On Formal Specification and Analysis of Security Policies
Tony Bourdier, Horatiu Cirstea, Mathieu Jaume, Hélène Kirchner

To cite this version:

HAL Id: inria-00429240
https://inria.hal.science/inria-00429240v3
Submitted on 30 Jul 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
On Formal Specification and Analysis of Security Policies

Tony Bourdier\textsuperscript{1}, Horatiu Cirstea\textsuperscript{1}, Mathieu Jaume\textsuperscript{2}, Hélène Kirchner\textsuperscript{3}

\textsuperscript{1} INRIA Nancy Grand Est · Nancy Université · LORIA
BP 101, 54602 Villers-lès-Nancy Cedex, France
Tel.: +33354958415

\textsuperscript{2} SPI - LIP6, Université Paris 6
104 av du Président Kennedy, 75016 Paris, France
Mathieu.Jaume@lip6.fr

\textsuperscript{3} INRIA Bordeaux Sud-Ouest
351, Cours de la Libération, 33405 Talence Cedex, France
Helene.Kirchner@inria.fr

\textbf{ABSTRACT.} Security policies are ubiquitous in information systems and more generally in the management of sensitive information. Access control policies are probably the most largely used policies but their application goes well beyond this application domain. The enforcement of security policies is useless if some of their key properties like the consistency, for example, cannot be stated and checked. We propose here a framework where the security policies and the systems they are applied on, are specified separately but using a common formalism. This separation allows us not only some analysis of the policy independently of the target system but also the application of a given policy on different systems. Besides the abstract formalism we also explore how rewrite and reduction systems can be used and combined in a rather systematic way to provide executable specifications for this framework. We also propose a notion of system and policy transformation that gives the possibility to study some properties which cannot be expressed only within the initial presentation. We have shown, in particular, how confidentiality, integrity and confinement can be expressed for the BLP policy that does not deal explicitly with information flows but only with objects containing tractable information.

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. In information systems, a security policy can put constraints on functions and information flows, on access to resources or data by external systems or persons. One may call security policies special programs 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

\textsuperscript{*} This work was partially supported by ANR project “SSURF”. The first author has obtained a partial PhD grant from the Région Lorraine.
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 such programs is their reactive behaviour with respect to their execution environment: on one hand, a running program may query the policy for an authorization before performing specific accesses or transactions; on the other hand, the answers of the policy may change the execution of the calling program by excluding possible executions that do not comply with the policy.

Most of the first general and simple policies for access control were implemented using access control matrices but, nowadays, we are confronted with a lot of specific policies whose aims can be very different. In large systems, there are many classes of subjects with different needs for processing a variety of resources. Different subjects usually have different (even competing) requirements on the use of resources and their security goals (confidentiality, availability, integrity) may be distinct. Hence, various access requirements have to be consistently authorized and maintained. Beyond access control, security policies also address complex authorizations with delegations that may involve quite intricate computations. The security policies have to be deployed in the actual context of distributed organizations, mobile users and on-line services, and this implies that these policies should be easily composable, highly reconfigurable, time dependent and reactive to their environment.

Security policies may be written by different parties at different times, so it is necessary to detect and resolve conflicts among them. Detecting and solving conflicts is crucial for the policy designers who need analysis and verification tools, as well as methodology for conflict resolution. Suitable properties of policies, such as consistency, have to be clearly identified and proved. This is impossible without a formal semantics in which to express the policies and their execution environment. Formal methods and logical approaches provide already some help but should be better tuned to the specific domain and properties of security policies. This is especially important for the formalization and understanding of specific properties such as privacy or trust. Indeed, the increasing complexity of policies raises up the complexity of techniques used to reason about them.

Faced to this increasingly complex situation, our first contribution is to propose here a framework where the security policies and the systems they are applied on, are specified separately but using a common formalism. This separation allows not only the analysis of the policy independently of the target system, but also the application of a given policy on different systems. Our second contribution is to explore how rewrite and reduction systems can be used and combined in a rather systematic way to provide executable specifications for this framework. Finally, we introduce a notion of transformation on environments that can be used to reason about properties a policy is supposed to ensure and that can be naturally expressed in a different presentation. Typically, this is the case for flow properties, such as confidentiality, integrity or confinement, which can be expressed in a generic way and checked on concrete systems secure w.r.t. an access control policy.

