, and makes it possible to automatically convert a given policy to a new security model.**
Keywords: Security policies, formal analysis, algebraic specifications, rewriting systems.

I. Introduction

Security constitutes a crucial concern in modern information systems. Several aspects are involved, such as user authentication (establishing and verifying users' identity), cryptology (changing secrets into unintelligible messages and back to the original secrets after transmission), authorization management (preventing illicit or forbidden operations from users). To define which actions are permitted and denied, a security policy must be established. A security policy describes first the constraints which must be satisfied to allow a given action (the security model) and then the information required for evaluating these constraints (the configuration of the policy).
The first security models appeared in the 70s. Since, the variety of information systems has led to the emergence of multiple models [2], ranging from the simple access control list (ACL) [24] to more elaborated models as role-based access control models (RBAC) [23] or organization-based access control models [17] including lattice-based models [22, 19]. The choice of a model rather than the other one can depend on the structure of the information system, on its features or simply on the sensibility of the administrator. Furthermore, information systems are rarely immutable. They can change to follow the evolution of the organization they support (merger, reorganization, ...) or to improve the information management. In this sorely changing context, maintaining a security policy can lead to the change of the model from which the policy is expressed. The translation of a policy from a model to another one is a very difficult and critical task. Indeed such process, when made "manually", favors error introduction and then security flaws. That is why, to preserve the confidence in a policy after its translation, it is necessary that the transformation is made in a automatic way by a formally verified algorithm.
More generally, is admitted for some years the importance of using formal methods to specify models and security policies. For example, to achieve high levels of certification (EAL[1] 5, 6, 7), it is necessary to provide a formal specification enabling to obtain mechanized formal proofs, to carry out techniques for test generation, or to perform static analyses ensuring required properties. Recently, much work has been done for testing or analysing security policies using formal methods. In [7], the authors proposed to describe some access control models using C-Datalog (an object-oriented extension of Datalog) and to compare them using results from logic programming. In [6], constraint logic programming is used for designing RBAC and temporal RBAC models. In [15], a preorder over security models, based on simulations, is defined and is used to compare some well-known models (Chinese Wall, HRU, BLP and RBAC). A method for generating security tests from the Common Criteria expression of a security policy is presented in [20] while a method for automatic validation of network security policy configurations using SMT solvers is proposed in [25]. More particularly, many works showed how security policies can be represented by using rewriting systems and how rewriting may be used for evaluating authorization requests and for proving some properties. In [14], a first formalization of security policies based on rewriting is proposed for controlling information leakage. Numerous papers proposed to describe classical or new security models as rewriting systems and prove properties over them : ACL and RBAC [4], ASAC [9], DEBAC [5] and an Action Control model [3]. The problem of composing security policies is addressed in [13, 8] with strategic rewriting and a combination of λ-calculus and rewriting respectively. Dynamic rewrite-based security policies enforcement is handled in [12] and a methodology for specifying and implementing security policies using the rewrite-based framework Tom is proposed in [10]. Lastly, a

[1] Evaluation Assurance Level

way to perform "*what-if* queries" over policies specified as rewriting systems is described in [18] in order to increase the trust of the policy author on the behavior of the policy.
In this paper, we introduce a new framework, based on rewriting systems, for specifying security policies. We propose a formalization of policies in two phases: first a model expressed with equational problems and secondly a configuration expressed as a rewriting system. We demonstrate that this approach preserves advantages of "mono-block" security policies specifications by rewriting systems and moreover opens new perspectives. Indeed, we show that such specifications are operational, they allow to show properties and that queries can be performed over them. We also propose an algorithm which transforms a policy from a model to another one. The methods we provide for analysis and transformation of policies are founded on tree automata techniques, a narrowing-based semantic unification algorithm and a recent technique for building regular logical model.
The structure of this paper is as follows. In section II we present notions and notations we use throughout this paper. Section III, devoted to the presentation of our framework, shows how security models and policies must be specified. Section IV presents main decidability results and shows how to analyse a policy in our formalism. Section V is dedicated to the transformation of policies. Finally, the section VI concludes with some perspectives for further work.

II. Background

In this section we present notions and notations used throughout this paper. More details about term algebra and rewriting systems can be found in [1] while more details about tree automata can be found in [11]. Non-standard definitions will be emphasized.

A. Term algebra

We call (many-sorted) *signature* any pair $\Sigma = (\mathscr{S}, \mathscr{F})$ such that $\mathscr{S}$ is a finite non-empty set whose elements are called *sorts* and $\mathscr{F}$ is a finite and non-empty set, disjoint from $\mathscr{S}$, whose elements are called functional *symbols*. Any f of $\mathscr{F}$ is provided together with a non-empty sequence $\langle s_1, \ldots, s_n, s \rangle$ of elements of $\mathscr{S}$, which is denoted by $f : s_1 \times \ldots \times s_n \rightarrowtail s$. $\langle s_1, \ldots, s_n \rangle$ is called the *source sort* of f, s its *target sort* and $s_1 \times \ldots \times s_n \rightarrowtail s$ its *profile*. Any signature Σ is supposed to be finite and partitioned into a set of *constructors* $\Sigma\mathscr{C}ons$ and a set of *defined symbols* $\Sigma\mathscr{D}ef$.
Given a signature $\Sigma = (\mathscr{S}, \mathscr{F})$, a set of *variables* is a countable set denoted by $\mathcal{X}$, disjoint from $\mathscr{F}$, and such that any of its elements x is associated with a sort s of $\mathscr{S}$. The set of *terms* over a signature $\Sigma = (\mathscr{S}, \mathscr{F})$ and a set of variables $\mathcal{X}$ of sort $s \in \mathscr{S}$ is the smallest set, denoted by $\mathcal{T}(\Sigma, \mathcal{X})^s$, such that any variable $x \in \mathcal{X}$ of sort s belongs to $\mathcal{T}(\Sigma, \mathcal{X})^s$ and for any $f : s_1 \times \ldots \times s_n \rightarrowtail s \in \mathscr{F}$ and $t_1 \in \mathcal{T}(\Sigma, \mathcal{X})^{s_1}, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})^{s_n}$, $f(t_1, \ldots, t_n)$ belongs to $\mathcal{T}(\Sigma, \mathcal{X})^s$. The set of terms over Σ and $\mathcal{X}$ is the set $\mathcal{T}(\Sigma, \mathcal{X}) = \bigcup_{s \in \mathscr{S}} \mathcal{T}(\Sigma, \mathcal{X})^s$. If $\mathcal{X}$ is empty, then $\mathcal{T}(\Sigma, \mathcal{X})$ is written $\mathcal{T}(\Sigma)$ and its elements are called *ground* terms. A term of $\mathcal{T}(\Sigma\mathscr{C}ons, \mathcal{X})$ is called a *constructor-term* and a term of $\mathcal{T}(\Sigma\mathscr{C}ons)$ is called a *data-term*.

Definition 1. *A sort s is* Σ-safe *iff $\mathcal{T}(\Sigma\mathscr{C}ons)$ is finite.*

Let t be a term over Σ and $\mathcal{X}$. $\mathcal{V}ar(t)$ denotes the variables occurring in t. If any variable of t occurs only once in t, then t is said *linear*. A term containing exactly one variable is called a *context* and is denoted by C where $C[t]$ denotes the term obtained by replacing in C the unique variable with t. A *position* within t is a sequence ω of integers describing the path from the root of t (seen as a finite labeled tree) to the root of the subterm at that position, denoted by $t_{|\omega}$. We use ε for the empty sequence. $\mathcal{P}os(t)$ denotes the set of positions of t. $t(\omega)$ is the symbol of t at position ω and $t\,[s]_\omega$ the term t with the *subterm* at position ω replaced by s. A *substitution* σ is a mapping from $\mathcal{X}$ to $\mathcal{T}(\Sigma, \mathcal{X})$ which is the identity except over a finite set of variables $\mathcal{D}om(\sigma)$, called the domain of σ, extended to an endomorphism of $\mathcal{T}(\Sigma, \mathcal{X})$. A substitution σ is often denoted by $\{x \mapsto \sigma(x) \mid x \in \mathcal{D}om(\sigma)\}$ and is said *ground* if all the variables of its domain are mapped to ground terms. For conciseness and readability reasons, we denote by $\vec{s}$ the tuple consisting of $s_1 \times \ldots \times s_n$ or $\langle s_1, \ldots, s_n \rangle$. Tuples of terms will sometimes be considered as particular terms whose head is the special symbol $\#$ (*e.g.* the tuple $\langle t_1, \ldots, t_n \rangle$ will be denoted by $\vec{t}$ and sometimes seen as the algebraic term $\#(\vec{t})$). If t_i is of sort s_i, then $\vec{t}$ (or $\#(\vec{t})$) is of sort $\vec{s}$.

B. First order logic

In order to simplify notations, predicate symbols are seen as functional symbols whose target sort is $\mathcal{B}ool$ (then, any signature is supposed to contain the sort $\mathcal{B}ool$ and $\Sigma\mathscr{C}ons^{\mathcal{B}ool} = \{true, false\}$) and thus, an *atom* is an equality over terms of sort $\mathcal{B}ool$ and $true$ or $false$. A *formula* is either an atom, an *equality* over terms of a same sort $s \neq \mathcal{B}ool$ or one of the following expressions: $\exists x : \varphi$, $\forall x : \varphi$, $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$, $\neg\varphi$ where φ and φ' are formulae ($\Rightarrow, \Leftrightarrow, \ldots$ and equalities over terms of sort $\mathcal{B}ool$ are shortcuts). *Free* and *bound* variables of a formula φ are denoted by $\mathcal{FV}ar(\varphi)$ and $\mathcal{BV}ar(\varphi)$, respectively. An *interpretation* $\Im$ of a signature $\Sigma = (\mathscr{S}, \mathscr{F})$ *w.r.t.* to a set $D = \bigcup_{s \in \mathscr{S}} D^s$ consists of a family of applications $\Im(f)$ from $D^{s_1} \times \ldots \times D^{s_n}$ to D^s associated to each symbol $f : s_1 \times \ldots \times s_n \rightarrowtail s \neq \mathcal{B}ool$ of Σ, and a family of relations $\Im(p)$ over $D^{s_1} \times \ldots \times D^{s_n}$ for each symbol $p : s_1 \times \ldots \times s_n \rightarrowtail \mathcal{B}ool$ of $\Sigma \setminus \{true, false\}$. An $\Im$-*valuation* is a map ν from variables to D. To each term of $\mathcal{T}(\Sigma, \mathcal{X})$ we associate a value $\Im(t)(\nu)$ as follows: for any variable x, $\Im(x)(\nu)$ is $\nu(x)$ and for any symbol f of Σ, $\Im(f(t_1, \ldots, t_n))(\nu)$ is $\Im(f)(\Im(t_1)(\nu), \ldots, \Im(t_n)(\nu))$. The semantics of a formula φ in $\Im$ *w.r.t.* the valuation ν is denoted by $\Im(\varphi)(\nu)$ and is defined as follows:

- $\Im(p(t))(\nu)$ is true iff $\Im(t)(\nu)$ belongs to $\Im(p)$,
- $\Im(t = t')(\nu)$ is true iff $\Im(t)(\nu)$ is equal to $\Im(t')(\nu)$,
- $\Im(\exists x : \varphi)(\nu)$ is true iff there exists a ν' such that $\mathcal{D}om(\nu') = \{x\}$ and $\Im(\varphi)(\nu \cup \nu')$ is true,
- $\Im(\forall x : \varphi)(\nu)$ is true iff for all ν' such that $\mathcal{D}om(\nu') = \{x\}$, $\Im(\varphi)(\nu \cup \nu')$ is true,
- $\Im(\neg\varphi)(\nu)$ is true iff $\Im(\varphi)(\nu)$ is false,
- $\Im(\varphi \wedge \varphi')(\nu)$ is true iff $\Im(\varphi)(\nu)$ and $\Im(\varphi')(\nu)$ are true, and
- $\Im(\varphi \vee \varphi')(\nu)$ is true iff either $\Im(\varphi)(\nu)$ or $\Im(\varphi')(\nu)$ is true.