The paper is organized as follows. After Section 2 that introduces the basic concepts and notations, Section 3 defines specifications (Section 3.1), environments (Section 3.2), systems (Section 3.3), security policies (Section 3.4); all these notions allow us to define (in Section 3.5) the application of a policy to a system. Section 4 defines environment transformations and show how to use them to check security properties. Section 5 concludes with related works and further research directions.
2 Preliminaries

We give in this section some definitions and notations that we use in the following. More details can be found, for example, in [EM85, End72]. A many-sorted signature \( \Sigma \) 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 \( w = s_1, \ldots, s_n \in \mathcal{S}^* \) and co-arity \( s \) is written \( f : w \rightarrow s \). A predicate symbol \( p \) with arity \( s_1, \ldots, s_n \in \mathcal{S}^* \) is written \( p : w \). Variables are also sorted and \( x : s \) means that variable \( x \) has sort \( s \).

The set \( \mathcal{X}_s \) denotes a set of variables of sort \( s \) and \( \mathcal{X} = \bigcup_{s \in \mathcal{S}} \mathcal{X}_s \) is the set of many-sorted variables. Many-sorted terms are built on many-sorted signatures and classified according to their sorts. The set of terms of sort \( s \), denoted \( \mathcal{T}_s^{\mathcal{S}, \mathcal{X}} \), is the smallest set containing \( \mathcal{X}_s \) and all the terms \( f(t_1, \ldots, t_n) \) such that \( f \in \mathcal{F}_s \) and \( t_i \in \mathcal{T}_s^{\mathcal{S}, \mathcal{X}} \) for \( i \in [1..n] \). The set of all sorted terms is \( \mathcal{T}_s^{\mathcal{S}, \mathcal{X}} = \bigcup_{s \in \mathcal{S}} \mathcal{T}_s^{\mathcal{S}, \mathcal{X}} \).

Let \( \mathbb{N}_+ \) be the set of positive naturals and \( \mathbb{N}_+^s \) the corresponding monoid with neutral element \( 1 \) and the concatenation operator \( . \). We call position any element \( \omega \) of \( \mathbb{N}_+^s \). For all \( m, n \in \mathbb{N}_+^s \), \( m \) is a prefix of \( n \), denoted by \( m \leq n \), if it exists \( n' \in \mathbb{N}_+^s \) such that \( n = m.n' \). A term (seen as a tree) is an application \( t \) from a non-empty part \( \text{Pos}(t) \) of \( \mathbb{N}_+^s \) to \( \mathcal{F} \cup \mathcal{X} \) such that \( \text{Pos}(t) \) is closed under prefix (i.e. if \( \omega \in \text{Pos}(t) \), then all prefixes of \( \omega \) belong to \( \text{Pos}(t) \)) and for all \( m \in \text{Pos}(t) \) and any \( i \in \mathbb{N}_+^s \), \( m.i \in \text{Pos}(t) \) if and only if \( t(m) = f \in \mathcal{F} \) and \( 1 \leq i \leq \text{arity}(f) \). \( \text{Pos}(t) \) is called the set of positions of \( t \). The notation \( t|_\omega \) is used to denote the subterm of \( t \) at position \( \omega \). We denote by \( t[s]_\omega \) the term \( t \) with the subterm at position \( \omega \) replaced by \( s \). The set of terms occurring in a term \( t \) is denoted by \( \forall \text{ar}(t) \). If \( \forall \text{ar}(t) \) is empty, \( t \) is called a ground term and \( \mathcal{T}_\mathbb{N} \) is the set of ground terms. We call substitution any mapping from \( \mathcal{X} \) to \( \mathcal{T}_{\mathbb{S}} \) which is the identity except over a finite set of variables called domain of \( \sigma \) and denoted by \( \text{Dom}(\sigma) \). The set \( \text{Codom}(\sigma) = \{ t \in \mathcal{T}_{\mathbb{S}} \mid \exists x, \sigma(x) = t \} \) is called codomain of \( \sigma \). Moreover, \( \sigma \) is extended to an endomorphism of \( \mathcal{T}_{\mathbb{S}} \). If \( \text{Codom}(\sigma) \subseteq \mathcal{T}_{\mathbb{S}} \), \( \sigma \) is said to be a ground substitution.