We say that φ holds in $\Im$ and we write $\Im \models \varphi$ iff $\Im(\varphi)(\nu)$ is true for any valuation ν. Given a formula φ over Σ, a (logical) model of φ is a Σ-interpretation $\Im$ such that $\Im \models \varphi$. An interpretation (or a model) *w.r.t.* D is said *finite* iff D is finite.

Definition 2. *Given a signature Σ, a* Σ-interpretation *is an interpretation $\Im$ of Σ w.r.t. $\mathcal{T}(\Sigma\mathscr{C}ons)$ such that for any $f \in \Sigma\mathscr{C}ons$, $\Im(f) : (t_1, \ldots, t_n) \mapsto f(t_1, \ldots, t_n)$.*

C. Tree automata

We call *tree automaton* any quadruple $\mathcal{A} = \langle Q, \Sigma, F, \rightarrow_A \rangle$ such that Σ is a signature, Q is a finite set of *states*, F is a subset of Q whose elements are called *final states* and $\rightarrow_A$ is a set of *transitions* of the form $f(q_1, \ldots, q_n) \rightarrow_A q$ where $q_1, \ldots, q_n, q$ are in Q and f is a symbol of Σ. A tree automaton is said *deterministic* iff all its transitions have a different left-hand side and it is said *complete* iff for any symbol f and states $q_1, \ldots, q_n$, $f(q_1, \ldots, q_n)$ is the left-hand side of at least one transition. Without loss of generality, we can consider that all automata are deterministic and complete. $\rightarrow_A$ is extended to a relation $\rightarrow_A^*$ as follows: if for any i in $\{1, \ldots, n\}$, $t_i \rightarrow_A^* q_i$ and $f(q_1, \ldots, q_n) \rightarrow_A q$, then $f(t_1, \ldots, t_n) \rightarrow_A^* q$. We call set of terms *recognized* by a tree automaton $\mathcal{A} = \langle Q, \Sigma, F, \rightarrow_A \rangle$ the set of ground terms $t \in \mathcal{T}(\Sigma)$ such that there is a final state $q \in F$ such as $t \rightarrow_A^* q$. We use the same notations to represent an automaton and the set it denotes. A set of terms is said *regular* iff it is recognized by a tree automaton. A set of n-tuples (or equivalently a n-ary relation) of ground terms E is said regular iff the set of terms $\#(\vec{t})$ such that $\vec{t} \in E$ is regular. It is said *n-regular* iff the set of terms $t_1 \otimes \ldots \otimes t_n$ such that $\vec{t} \in E$ is recognized by an automaton were $t = t_1 \otimes \ldots \otimes t_n$ is the term over $(\Sigma \cup \{\Lambda\})^n$ such that: $\forall \omega \in \bigcup_{i=1}^{n} \mathcal{P}os(t_i)$, $t(\omega) = \langle t_1[\omega), \ldots, t_n[\omega) \rangle$ where $u[\omega) = u(\omega)$ if $\omega \in \mathcal{P}os(t)$ and Λ otherwise. If all terms of a regular set E have the same head (root) symbol, we denote it by $E_{|\varepsilon}$. If t is a linear term, $\mathcal{R}ec(t)$ denotes the automaton recognizing ground instances of t. For any regular sets $\mathcal{A}$ and $\mathcal{A}'$, $\mathcal{A} \cap \mathcal{A}'$, $\mathcal{A} \cup \mathcal{A}'$, $\mathcal{A} \setminus \mathcal{A}'$ are regular. For any sort s, $\mathcal{T}(\Sigma)^s$ is also regular. If $\mathcal{A}$ is a regular set and C a context, the sets $\{C[t] \in \mathcal{T}(\Sigma) \mid t \in \mathcal{A}\}$ and $\{t \in \mathcal{T}(\Sigma) \mid C[t] \in \mathcal{A}\}$ are regular. They are respectively denoted by $C[\mathcal{A}]$ and $C^{-1}[\mathcal{A}]$.

D. Rewriting systems and equational problems

A *rewrite rule* is an ordered pair of terms denoted by $lhs \rightarrow rhs$ such that $\mathcal{V}ar(rhs) \subseteq \mathcal{V}ar(lhs)$. The terms lhs and rhs are respectively called the *left-hand side* and the *right-hand side* of the rule. A *(term) rewriting system* (or TRS) is a finite (in this paper) set of rewrite rules. Given a rewriting system $\mathcal{R}$, a term t *rewrites* to a term t', which is denoted by $t \rightarrow_{\mathcal{R}} t'$ iff there exists a rule $lhs \rightarrow rhs$ in $\mathcal{R}$, a position ω in t and a substitution σ satisfying $t_{|\omega} = \sigma(lhs)$ such that $t' = t\,[\sigma(rhs)]_\omega$. We say that t matches lhs at the position ω. If a term matches no left-hand side of a rewriting system $\mathcal{R}$, then it is said to be *irreducible* for $\mathcal{R}$ or in $\mathcal{R}$-normal form. The set of ground terms of Σ in $\mathcal{R}$-normal form is denoted by $NF(\mathcal{R})$. $\rightarrow_{\mathcal{R}}^*$ denotes the reflexive transitive closure of $\rightarrow_{\mathcal{R}}$. If $t \rightarrow_{\mathcal{R}}^* t'$ and t' is irreducible, then t' is called a $\mathcal{R}$-normal form of t. $\mathcal{R}$ is *confluent* iff for any terms t, t_1 and t_2 such that $t \rightarrow_{\mathcal{R}}^* t_1$ and $t \rightarrow_{\mathcal{R}}^* t_2$, there exists a term t' such that $t_1 \rightarrow_{\mathcal{R}}^* t'$ and $t_2 \rightarrow_{\mathcal{R}}^* t'$. $\mathcal{R}$ is *terminating* iff there is no map α from $\mathbb{N}$ to terms such that $\alpha(i) \rightarrow_{\mathcal{R}} \alpha(i+1)$ for any $i \in \mathbb{N}$. $\mathcal{R}$ is *convergent* if it is confluent and terminating. In that case, any term t has exactly one $\mathcal{R}$-normal form which is denoted by $t{\downarrow_{\mathcal{R}}}$.

Definition 3 (Constrained data rewrite system). *A constrained data rewrite system (CD-TRS) over a signature Σ is a pair $\langle \mathcal{R}, \mathbb{A} \rangle$, simply denoted by $\mathcal{R}$, such that: $\mathbb{A}$ is a set of regular sets of Σ data-terms, $\mathcal{R}$ is a set of triples called rules denoted by $f(\vec{s}) \rightarrow r \parallel \varphi$ where f is a defined symbol, s_i are Σ constructor terms, r is a Σ data-term and φ is a conjunction of membership constraints of the form $t \in \mathcal{A}$ where t is a Σ constructor term and $\mathcal{A}$ belongs to $\mathbb{A}$.*

To make specifications of CD-TRS easier to write, we allow use of a "default rule" for each defined symbol f of the form $f(\_, \ldots, \_) \rightarrow r$ which means that if $f(t_1, \ldots, t_n)$ matches no rule, then it rewrites to r[2]. Any CD-TRS $\mathcal{R}$ induces the following relation $\rightarrow_{\mathcal{R}}$ over $\mathcal{T}(\Sigma)$: $t \rightarrow_{\mathcal{R}} t'$ iff it exists a rule $l \rightarrow r \parallel \varphi$ of $\mathcal{R}$, a position ω and a ground substitution σ such that $\sigma(l) = t_{|\omega}$, $t' = t\,[\sigma(r)]_\omega$ and $\sigma(\varphi)$ holds.

Definition 4 (CD-interpretation). *A Σ-interpretation $\Im$ is called a CD-interpretation iff it exists a convergent CD-TRS $\mathcal{R}$ such that $NF(\mathcal{R}) = \mathcal{T}(\Sigma\mathscr{C}ons)$, and for any ground term t of sort $s \neq \mathcal{B}ool$, $\Im(t) = t{\downarrow_{\mathcal{R}}}$ and for any atom at, $at{\downarrow_R}$ is $true$ if at holds in $\Im$ and $false$ otherwise. By an abuse of language, we say that $\mathcal{R}$ is a CD-interpretation.*

A *constrained equational problem* over Σ is an expression P of the form $\exists \vec{x} : E, C$ where E is a conjunction of equations over $\mathcal{T}(\Sigma, \mathcal{X})$ and C a conjunction of membership constraints $t \in A$ where $t \in \mathcal{T}(\Sigma, \mathcal{X})$ and A is a regular subset of $\mathcal{T}(\Sigma\mathscr{C}ons)$. If C is empty, then P is an *equational problem*. We sometimes write $P[\vec{x}]$ to denote that $\vec{x} = \mathcal{FV}ar(P)$. A (constrained) equational problem P is Σ-safe iff for any equality $t = t'$ in P, t (and t') are of a Σ-safe sort.

III. Specification of models and policies

In this section, we show how to declare a security policy by means of equational problems and a constrained data rewriting system. To allow the reuse of a security policy specification, we naturally dissociate it into two parts. The most generic part of a policy is called model. It consists of the specification of the security requirements, in other words the constraints on values of some security information which must be satisfied to grant an action. For example, a model based on security levels could impose that a user can read a file if its security level is greater than the one of the file. However, the model gives no information concerning the security levels in question. It is the subject of what we call in this paper the configuration of the policy. The configuration consists of the definition of the evaluation of functions

[2] Of course any linear CD-TRS with default rules is equivalent to a linear CD-TRS without ones and conversely.

and predicates used in the security model. For example, in a model based on security levels, the configuration of a policy consists in assigning the security levels to users and files. To define the vocabulary describing the data (subjects, objects, authorizations, ...) required for the definition of security policies (the domain of discourse), we introduce the notion of security signature.

Definition 5 (Security signature). *A security signature is a many-sorted signature* $\mathbf{\Sigma} = (\mathscr{S} \cup \{\mathcal{A}ction\}, \mathscr{F} \cup \mathscr{F}^{\mathcal{A}ction})$ *such that the only target sort of the symbols of* $\mathscr{F}^{\mathcal{A}ction} \neq \varnothing$ *is* $\mathcal{A}ction$ *and such that no source sort contains* $\mathcal{A}ction$*. We denote by* Σ *the signature* $(\mathscr{S}, \mathscr{F})$*. Moreover, we require that* $\Sigma\mathscr{C}ons = \Sigma$ *(and then* $\Sigma\mathscr{D}ef = \varnothing$*).*

The signature $\mathbf{\Sigma}$ describes the security data which do not depend on the policy we want to define. It specifies for example the users and the files but it does not specify, for example, the security roles of the users which are specific to the implementation of role-based policies. Terms of sort $\mathcal{A}ction$ correspond to actions that can be performed in the system. Terms generated by Σ are only data-terms, what means that two elements of $\mathcal{T}(\Sigma)$ cannot represent the same element.

Example 1. *Consider a system containing three users (e.g.* $Alice$*,* Bob *and* $Charlie$*) and a countable set of files such that each file is identified by a natural number. Assume that actions which can be performed on the system are reading and writing (a file by a user). Let* $\mathscr{S}$ *be the set of sorts* $\mathcal{S}ubject$*,* $\mathcal{O}bject$*,* $\mathcal{N}at$*,* $\mathscr{F}$ *be the following set of symbols:*

$$\begin{cases} Alice, Bob, Charlie & : & \rightarrowtail \mathcal{S}ubject \\ file & : \mathcal{N}at & \rightarrowtail \mathcal{O}bject \\ zero & : & \rightarrowtail \mathcal{N}at \\ succ & : \mathcal{N}at & \rightarrowtail \mathcal{N}at \end{cases}$$

and $\mathscr{F}^{\mathcal{A}ction}$ *be composed by* $read$ *and* $write$ *of profile* $\mathcal{S}ubject \times \mathcal{O}bject \rightarrowtail \mathcal{A}ction$*. Then* $\mathbf{\Sigma} = (\mathscr{S} \cup \{\mathcal{A}ction\}, \mathscr{F} \cup \mathscr{F}^{\mathcal{A}ction})$ *is a security signature specifying the data described above. For example,* $read(Alice, file(succ(zero)))$ *describes the action consisting of the reading of the file identified by the number* 1 *by* $Alice$*.*

Security models specify mechanisms to be implemented within the system in order to guarantee the desired security properties (confidentiality, integrity, ...). In our formalism, a security model describes the structures which are specific to it, the security rules associating authorizations to constraints and the shape of "acceptable" configurations.

Definition 6 (Security model). *A security model* $\mathfrak{M}$ *over a security signature* $\mathbf{\Sigma} = (\mathscr{S} \cup \{\mathcal{A}ction\}, \mathscr{F} \cup \mathscr{F}^{\mathcal{A}ction})$ *consists of:*

- *an extension of* Σ *denoted by* $\Sigma_{\mathfrak{M}} = (\mathscr{S} \cup \mathscr{S}_{\mathfrak{M}}, \mathscr{F} \cup \mathscr{F}_{\mathfrak{M}})$ *such that:*
  - (i) *for any* $f : s_1 \times \ldots \times s_n \rightarrow s \in \Sigma\mathscr{C}ons_{\mathfrak{M}}$*,* $s \in \mathscr{S}_{\mathfrak{M}}$*,*
  - (ii) *target sorts of defined symbols of* $\Sigma\mathscr{D}ef_{\mathfrak{M}}$ *are* $\Sigma_{\mathfrak{M}}$*-safe,*
- *a set of security rules* $\Gamma_{\mathfrak{M}}$ *of the form* $action \mapsto P[\vec{x}]$ *where* $action$ *is a linear term of* $\mathcal{T}(\mathbf{\Sigma}, \mathcal{X})^{\mathcal{A}ction}$ *and* P *is a* $\Sigma_{\mathfrak{M}}$*-safe equational problem such as* $\vec{x} = \mathcal{V}ar(action)$*. The term* $action$ *is called the pattern of the rule and* P *is called the constraint of the rule.*
- *a first order theory* $\mathscr{T}_{\mathfrak{M}}$ *over* $\Sigma_{\mathfrak{M}}$ *in which any variables is either of a* $\Sigma_{\mathfrak{M}}$*-safe sort or occurs at most once in each equality, and*
- *a finite set of profiles* $\mathscr{P}_{\mathfrak{M}}$

A security model can require security information which are specific to it. It is the aim of the signature $\Sigma_{\mathfrak{M}}$. For example, a model based on security levels must indicate that some levels and an order over these levels are required. The first condition on the signature indicates that we cannot define new constructor symbols whose target sort belongs to the security signature (*e.g.* we cannot define new users or new files). The second condition indicates that functions always take their values in a finite set. Security rules associate with every possible action a security constraint which must be satisfied so that the action is permitted. As explained previously, the semantics of the security information specific to the policy is not known at the level of the definition of the model but can be constrained by means of a theory. For example the model can define a functional symbol inf which must be interpreted by a partial order, in which case, $\mathscr{T}_{\mathfrak{M}}$ should include $\forall x, y : inf(x, y) = true \wedge inf(y, z) = true \Rightarrow inf(x, z) = true$, $\forall x, y : inf(x, y) = true \wedge x \neq y \Rightarrow inf(y, z) = false$, ... The model can also not include all the security information and specify which type of information the policy can define. For example a model with security levels can either fix the set of levels in the model, or let the policy define its own levels. In the former case, if security levels are represented by terms of sort $\mathcal{L}evel$, then $\mathscr{P}_{\mathfrak{M}} = \{\rightarrowtail \mathcal{L}evel\}$.

Example 2. *In this example we show how to specify the RBAC model [23] in our framework in the context of a system in which reading and writing files are the only actions. We do not claim that this is the only way to formalize the RBAC security model. Let us recall in simple terms the main idea of RBAC. First, each user is assigned to a set of roles. Next, access modes (read or write) on resources (files in our case) are also assigned to roles. Finally, a user may perform an action on a resource if and only if it is assigned to a role to which the corresponding access mode is assigned. Let* $\mathbf{\Sigma}$ *be the security signature defined in the previous example. As previously said, security information include roles and access modes. Then,* Σ_{RBAC} *extends* Σ *with sorts* $\mathcal{R}ole$ *and* $\mathcal{M}ode$*. We consider that roles are not fixed at the level of the model but access modes are. Thus,* Σ_{RBAC} *contains the constants* R *and* W *of target sort* $\mathcal{M}ode$ *which respectively correspond to the reading and writing mode. Since the (finite) set of roles must be defined at the level of the policy, then* $\mathscr{P}_{\text{RBAC}} = \{\rightarrowtail \mathcal{R}ole\}$ *(if we allowed an infinite set of roles, we could define* $\mathscr{P}_{\text{RBAC}}$ *with* $\mathcal{N}at \rightarrowtail \mathcal{R}ole$ *instead). We use the function* $ura : \mathcal{S}ubject \times \mathcal{R}ole \rightarrowtail \mathcal{B}ool$ *to represent the assignment of roles to users (*ura *for user-role assignment) and the function* $pra : \mathcal{R}ole \times \mathcal{M}ode \rightarrowtail \mathcal{B}ool$ *to represent the assignment of access modes to roles (*pra *for privilege-role assignment). Note that* ura *and* pra *are defined symbols while* R *and* W *are constructors. Security*

rules are simply defined by formally expressing that a user s can read (resp. write) a file o when it has a role r (that is $ura(s,r) = true$) to which the R *mode (resp.* W *mode) over o is assigned (that is $pra(r, \mathrm{R}, o)$ and $pra(r, \mathrm{W}, o)$ respectively). Thus, Γ_{RBAC} consists of the following rules:*

$$read(s,o) \mapsto \exists r : \{\ ura(s,r) = true\ ;\ pra(r,\mathrm{R},o) = true\ \}$$
$$write(s,o) \mapsto \exists r : \{\ ura(s,r) = true\ ;\ pra(r,\mathrm{W},o) = true\ \}$$

In the following, we will denote RBAC *the security model defined by $(\Sigma_{\text{RBAC}}, \varnothing, \mathscr{P}_{\text{RBAC}}, \Gamma_{\text{RBAC}})$.*

One can notice that security models defined in this manner do not induce a unique way to evaluate action authorizations. For example, which semantics one must give to a model containing two rules $read(s,o) \mapsto \varphi_1$ and $read(s,o) \mapsto \varphi_2$? To define a unique semantics for a set of security rules, we provide for models a strategy of interpretation.

Definition 7 (Interpretation strategy). *An interpretation strategy indicates the way in which action authorizations must be evaluated depending on the security rules of the model. In other terms, it associates with any action (data-term of sort $\mathcal{A}ction$) an equational problem whose satisfaction computation corresponds to the evaluation of the action authorization. We define the following strategies:*

- `and`*, which permits an action iff all the constraints associated with a pattern matching the action are satisfied:*
$$\Gamma^{and}_{\mathfrak{M}} : ac \mapsto \bigwedge_{\substack{a \mapsto \varphi \in \Gamma_{\mathfrak{M}} \\ \sigma(a) = ac}} \sigma(\varphi)$$
- `or`*: which permits an action iff at least one constraint associated with a pattern matching the action is satisfied:*
$$\Gamma^{or}_{\mathfrak{M}} : ac \mapsto \bigvee_{\substack{a \mapsto \varphi \in \Gamma_{\mathfrak{M}} \\ \sigma(a) = ac}} \sigma(\varphi)$$
- `or_else`$^{\nu}$*, which permits an action iff the constraint associated with the first pattern matching the action is satisfied and permits (if $\nu = \top$) or denies (if $\nu = \bot$) actions which match no pattern:*
$$\Gamma^{or\_else^{\nu}}_{\mathfrak{M}} : ac \mapsto \begin{cases} \sigma(\varphi_i) \text{ if } \sigma(a_i) = ac \\ \qquad \text{and } \nexists \sigma' : \sigma'(a_j) = ac \\ \qquad\qquad \text{for any } j < i \\ \nu \text{ otherwise} \end{cases}$$

where $\nu \in \{\top, \bot\}$ and where rules of $\Gamma_{\mathfrak{M}}$ are ordered. Note that by definition, the default authorization ($i.e.$ when an action matches no pattern) for the `and` *interpretation is $\top$ while it is $\bot$ for* `or`*.*

We suppose in what follows that models are fitted with an interpretation strategy which will be denoted as an exponent of the model, except when all strategies coincide ($i.e.$ when any action matches exactly one pattern, which is the case in the previous example because patterns do not overlap).

In order to completely define a security policy, one must indicate the model on which it is based, the data which are specific to it as well as the evaluation process of functions and predicates described in the security model.

Definition 8 (Security policy). *A security policy $\wp$ over a security signature $\mathbf{\Sigma}$ consists of:*

- *a security model (fitted with a strategy if needed) $\mathfrak{M}^{\zeta}$ over $\mathbf{\Sigma}$,*
- *a finite set of constructors κ whose profiles are $s_1 \times \ldots \times s_n \rightarrowtail s \in \mathscr{P}_{\mathfrak{M}}$. This set extends $\Sigma_{\mathfrak{M}}$ to another signature denoted by $\Sigma^{\kappa}_{\mathfrak{M}}$ and*
- *a linear CD-rewriting system ρ over $\Sigma^{\kappa}_{\mathfrak{M}}$ called configuration.*

A policy composed by $\mathfrak{M}^{\zeta}$, κ and ρ is denoted by $\langle\, \mathfrak{M}^{\zeta}, \rho^{\kappa} \,\rangle$.

Example 3. *Consider the model* RBAC *described in the previous example.* RBAC *specified that any role-based policy should define its own roles. Let us define two roles r_1 and r_2 (and then define κ with $\{r_1, r_2 : \ \rightarrowtail \mathcal{R}ole\}$). Consider that r_1 is assigned to $Alice$ and $Charlie$ while r_2 is assigned to Bob and $Charlie$. Finally, assume that r_1 has the reading privilege over files identified by an even number while r_2 allows accesses in writing mode to files identified by a number which is a multiple of three. We obtain the following configuration:*

$$\rho = \begin{cases} ura(Alice, r_1) & \to_{\rho} true & & \\ ura(Bob, r_2) & \to_{\rho} true & & \\ ura(Charlie, r_1) & \to_{\rho} true & & \\ ura(Charlie, r_2) & \to_{\rho} true & & \\ pra(r_1, \mathrm{R}, file(n)) & \to_{\rho} true & \| & n \in \mathcal{M}_2 \\ pra(r_2, \mathrm{W}, file(n)) & \to_{\rho} true & \| & n \in \mathcal{M}_3 \\ ura(\_, \_, \_) & \to_{\rho} false & & \\ pra(\_, \_, \_) & \to_{\rho} false & & \end{cases}$$

where $\mathcal{M}_2$ is the automaton recognizing natural number which are a multiple of 2, $i.e.$:

$$\langle\, \Sigma\mathscr{C}ons\,, \{q_1, q_2\}, \{q_2\}, \to_{\delta} \,\rangle \ \text{ with } \begin{cases} zero & \to_{\delta} q_2 \\ succ(q_2) & \to_{\delta} q_1 \\ succ(q_1) & \to_{\delta} q_2 \end{cases}$$

and $\mathcal{M}_3$ is the automaton recognizing natural number which are a multiple of 3, $i.e.$ $\langle\, \Sigma\mathscr{C}ons\,, \{q_1, q_2, q_3\}, \{q_3\}, \to_{\delta'} \,\rangle$ with

$$\begin{cases} zero & \to_{\delta'} q_3 \\ succ(q_3) & \to_{\delta'} q_1 \\ succ(q_2) & \to_{\delta'} q_3 \\ succ(q_1) & \to_{\delta'} q_2 \end{cases}.$$

IV. Semantics and properties

In this section, we show that our framework allows to show properties over policy specifications. Moreover, we show that it provides an operational way to evaluate if an action is permitted or denied (the semantics of policies). Finally, we prove that we can perform "queries" over security policies.
So that a security policy is "admissible" and can be interpreted in a unique way, it is necessary that it satisfies certain number of properties. For example, it must not be possible that the evaluation of functions and predicates produces several results. Furthermore, the policy configuration must be "acceptable" for the model on which it is based at the risk of giving to the security model a different meaning that its author attributed to it, so altering the security property which it has to guarantee.

Definition 9 (Consistency). *A security policy* $\wp = \langle\, \mathfrak{M}^\zeta, \rho^\kappa \,\rangle$ *is consistent if*

- ρ *is a CD-interpretation of* $\Sigma_\mathfrak{M} \cup \{\kappa\}$ *and*
- $\rho \models \mathscr{T}_\mathfrak{M}$.

Proposition 1. *Consistency is decidable.*

Proof. *Let* $\langle\, \mathfrak{M}^\zeta, \rho^\kappa \,\rangle$ *be a security policy. The first condition (*ρ *is a CD-interpretation of* $\Sigma_\mathfrak{M} \cup \{\kappa\}$*) is equivalent to the convergence of* ρ *and its sufficient completeness w.r.t. each defined symbol* $f \in \Sigma\mathscr{D}ef^\kappa_\mathfrak{M}$*. By definition, the rewriting system* ρ *is composed by rules whose right hand-side is a data-term and whose constraint contains only irreducible terms. Straightforwardly, such a system terminates. All head symbols of left hand-side are defined symbols, thus any constructor term is irreducible. The sufficient completeness w.r.t. a defined symbol* f *is ensured either by a default rule or by checking that the set of all (constructor) ground instances of the constrained left hand-side (which is regular) covers all well-formed data-term whose head is* f *(which is a regular set). Finally,* ρ *is confluent iff two rules whose left hand-sides overlap have the same right-hand side (which is decidable). Thus, the first condition of consistency can be verified by the following procedure:*

- *for any rule* $r = f(\vec{s}) \to rhs \parallel \varphi$*, we denote by* $A(r)$ *the tree automaton recognizing the set of data-terms* $\{\sigma(\#(\vec{s})) \mid \sigma(\varphi)$ *holds*$\}$*, which can always be built since* lhs *is linear and* φ *contains only constructor terms.*
- *for any rule* $r = f(\vec{s}) \to rhs \| \varphi$ *and* $r' = f(\vec{s'}) \to rhs' \| \varphi'$ *of* ρ*, check that* $A(r) \cap A(r') \neq \varnothing \Rightarrow rhs = rhs'$
- *check that* $\bigcup_{r=f(\vec{s})\to r \parallel \varphi} A(r) = \{\#(\vec{t}) \mid t_i \in \mathcal{T}(\Sigma\mathscr{C}ons)^{s_i}, f : s_1 \times \ldots \times s_n \to s\}$

Second, in a CD-interpretation, interpretations of predicates and function symbols can be represented with regular sets. We can evaluate any linear atom $p(C[\vec{x}])$ *(and non linear atoms whose variables occurring several times belong to a finite set) as the regular set recognizing data-terms tuples* $\#(\vec{t})$ *such that* $p(C[\vec{t}])$ *holds (*$f(\vec{t}) = u$ *being seen as the atom* $f(\vec{t}, u)$*). Operators over tree automata (projection, cylindrification, context adding or deleting, union and intersection) allow to evaluate any combination of such atoms.* □

We will see that our framework provides an operational way to compute authorizations. More precisely, we can compute a (convergent) rewrite system $\to_\wp$ from any consistent security policy $\wp$ such that for any $action \in \mathcal{T}(\mathbf{\Sigma})^{\mathcal{A}ction}$, $action \downarrow_\wp = permit$ if $\wp$ permits $action$ and $action \downarrow_\wp = deny$ otherwise. First, let us give the definition of the semantics of a consistent security policy.

Definition 10 (Semantics). *Let be* $\wp = \langle\, \mathfrak{M}^\zeta, \rho^\kappa \,\rangle$ *a consistent security policy. We call semantics of* $\wp$ *and we denote by* $[\![\, \wp \,]\!]$ *the set of actions that* $\wp$ *permits, that is:*

$$[\![\, \wp \,]\!] = \left\{ ac \in \mathcal{T}(\mathbf{\Sigma})^{\mathcal{A}ction} \mid \rho \models \Gamma^\zeta_\mathfrak{M}(ac) \right\}$$

This induces the following equivalence relation: $\wp \approx \wp'$ *iff* $[\![\, \wp \,]\!] = [\![\, \wp' \,]\!]$.

Example 4. *Let* $\wp$ *be the policy defined in the previous example.* $[\![\, \wp \,]\!]$ *contains:*

- $read(Alice, f(2k))$,
- $write(Bob, f(3k))$,
- $read(Charlie, f(2k))$, $write(Charlie, f(3k))$

for any $k \in \mathbb{N}$.

Proposition 2. *For any consistent security policy* $\wp$*, the set* $[\![\, \wp \,]\!]$ *is decidable, that is to say there exists a terminating algorithm deciding if an action belongs to* $[\![\, \wp \,]\!]$*. Moreover, the set* $[\![\, \wp \,]\!]$ *is regular, that is there exists a tree automaton* $\langle\, Q, \mathbf{\Sigma}, F, \to_\wp \,\rangle$ *with* $permit, deny \in Q$ *and* $F = \{permit\}$ *such that* $action \in [\![\, \wp \,]\!]$ *iff* $action \downarrow_\wp = permit$ *(and* $action \downarrow_\wp = deny$ *otherwise).*

Proof. *The construction of* $\to_\wp$ *is based on the procedure of CD-unification depicted in Figure 1. CD-unification consists in computing the solutions of an equational problem w.r.t. a CD-interpretation, that is by interpreting the equality symbol as the equality induced by a CD-rewriting system. The proposed algorithm is an extension of the standard narrowing-based unification algorithm described in [16]. First, let us prove that our transformation algorithm preserves the set of solutions by analysing the rules introduced in our version:*

- *utilize 2 and empty: for any formula* ϕ*, it is obvious that* $\phi \wedge \exists x : x \in A$ *is equivalent to* ϕ *if* A *is not empty and that* $\phi \wedge x \in A$ *has no solution if* A *is empty.*
- *merge: straightforward.*
- *propagate: since membership constraints contain only automata recognizing data-terms, the replacement of variables by terms in a membership constraint must be performed only for terms containing no defined symbol.*
- *deconstruct, search: given a tree automaton* $A = (Q, \Sigma, F, \delta)$*, if* f *is a constructor symbol, then* $f(\vec{s}) \in A$ *is equivalent to* $\bigvee_{q\in F} \bigvee_{f(q_1,\ldots,q_n)\to q\in\delta} \bigwedge s_i \in A[q_i]$ *where* $A[q_i]$ *is the (reduced) automaton obtained from* A *by replacing the set of final states with* $\{q_i\}$*. If* f *is a defined symbol,* $f(\vec{s})$ *must be "solved" by the narrowing process. Thus,* $f(\vec{s}) \in A$ *is replaced by* $\exists x : x = f(\vec{s}) \wedge x \in A$*. When* x *will be assigned with a constructor term* t*,* $x \in A$ *will be rewritten into* $t \in A$ *(propagate) and then the deconstruction rule will be applied.*

Second, let us prove the termination of the algorithm. We define the size of a problem $P = \exists x : E, C$ *as the tuple* $\|P\| = \langle\, n_1, \ldots, n_9 \,\rangle \in \mathbb{N}^9$ *where:*

- n_1 *is the cardinality of* $Solved(P) \cap \mathcal{V}ar(C)$*, where* $Solved(P)$ *is the set of variables such that* E *contains an equality of the form* $x = t$ *with* t *a constructor term,*
- n_2 *is the number of defined symbols occurring in* C
- n_3 *is the number of defined symbols occurring in* E
- n_4 *is the number of constructor symbols occurring in* C
- n_5 *is the number of bound variables*
- n_6 *is the cardinality of* $\mathcal{V}ar(P) \setminus Solved(P)$
- n_7 *is the number of constructor symbols occurring in* E
- n_8 *is the number of equations in* E
- n_9 *is the number of constraints in* C

We do not study the size modification through rules Empty and Fails since these rules are normalizing. The following table indicates how the size of a problem evolves depending

Utilize 1	$\exists x : E, C$	$\rightsquigarrow_\rho$	E, C	if $x \notin \mathcal{V}ar(E) \cup \mathcal{V}ar(C)$
Utilize 2	$\exists x : E, C \cup \{x \in A\}$	$\rightsquigarrow_\rho$	E, C	if $x \notin \mathcal{V}ar(E) \cup \mathcal{V}ar(C)$ and $\mathcal{L}(A) \neq \varnothing$
Eliminate	$\exists x : E \cup \{x = t\}, C$	$\rightsquigarrow_\rho$	E, C	if $x \notin \mathcal{V}ar(E) \cup \mathcal{V}ar(C) \cup \mathcal{V}ar(t)$
Empty	$E, C \cup \{x \in A\}$	$\rightsquigarrow_\rho$	$\bot$	if $\mathcal{L}(A) = \varnothing$
Alias	$\exists z : E \cup \{x = z\}, C$	$\rightsquigarrow_\rho$	$E_{\{z \mapsto x\}}, C_{\{z \mapsto x\}}$	if $x \in \mathcal{X}$
Delete	$E \cup \{s = s\}, C$	$\rightsquigarrow_\rho$	E, C	
Merge	$E, C \cup \{x \in A, x \in B\}$	$\rightsquigarrow_\rho$	$E, C \cup \{x \in A \cap B\}$	
Narrow	$E \cup \{s = t\}, C$	$\rightsquigarrow_\rho$	$\exists \vec{z} : E \cup \{u = t\} \cup \theta, C \cup \varphi$	if $\theta = mgu(s_{\|p}, l), u = \theta(s\,[r]_p)$ and $l \to r \parallel \varphi$ is a renamed rule of ρ and $\vec{z} = \mathcal{V}ar(\theta) \setminus \mathcal{V}ar(s)$
Deconstruct	$E, C \cup \{f(\vec{s}) \in A\}$	$\rightsquigarrow_\rho$	$E, C \cup \{s_i \in A_i\}_{i=1\ldots n}$	if $A = \langle\, Q, \Sigma\mathscr{C}ons, F, \to_\delta \,\rangle$ and $f(\vec{q}) \to_\delta q_F$ with $q_F \in F$ and $A_i = \langle\, Q, \Sigma\mathscr{C}ons, \{q_i\}, \to_\delta \,\rangle$
Search	$E, C \cup \{f(\vec{s}) \in A\}$	$\rightsquigarrow_\rho$	$\exists x : E \cup \{x = f(\vec{s})\}, C \cup \{x \in A\}$	if f defined symbol
Decompose	$E \cup \{f(\vec{x}) = f(\vec{y})\}$	$\rightsquigarrow_\rho$	$E \cup \{x_i = y_i\}$	
Fails 1	$E \cup \{f(\vec{x}) = g(\vec{y})\}$	$\rightsquigarrow_\rho$	$\bot$	if $f \neq g$
Fails 2	$E \cup \{x = t\}$	$\rightsquigarrow_\rho$	$\bot$	if $x \in \mathcal{V}ar(t)$
Propagate 1	$E \cup \{x = t\}, C$	$\rightsquigarrow_\rho$	$E_{\{x \mapsto t\}} \cup \{x = t\}, C$	if $x \in \mathcal{V}ar(E)$
Propagate 2	$E \cup \{x = t\}, C$	$\rightsquigarrow_\rho$	$E \cup \{x = t\}, C_{\{x \mapsto t\}}$	if $x \in \mathcal{V}ar(C)$ and t constructor

Figure. 1: *CD-Unification*: narrowing-based procedure for CD-semantic unification

on the other rules:

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
Utilize 1					$-$				
Utilize 2					$-$				
Eliminate					$-$				
Alias					$-$				
Delete								$-$	
Merge									$-$
Narrow			$-$	$+$	$+$	$+$	$+$		$+$
Deconstruct				$-$					
Search		$-$	$+$		$+$	$+$	$+$		
Decomposition							$-$		$+$
Propagate 1						$-$	$+$		
Propagate 2	$-$			$+$					

where $+$ means "may increase", $-$ means "always strictly decrease" and when no sign is indicated, it means that the corresponding size is preserved. Since for any rule the sign "$+$" is always preceded by a sign "$-$", it means that for any P, if $P \rightsquigarrow_\rho P'$ then $\|P'\| <_{\mathbb{N}^9} \|P\|$. Since $<_{\mathbb{N}^9}$ is well-founded (assuming that $\langle\, n_1, \ldots, n_9 \,\rangle <_{\mathbb{N}^9} \langle\, n'_1, \ldots, n'_9 \,\rangle$ iff there is a j such that $n_i = n'_i$ for any $i < j$ and $n_j < n'_j$), we can conclude that $\rightsquigarrow_\rho$ terminates. To formally prove this result, we must prove the table presented above. Most of values of this table can be easily checked. Let us see the most difficult results, namely the value of n_1 for the rule Propagate 1 and n_3 for the rule Narrow. Applying the rule Propagate 1 replaces the "solved" variable x with a term t. One must check that t contains no solved variable. First, t can not contain x since this case is handled by a prior rule (Fails 2). Secondly, if t contains other solved variables, thus Propagate 1 is applicable and Propagate 1 have priority. Let us now study the value of n_3 for the narrowing rule. By definition, rules of ρ are of the form $f(c_1, \ldots, c_n) \to d$ with c_i constructors terms and d a data-term. Then, the narrowing process replace a subterm of s containing at least one defined symbol (f) with a data-term. Thus, The number of defined symbols occurring in E strictly decrease. Finally, let us prove that the set of solutions of a safe equational problem is a regular set. First, it is easy to check that normal forms of $\rightsquigarrow_\rho$ are problems of the form $\exists \vec{z}{:}\{\vec{y}_E = \vec{t}\}, \{\vec{z}_C \in \vec{A}, \vec{y}_C \in \vec{B}\}$ where $\vec{y} = \vec{y}_E \cup \vec{y}_C$, $\vec{z} = \vec{z}_E \cup \vec{z}_C$ and $\vec{t}$ are constructor terms such that $\vec{z} \subseteq \mathcal{V}ar(\vec{t}) \subseteq \vec{y} \cup \vec{z}$. Two cases are possible:

- *$\mathcal{V}ar(\vec{t}) \cap \vec{y} = \varnothing$. Thus, the set of solutions is clearly regular.*
- *$\mathcal{V}ar(\vec{t}) \cap \vec{y} \neq \varnothing$. One must prove that if $y_i = C[y_j]$ occurs in E, then y_i and y_j are of a safe sort, i.e. can be assigned only with a finite set of possible values. The following cases are possible:*
  - *the equality $y_i = y_j$ belonged to the original problem. Thus, by definition, the sort of y_i and y_j is safe.*
  - *the original problem contained $C[y_i] = C'[y_j]$ with $C_{|\varepsilon}$ or $C'_{|\varepsilon}$ constructor. This case is not possible since no safe sort contains constructors which are not constants.*
  - *the original problem contained $C[y_i] = C'[y_j]$ with $C_{|\varepsilon}$ and $C'_{|\varepsilon}$ defined. To obtain an equality of the form $y_i = C''[y_j]$, at least one step of narrowing occurred. Then, $C[y_i] = C'[y_j]$ is rewritten to $t = C'[y_j] \wedge \ldots$ where t is a data-term. Thus, this case is also impossible.*

  In conclusion, if two free variables of a safe problem occurs in the same equality in its normal form, then these variables are of a safe sort and then can be assigned to a finite set of values. Then, even in this case, the set of solutions is regular.

Given a convergent CD-rewriting system, we denote by $\rightsquigarrow_\rho$ the rewriting relation induced by applying the rules of the figure 1 in the order in which they appear and under the innermost strategy (note that only rules Narrow and Deconstruct can create divergences in the reduction tree). If ρ is the configuration of a security policy and $P = \exists \vec{x}{:}E, C$ is a constrained equational problem in normal form w.r.t. $\rightsquigarrow_\rho$ where $\vec{y} = \mathcal{F}\mathcal{V}ar(P)$, then it is necessary of the shape $\exists \vec{z}{:}\{\vec{y}_E = \vec{t}\}, \{\vec{z}_C \in \vec{A}, \vec{y}_C \in \vec{B}\}$ with $\vec{y} = \vec{y}_E \cup \vec{y}_C$, $\vec{z} = \vec{z}_E \cup \vec{z}_C$ and $\vec{t}$ are constructor terms of the security signature such that $\vec{z} \subseteq \mathcal{V}ar(\vec{t}) \subseteq \vec{y} \cup \vec{z}$. The set of solu-

tions of this problem can be represented by a tree automaton denoted by $\mathcal{S}ol(P)$ which recognizes data-terms of the form $\#(t_1, \ldots, t_n)$ such that $\{y_j \mapsto t_j\}_{j=1\ldots n}$ is a solution. For any constrained equational problem P, we denote by $P\downarrow_\rho$ the set of normal forms of P w.r.t. $\rightsquigarrow_\rho$. Thus, we define $\mathcal{S}ol_\rho(P) = \bigcup_{P' \in P\downarrow_\rho} \mathcal{S}ol(P')$. We extend the definition of $\mathcal{S}ol_\rho$ to disjunctions of problems as follows: $\mathcal{S}ol_\rho(\{P_1 \vee \ldots \vee P_n\}) = \bigcup_{i=1}^n \mathcal{S}ol_\rho(P_i)$. The CD-Narrow procedure is terminating, correct and complete for safe equational problems in that sense where any ground substitution $\sigma = \{y_j \mapsto t_j\}$ is a solution of a safe equational problem P w.r.t. a CD-rewriting system ρ (with $\vec{y} = \mathcal{FV}ar(P)$), i.e. $\rho \models \sigma(P)$ iff $\#(\vec{t}) \in \mathcal{S}ol_\rho(P)$.

To build $\rightarrow_\wp$, we need, together with the CD-unification procedure, to compute the function $\Gamma^\zeta_{\mathfrak{M}}$ introduced in the previous section. We refine the definition 7 by regrouping data-terms associated with the same security constraint (up to a change of variables). Thus, we redefine $\Gamma^\zeta_{\mathfrak{M}}$ as follows:

- *if ζ is* `and` *or* `or`*, for any $I \subseteq [1, n]$ such that $\mathcal{R}ec_I \neq \varnothing$, $\Gamma^\zeta_{\mathfrak{M}}$ associates with $\mathcal{R}ec_I$ either $\bigwedge_{i\in I} \tilde{P}_i[\vec{x}]$ if $\zeta =$* `and` *or $\bigvee_{i\in I} \tilde{P}_i[\vec{x}]$ if $\zeta =$* `or` *and associates either $\top$ if $\zeta =$* `and` *or $\bot$ if $\zeta =$* `and` *with $\mathcal{T}(\mathbf{\Sigma})^{\mathcal{A}ction} \setminus \bigcup_{i=1}^n \mathcal{R}ec(action_i)$.*
- *if ζ is* `or_else`$^\nu$*, $\Gamma^\zeta_{\mathfrak{M}}$ associates $\tilde{P}_i[\vec{x}]$ with $\mathcal{R}ec_i^{fst}$ and ν with $\mathcal{T}(\mathbf{\Sigma})^{\mathcal{A}ction} \setminus \bigcup_{i=1}^n \mathcal{R}ec(action_i)$.*

knowing that:

- $\Gamma_{\mathfrak{M}} = \{action_i \mapsto t_i\}_{1=1\ldots n}$
- $\mathcal{R}ec_I = \bigcap_{i\in I} \mathcal{R}ec(action_i) \cap \bigcup_{i\in[1,n]\setminus I} \overline{\mathcal{R}ec(action_i)}$
- $\mathcal{R}ec_i^{fst} = \mathcal{R}ec(action_i) \cap \bigcap_{j<i} \overline{\mathcal{R}ec(action_j)}$
- *$\tilde{P}_i[\vec{x}]$ is $\exists \vec{y} : \vec{x} = \vec{t} \wedge P_i[\vec{y}]$, $action_i = f(\vec{t}), f \in \mathscr{F}^{\mathcal{A}ction}$, $\vec{y} = \mathcal{V}ar(\vec{t})$, $\vec{x}$ are fresh variables (they are the new variables of the problem).*

$\Gamma^\zeta_{\mathfrak{M}}$ must be understood as follows: given an action (a data-term of sort $\mathcal{A}ction$) $ac = f(\vec{t})$ and the regular set $\mathcal{L}$ such as $\mathcal{L} \in \mathcal{D}om(\Gamma^\zeta_{\mathfrak{M}})$ and $ac \in \mathcal{L}$ (which exists and is unique because the domain of $\Gamma^\zeta_{\mathfrak{M}}$ constitutes a partition of the whole set of data-terms of sort $\mathcal{A}ction$), then ac is permitted iff $\vec{t}$ is a solution of $\Gamma^\zeta_{\mathfrak{M}}(\mathcal{L})$. Thus, the automaton recognizing $[\![\, \wp \,]\!]$ is $\bigcup_{\mathcal{L}\in\mathcal{D}om(\Gamma^\zeta_{\mathfrak{M}})} \left(\mathcal{L} \cap \mathcal{S}ol^f_\rho(\Gamma^\zeta_{\mathfrak{M}}(\mathcal{L}))\right)$ where $f = \mathcal{L}_{|\varepsilon}$ and $\mathcal{S}ol^f_\rho(P)$ recognizes $\{f(\vec{t}) \mid \#(\vec{t}) \in \mathcal{S}ol_\rho(P)\}$. □

As a consequence of proposition 2, we obtain the capacity of "querying" the policy. Query analysis provides a way to ask questions of the form "Is there at least one user which can access the file f" or "Can any user read or write at least a file ?". In the literature, this is sometimes referred to as "what-if analysis" or as "administrator queries". In [18], it is shown how to solve queries defined as terms of sort $\mathcal{A}ction$ containing variables. In this paper, we consider a larger class of queries.

Definition 11 (Query). *Given a policy $\wp$ over $\mathbf{\Sigma}$, a query over $\wp$ is a first order formula built with the only predicate $\downarrow_\wp$ and constructor terms of $\mathbf{\Sigma}$.*

Example 5. *Consider the policy $\wp$ defined in the previous example. The following formula is a query over $\wp$: $Q(n) \triangleq \forall s : (read(s, file(n)) \downarrow_\wp permit) \Rightarrow (write(s, file(succ(n))) \downarrow_\wp deny)$. The solution of this query is the set of valuations of n such that $Q(n)$ holds.*

Proposition 3. *Given a security policy $\wp$, the set of solutions of any query over $\wp$ is decidable. Moreover, it is a regular set.*

Proof. *This follows directly from the fact that $\downarrow_\wp$ is the transition relation of a tree automaton. The solution of a query is obtained by using tree automata operators (boolean operators, adding and deleting contexts and projection).* □

V. Policy transformation

To allow the maintainability of a security policy, to simplify its management, to make less complex its understanding or to increase the degree of granularity of authorizations, we can be led up to change the model in which the policy is expressed. The translation of a policy from a model towards another one is a very difficult and dangerous task. This process is error-prone and thus can introduce security faults. We propose in this section an automatic method for performing this transformation. This process could also be called "auto-configuration" of a policy. The proposed method is based on a recent algorithm of construction of logical models described by n-regular tree automata [21].

Formally, our goal is the following: Given $\wp_1 = \langle\, \mathfrak{M}_1^{\zeta_1}, \rho_1^{\kappa_1} \,\rangle$ a security policy over $\mathbf{\Sigma}$ together with a security model $\mathfrak{M}_2^{\zeta_2}$ and a signature extension κ_2, we want to compute a configuration ρ_2 such that $\langle\, \mathfrak{M}_1^{\zeta_1}, \rho_1^{\kappa_1} \,\rangle \approx \langle\, \mathfrak{M}_2^{\zeta_2}, \rho_2^{\kappa_2} \,\rangle$. We note this problem under the form of an equation: $\langle\, \mathfrak{M}_2^{\zeta_2}, X^{\kappa_2} \,\rangle \approx \langle\, \mathfrak{M}_1^{\zeta_1}, \rho_1^{\kappa_1} \,\rangle$ where X is the variable of the equation.

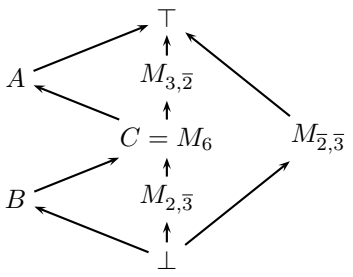
Example 6. *Let us now consider another security model* LBAC *based on a security lattice. The idea consists in assigning to each subject and each file a security level and to order levels as a lattice. Thus, Σ_{LBAC} extends Σ with the sort $\mathcal{L}evel$ and three functional symbols : $f_S : \mathcal{S}ubject \rightarrowtail \mathcal{L}evel$, to represent the assignment of levels to subjects, $f_O : \mathcal{O}bject \rightarrowtail \mathcal{L}evel$, to represent the assignment of levels to objects (files) and $inf : \mathcal{L}evel \times \mathcal{L}evel \rightarrowtail \mathcal{B}ool$ to represent an order over levels. Since we want inf to describe a lattice, we add the following axioms in $\mathscr{T}_{\text{LBAC}}$:*

$$\begin{cases} \forall x : inf(x,x) = true \\ \forall x,y : inf(x,y) = true \wedge inf(y,z) = true \\ \qquad\qquad \Rightarrow inf(x,z) = true \\ \forall x,y : inf(x,y) = true \wedge x \neq y \Rightarrow inf(y,x) = false \\ \exists x_{max}, x_{min}, \forall x, inf(x, x_{max}) = true \\ \qquad\qquad \wedge inf(x_{min}, x = true) \end{cases}$$

The LBAC *model we consider applies the well-known principles of "no read up" and "no write down", which consists in defining Γ_{LBAC} by the following two rules:*

$$\begin{array}{rcl} read(s,o) & \mapsto & \{\, inf(f_O(o), f_S(s)) = true \,\} \\ write(s,o) & \mapsto & \{\, inf(f_S(s), f_O(o)) = true \,\} \end{array}$$

Finally, the model give no information about the levels, then $\mathscr{P}_{\text{LBAC}} = \{\rightarrowtail \mathcal{L}evel\}$. "Transforming the policy $\wp = \langle\, \text{RBAC}, \rho^\kappa \,\rangle$ toward the model LBAC *" consists in finding a CD-rewrite system which interprets the symbols inf, f_O*



$$\begin{cases} f_S(Alice) \rightarrow A \\ f_S(Bob) \rightarrow B \\ f_S(Charlie) \rightarrow C \\ f_O(file(n)) \rightarrow C \quad \| \ n \in \mathcal{M}_6 \\ f_O(file(n)) \rightarrow M_{3,\overline{2}} \quad \| \ n \in \mathcal{M}_3 \setminus \mathcal{M}_2 \\ \vdots \\ inf(\bot, B) \rightarrow true \end{cases}$$

Figure. 2: Illustration of the Example 6

and f_S so that together with the model LBAC *we obtain exactly the same set of authorizations as $\langle$ RBAC, $\rho^{\kappa}\rangle$. In other terms, it consists in assigning security levels to subjects and objects (f_O et f_S) and to order these levels (inf). A solution, if κ contains eight constants of sort $\mathcal{L}evel$, is depicted in the Figure 2. where A (resp. B, resp. C) is the level of $Alice$ (resp. Bob, resp. $Charlie$), arrows between levels denote the order over them (inf) and M_n is the level assigned to files identified by a number which is (resp. is not) a multiple of n if n is not (resp. is) overlined.*

To build the target policy, we generate a set of constraints describing that the target policy must have the same semantics as the source policy. Pointing out that a subset of A-interpretation is isomorphic with the set of CD-interpretations, we propose to take advantage of a recent work on automated building of A-models (in a logical sense) [21]. A precise definition of A-interpretations (and then A-models) will be given later but roughly speaking, it consists of interpretations in which predicates are interpreted as n-regular relations. To be as clear as possible, we split the description of our algorithm into two main steps. The first concerns the generation of constraints and the second concerns the construction of the target configuration from these constraints.

Constraints generation We must describe that the semantics of the source policy is equivalent to the semantics of the target one. Since semantics of security policies are regular, we need a way to express regular sets as first order axioms. More precisely, given a tree automaton $A = \langle Q, \Sigma, F, \delta\rangle$, we denote by $Ax(A)[x]$ the first order formula such that for any ground term t, $Ax(A)[t]$ is a theorem if $t \in A$ and is contradictory otherwise. In the same way, we denote by $Ax(A)^{-h}[\vec{x}]$ the formula such that for any tuple of ground terms $\vec{t}$, $Ax(A)[\vec{t}]$ is a theorem if $h(\vec{t}) \in A$ and is contradictory otherwise. Let us give a formal definition of these formulae (note that we consider that all automata are complete, minimalized and deterministic). $Ax(A)[x]$ is the following formula:

$$\bigvee_{q_F \in F} q_F(x) \wedge \bigwedge_{q \notin F} \neg q(x)$$
$$\wedge \bigwedge_{f(q_1,\ldots,q_n)\rightarrow_\delta q} \left(\forall \vec{y} : \bigwedge_{i=1}^{n} q_i(y_i) \Rightarrow q(f(\vec{y}))\right)$$

and $Ax^{-h}(A)[\vec{x}]$ is the conjunction of the two following formulae:

$$\left(\bigvee_{h(q_1,\ldots,q_n)\rightarrow q_F \in F} \left(\bigwedge_{i=1}^{n} q_i(x_i)\right)\right)$$
$$\wedge \left(\bigvee_{h(q_1,\ldots,q_n)\rightarrow q \notin F} \neg \left(\bigwedge_{i=1}^{n} q_i(x_i)\right)\right)$$

$$\textit{and} \bigwedge_{f(q_1,\ldots,q_n)\rightarrow_\delta q \notin F} \left(\forall \vec{y} : \bigwedge_{i=1}^{n} q_i(y_i) \Rightarrow q(f(\vec{y})\right)$$

Let us now consider a consistent security policy $\langle \mathfrak{M}_1^{\zeta_1}, \rho_1^{\kappa_1}\rangle$ over $\mathbf{\Sigma}$, together with a security model $\mathfrak{M}_2^{\zeta_2}$ over $\mathbf{\Sigma}$ and an extension κ_2 of $\Sigma_{\mathfrak{M}_2}$. For any $(\mathcal{L}_1 \times \mathcal{L}_2) \in \mathcal{D}om(\Gamma^{\zeta_1}_{\mathfrak{M}_1}) \times \mathcal{D}om(\Gamma^{\zeta_2}_{\mathfrak{M}_2})$, such that $\mathcal{L}_1 \cap \mathcal{L}_2 \neq \varnothing$ and $act = \mathcal{L}_{1|\varepsilon} = \mathcal{L}_{2|\varepsilon}$, we denote by $Cons(\mathcal{L}_1, \mathcal{L}_2)$ the following formula:

$$\forall \vec{x} : Ax^{-act}(\mathcal{L}_1 \cap \mathcal{L}_2)[\vec{x}]$$
$$\Rightarrow \left(Ax^{-\#}\big(\mathcal{S}ol_{\rho_1}(\Gamma^{\zeta_1}_{\mathfrak{M}_1}(\mathcal{L}_1))\big)[\vec{x}] \Leftrightarrow \Gamma^{\zeta_2}_{\mathfrak{M}_2}(\mathcal{L}_2)[\vec{x}]\right)$$

Finally, we denote by $Cons\left(\langle \mathfrak{M}_2^{\zeta_2}, X^{\kappa_2}\rangle \approx \langle \mathfrak{M}_1^{\zeta_1}, \rho_1^{\kappa_1}\rangle\right)$ the formula:

$$\mathcal{T}_{\mathfrak{M}_2} \wedge \bigwedge_{(\mathcal{L}_1,\mathcal{L}_2)\in \mathcal{D}om(\Gamma^{\zeta_1}_{\mathfrak{M}_1})\times\mathcal{D}om(\Gamma^{\zeta_2}_{\mathfrak{M}_2})} Cons(\mathcal{L}_1, \mathcal{L}_2)$$

This formula is a theory whose logical models are exactly interpretations inducing with $\mathfrak{M}_2$ the same set of authorizations as the source policy. This formula can have models which are not CD-interpretations. That is why it must be transformed in order to build only CD-interpretations (that is to say policy configurations).

Construction of a CD-model As previously said, the construction of the target policy is based on an algorithm building A-models presented in [21]. Let us define what are A-interpretations (and then A-models).

Definition 12 (A-interpretation [21]). *An interpretation $\Im$ of a signature Σ w.r.t. D is called an A-interpretation w.r.t. Σ' iff $D = \mathcal{T}(\Sigma') \cup \{\Lambda\}$ and for every n-ary predicate symbol $q \in \Sigma$, there exists an n-regular tree automaton accepting exactly the set of tuples $\vec{t}$ such that $\Im(q(\vec{t}))$ holds.*

A-interpretations are interpretations in which predicates and function symbols (seen as particular predicates) are interpreted as n-regular relations. In [21], Peltier proposes a method which, given a first order formula φ over a signature Σ' and a signature Σ, defines two applications denoted by Δ_Σ and $*$ such that: $\Im$ is a A-model of φ of domain $\mathcal{T}(\Sigma)$ iff $\Im = \mathcal{I}^*$ where $\mathcal{I}$ is a finite (logical) model of $\Delta_\Sigma(\varphi)$. Thus, building A-models comes down to building finite models. However, the set of A-interpretations does not correspond to the set of CD-interpretations. Nevertheless, we can show that a subset of A-interpretations is isomorphic to the set of CD-interpretations and that any first order formula can be transformed so that all its A-models are isomorphic to CD-models. This is formally stated in the following proposition.

Proposition 4. *Let φ be a first order formula. There exist two maps Θ and $\star$ such that $\Im$ is a CD-model of φ iff $\Im = \mathcal{I}^\star$ and $\mathcal{I}$ is an A-model of $\Theta(\varphi)$.*

Proof. Let us give the main ideas behind the construction of Θ and $\star$. It is based on the fact that any CD-interpretation is in bijection with an interpretation interpreting predicates and functions as regular sets of particular terms. Indeed, one can simulate n-ary functions and predicates with unary ones by adding a particular function symbol $\#$. More precisely, any n-ary predicate $p : \vec{s} \rightarrowtail \mathcal{B}ool$ is encoded by a predicate $p^\#$ such that $\Im(p^\#)$ recognizes the term $\#(\vec{t})$ iff $\Im^\star \models p(\vec{t})$. In the same way, any functional symbol $f : s_1 \times \ldots \times s_n \rightarrowtail s \neq \mathcal{B}ool$, is replaced by $f^\#$ so that $\Im(f^\#)$ recognizes tuples $\#(\vec{t}, u)$ iff $f(\vec{t}) = u$ holds in $\Im^\star$. The building of $\Im^\star$ from $\Im$ is based on the fact that any regular set E is equivalent to a finite set of constrained linear patterns CP in the sense where $t \in E$ iff there is $l \parallel \bigwedge_i x_i \in A_i$ (where $\vec{x} = \mathcal{V}ar(l)$) in CP such that $t = \sigma(l)$ and $\sigma(x_i) \in A_i$ for all i (several sets satisfying this definition exist but there is only one which factorizes common contexts as much as possible). More precisely, Θ is built as follows (we suppose that all formula are polite, that is two occurrences of the same bound variable is bound by the same quantifier and free and bound variables do not share any variable name). For any first order formula φ, we define $\Theta(\varphi)$ as the conjunction $\bigwedge_{f\in\Sigma} \mathcal{A}_f \wedge \mathcal{A}_\# \wedge \varphi\downarrow$ where the reduction relation $\twoheadrightarrow$ is defined by the following system:

$$
\begin{array}{|lcll|}
\hline
p(\vec{t}) = true & \twoheadrightarrow & \exists x : p(x) \wedge \#(\vec{t}, x) & \\
p(\vec{t}) = false & \twoheadrightarrow & \exists x : \neg p(x) \wedge \#(\vec{t}, x) & \\
at[t]_\omega & \twoheadrightarrow & \forall x : t = x \Rightarrow at[x]_\omega & t \notin \mathcal{X} \\
t = t' & \twoheadrightarrow & \exists x : t = x \wedge t' = x & t, t' \notin \mathcal{X} \\
f(\vec{s}) = x & \twoheadrightarrow & \forall y : \#(\vec{s}, x, y) \Rightarrow f(y) & \text{if } x \in \mathcal{X} \\
\hline
\end{array}
$$

with $\mathcal{A}_\# \triangleq \forall \vec{x}, \exists y : (\#(\vec{x}, y))$ and for any $f : \vec{s} \rightarrowtail s'$,

$$
\mathcal{A}_f \triangleq \left\{ \begin{array}{l} \forall \vec{x}, y_1, y_2, z_1, z_2 : \left(\begin{array}{l} f(z_1) \\ \wedge f(z_2) \\ \wedge \#(\vec{x}, y_1, z_1) \\ \wedge \#(\vec{x}, y_2, z_2) \end{array} \right) \Rightarrow y_1 \equiv y_2 \\ \forall \vec{x} : \left(\begin{array}{l} \bigwedge_i \mathcal{A}x(s_i)[x_i] \\ \Leftrightarrow \exists y, \forall z : \left(\begin{array}{l} \mathcal{A}x(s')[y] \wedge \#(\vec{x}, y, z) \\ \Rightarrow f(z) \end{array} \right) \end{array} \right) \end{array} \right.
$$

Next, let us explain how to translate tree automata into constrained pattern. Given an automaton $A = \langle Q, \Sigma, F, \rightarrow_\delta \rangle$ and $q \in Q$, we denote by $\rhd$ the relation $\{q \rhd q' \mid \exists f(\ldots, q', \ldots) \rightarrow_\delta q\}$. We denote by $\rhd^+$ the transitive closure of $\rhd$. A safe unfolding step of A is a pair $q \rightarrowtail f(q_1, \ldots, q_n)$ such that $f(q_1, \ldots, q_n) \rightarrow_\delta q$ and for any i, $q_i \not\rhd^+ q$. A state q is unsafe iff there exist no safe unfolding of A starting from q. We denote by $\rightarrowtail^+$ the relation over $Q \times \mathcal{T}(\Sigma \cup Q)$ such that $q \rightarrowtail^+ C[\vec{q_1}, \ldots, \vec{q_m}]$ iff $q \rightarrowtail f(q'_1, \ldots, q'_m), \forall i : q'_i \rightarrowtail^+ C_i[\vec{q_i}], C[\vec{q}] = f(C_1[\vec{q_1}], \ldots, C_m[\vec{q_m}])$. An unfolding $q \rightarrowtail^+ C[q_1, \ldots, q_n]$ is safe iff it is composed by only safe unfolding steps and it is maximal iff for any i, q_i is an unsafe state. We denote by $q\downarrow$ the set of safe and maximal unfolding starting from q. For any $q \in Q$, the set $q\downarrow$ is finite and the algorithm computing this set is straightforward (it is directly induced by the definition of $\rightarrowtail^+$). Finally, we denote by $CPattern(A)$ the set $\{C[\vec{x}]^{\parallel \bigwedge_i x_i \in A_i} \mid \exists q_F \in F, C[\vec{q}] \in q_F\downarrow, \forall i : A_i = \langle Q, \Sigma, \{q_i\}, \rightarrow_\delta \rangle\}$. Note that in $t^{\parallel x\in A \wedge \varphi}$, $x \in A$ is omitted if A recognizes $\mathcal{T}(\Sigma)^s$ where s is the sort of x.

Finally, let us explain how $\star$ is built. Let $\Im$ be an A-interpretation whose relations are unary and contains only terms of the form $\#(\vec{t})$. $\rho = \Im^\star$ is built as follows: for any $p : \vec{s} \rightarrowtail \mathcal{B}ool$, $\Im(p)$ is the tree automaton recognizing the set of data-terms $\#(\vec{t})$ such that $p(\vec{t})$ must hold in ρ.

- *for $\#(\vec{t})^{\parallel\varphi} \in CPattern(\Im(p))$, add $p(\vec{t}) \rightarrow true \parallel \varphi$ in ρ.*
- *add the default rule $p(\_, \ldots, \_) \rightarrow false$ in ρ*

For any $f : s_1 \times \ldots \times s_n \rightarrowtail s \neq \mathcal{B}ool$, $\Im(f)$ recognizes tuples $\#(\vec{t}, u)$ such that $f(\vec{t}) = u$ must hold in ρ.

- *for each $u \in \Pi_{n+1}(\Im(f))$ (Π_k is the projection over the k^{st} component), necessarily in a limited number, we denote by $\Im(f)^{-1}(u)$ the regular set of terms $\{\vec{t} \mid \#(\vec{t}, u) \in \Im(f)\}$ and then, for any $\#(\vec{t})^{\parallel\varphi} \in CPattern(\Im(f)^{-1}(u))$, we add $f(\vec{t}) \rightarrow u \parallel \varphi$ in ρ.*

□

Summary of the algorithm Our algorithm can be summarized by the Figure V.

Although there exist theories for which the existence of a finite model is undecidable, we have the two main following results. First, if a finite model of ϑ is found, then we can compute a security policy expressed in the target security model equivalent to the source policy. Secondly, if one proves that ϑ has no finite model, then we can conclude that there exists no policy expressed in the target security model equivalent to the source policy.

VI. Conclusion

The works presented in this paper come within the scope of designing a formal framework for specifying, analysing and maintaining security policies. We have argued that the specification of security policies in two steps with equational problems and rewriting systems is particularly attractive in allowing the reuse of specifications. Moreover, we claimed that such an approach is suitable to handle the problem of translating a policy to a given security model. We established in this paper several results and algorithms taking advantage on research advances in the rewriting and first order logical fields. We currently work on the implementation of our framework and on delineating the class of policies for which the transformation process is decidable (that is such

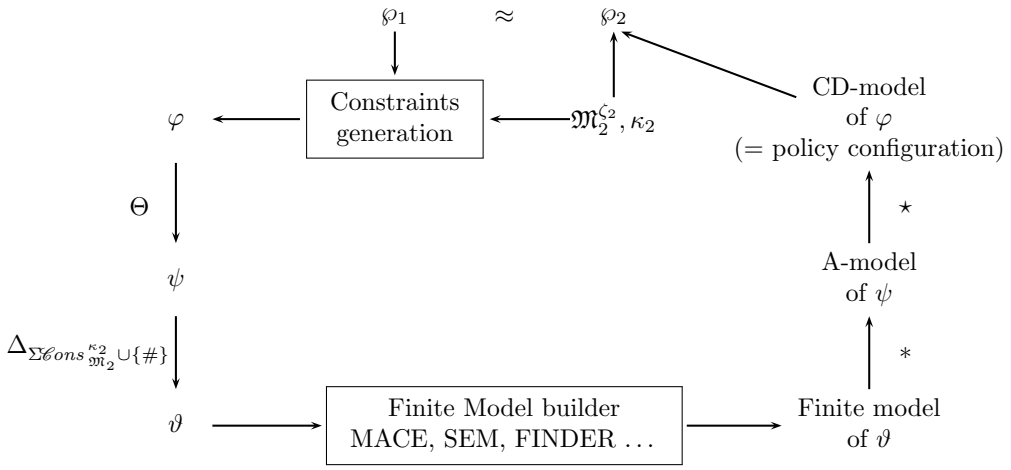


Figure. 3: Summary of the algorithm

that our algorithm answers either no if no translation exists or provides a target policy). In future work, we intend to consider more elaborated specifications for security models. More precisely, we wish to consider the use of a rewriting system which rewrites an action to an equational problem. This should lead to several new problems, like the consistency of a model (instead of the consistency of a policy). More generally, we want to investigate separately specific properties to models, specific ones to configurations and to obtain in a modular way properties over policies. Still in a context of reuse, we plan to consider the problem of security policies composition.

VII. Acknowledgments

We are grateful to Horatiu Cirstea and Hélène Kirchner for helpful feedback on this work and very fruitful discussions. We also thank Pierre-Etienne Moreau and Martin Quinson for having read and commented on an earlier draft. This work has been made possible through grants from the Region Lorraine and the french National Institute for Research in Computer Science and Control (INRIA).

References

[1] F. Baader and T. Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[2] S. Barker. The next 700 access control models or a unifying meta-model? In *ACM Symposium on Access control models and technologies*, pages 187–196. ACM, 2009.

[3] S. Barker, C. Bertolissi, and M. Fernández. Action control by term rewriting. *Electronic Notes in Theoretical Computer Science*, 234:19–36. Elsevier Science B.V., 2009.

[4] S. Barker and M. Fernández. Term rewriting for access control. In *Data and applications security XX*, volume 4127 of *Lecture Notes in Computer Science*, pages 179–193. Springer-Verlag, 2006.

[5] S. Barker and M. Fernàndez. Action-status access control as term rewriting. In *International Workshop on Security and Rewriting Techniques*, 2007.

[6] S. Barker and P. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Transactions on Information and System Security*, 6(4):501–546. ACM, 2003.

[7] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security*, 6(1):71–127. ACM, 2003.

[8] C. Bertolissi and M. Fernández. A rewriting framework for the composition of access control policies. In *International ACM SIGPLAN conference on Principles and Practice of Declarative Programming*, pages 217–225. ACM, 2008.

[9] C. Bertolissi, M. Fernández, and S. Barker. Dynamic event-based access control as term rewriting. In *Data and Applications Security XXI*, volume 4602 of *Lecture Notes in Computer Science*, pages 195–210. Springer-Verlag, 2007.

[10] H. Cirstea, P.-E. Moreau, and A. de Oliveira. Rewrite based specification of access control policies. *Electronic Notes in Theoretical Computer Science*, 234:37–54. Elsevier Science B.V., 2009.

[11] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://gforge.inria.fr/projects/tata/, 2008.

[12] A. de Oliveira, E. Wang, C. Kirchner, and H. Kirchner. Weaving rewrite-based access control policies. In *ACM Workshop on Formal methods in security engineering*, pages 71–80. ACM, 2007.

[13] D. Dougherty, C. Kirchner, H. Kirchner, and A. de Oliveira. Modular Access Control via Strategic Rewriting. volume 4734 of *Lecture Notes in Computer Science*, pages 578–593. Springer-Verlag, 2007.

[14] R. Echahed and F. Prost. Security policy in a declarative style. In *International ACM SIGPLAN conference on Principles and Practice of Declarative Programming*, pages 153–163. ACM, 2005.

[15] 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. Dynamic Publishers Inc., USA, 2009.

[16] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In *Computational Logic: Essays in Honor of Alan Robinson*, chapter 8, pages 257–321. The MIT-Press, 1991.

[17] A. Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin. Organization based access control. *IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 120–131. IEEE Computer Society, 2003.

[18] C. Kirchner, H. Kirchner, and A. de Oliveira. Analysis of rewrite-based access control policies. *Electronic Notes in Theoretical Computer Science*, 234:55–75. Elsevier Science B.V., 2009.

[19] L. LaPadula and D. Bell. Secure Computer Systems: A Mathematical Model. *Journal of Computer Security*, 4:239–263. IOS Press, 1996.

[20] P.-A. Masson, M.-L. Potet, J. Julliand, R. Tissot, G. Debois, B. Legeard, B. Chetali, F. Bouquet, E. Jaffuel, L. Van Aertrick, J. Andronick, and A. Haddad. An access control model based testing approach for smart card applications: Results of the POSE project. *Journal of Information Assurance and Security*, 5(1):335–351. Dynamic Publishers Inc., USA, 2010.

[21] N. Peltier. Constructing infinite models represented by tree automata. *Annals of Mathematics and Artificial Intelligence*, 56(1):65–85. Springer Science + Business Media B.V, 2009.

[22] R. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19. IEEE Computer Society, 1993.

[23] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *Computer*, 29(2):38–47. IEEE Computer Society, 1996.

[24] R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48. IEEE Communications Society, 1994.

[25] N. Souayeh, A. Bouhoula, and F. Jacquemard. Automatic Validation of Firewall Configurations using SMT Solvers. *Journal of Information Assurance and Security*, 5(1):561–568. Dynamic Publishers Inc., USA, 2010.

Author Biography

Tony Bourdier received his master's degree in applied Mathematics from the University Henri Poincaré and his engineer's degree in Computer Science from the École Supérieure d'Informatique et Applications de Lorraine in 2007. He currently works as a PhD candidate on specification and verification of security policies with formal methods at the french National Institute for Research in Computer Science and Control (INRIA).