A \( \Sigma \)-atom is either an equality \( s = t \) where \( s \) and \( t \) are terms of the same sort, or an object of the form \( p(t_1, \ldots, t_n) \) where \( p : s_1, \ldots, s_n \in \mathcal{P} \) and \( t_i \in \mathcal{T}_{s_i}^{\mathcal{S}, \mathcal{X}} \) for \( i \in [1..n] \). The set of all \( \Sigma \)-atoms is denoted by \( \mathcal{A}_{\Sigma} \). A \( \Sigma \)-literal if either a \( \Sigma \)-atom (positive literal) or \( \neg a \) where \( a \) is a \( \Sigma \)-atom (negative literal). A \( \Sigma \)-clause is either a literal or \( l \lor c \) where \( l \) is a literal and \( c \) a clause. If a clause contains at most one positive literal, it is called a Horn clause; if it contains exactly one positive literal, it is called a definite Horn clause. The set of \( \Sigma \)-formulae is the smallest set of expressions, denoted by \( \mathcal{F}_{\Sigma} \), containing \( \mathcal{A}_{\Sigma} \) and such that if \( \varphi, \psi \in \mathcal{F}_{\Sigma} \) and \( x \in \mathcal{X} \) then \( \neg \varphi, \varphi \land \psi, \varphi \lor \psi, (\forall x) \varphi, (\exists x) \varphi \) and \( (\exists x) \varphi \) are in \( \mathcal{F}_{\Sigma} \). \( \varphi \Rightarrow \psi \) is a shortcut for \( \neg \varphi \lor \psi \). Free and bound variables of a formula \( \varphi \) are defined as usual in first-order logic and are respectively denoted by \( \mathcal{F} \text{Var}(\varphi) \) and \( \mathcal{B} \text{Var}(\varphi) \). A formula without free variables is said closed and one without any variable is said ground. We call theory any set of formulae \( \mathcal{F} \). A Horn (resp. definite Horn) theory is a theory containing only Horn (resp. definite Horn) clauses.

**Definition 1.** [\( \Sigma \)-algebra] Given a signature \( \Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}) \), a \( \Sigma \)-algebra \( \mathcal{A} \) is given by:
- for all sort \( s \in \mathcal{S} \), a set \( \mathcal{A}^s \) called domain of sort \( s \). We denote by \( |\mathcal{A}| = \bigcup_{s \in \mathcal{S}} |\mathcal{A}^s| \)
- for each \( f : s_1, \ldots, s_n \in \mathcal{S} \), a mapping \( f_{\mathcal{A}} : \mathcal{A}^{s_1} \times \ldots \times \mathcal{A}^{s_n} \rightarrow \mathcal{A}^s \)
- for each \( p : s_1, \ldots, s_n \in \mathcal{P} \), a relation \( p_{\mathcal{A}} \) over \( \mathcal{A}^{s_1} \times \ldots \times \mathcal{A}^{s_n} \)

We denote by \( \mathcal{A} |_{\text{sg}} \Sigma \) the set of all \( \Sigma \)-algebras.

**Definition 2.** [Valuation, semantics, model, \( \mathcal{A} \)-solutions] Given a signature \( \Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{P}) \), a \( \Sigma \)-algebra \( \mathcal{A} \) and a set of variables \( \mathcal{X} \) sorted by \( \mathcal{S} \), an \( \mathcal{A} \)-valuation \( \alpha \) is a mapping which associates
to each variable $x$ a set of rewrite rules over $\Sigma$ and a countable set of variables $\mathcal{X}$, we call rewrite rule over $\Sigma$ any pair $(\text{lh}s, \text{rh}s) \subseteq \Sigma_\mathcal{X} \times \Sigma_\mathcal{X}$ such that $\forall \text{ar}(\text{rh}s) \subseteq \forall \text{ar}(\text{lh}s)$ and $\text{lh}s \notin \mathcal{X}$. A term rewriting system or TRS over $\Sigma$ is a set of rewrite rules over $\Sigma$. Given a TRS $\mathcal{R}$ and a term $t \in \mathcal{F}_\Sigma$, $\mathcal{R}$ rewrites $t$ to $u \in \mathcal{F}_\Sigma$ with $\mathcal{R}$, denoted by $t \rightarrow_{\mathcal{R}} u$, iff there exists a position $\omega$ in the term $t$ and a ground substitution $\sigma$ such that $\sigma(\text{lh}s) = t_{\omega}$ and $u = t \sigma(\text{rh}s)_{\omega}$ (in such a case we also say that $t$ rewrites to $u$ with the rule $(\text{lh}s \rightarrow \text{rh}s)$).

We call constrained rule any tuple $(\text{lh}s, \varphi, \text{rh}s) \subseteq \Sigma_\mathcal{X} \times \mathcal{F}_\Sigma \times \Sigma_\mathcal{X}$ such that $\forall \text{ar}(\text{rh}s) \subseteq \forall \text{ar}(\text{lh}s) \cup \forall \text{ar}(\varphi)$ and $\text{lh}s \notin \mathcal{X}$. A constrained term rewriting system or CTRS is a set of constrained rules. Given a CTRS $\mathcal{R}$, a term $t \in \mathcal{F}_\Sigma$ and a $\mathcal{A}$-algebra $\mathcal{A}$ whose domain is a subset of $\mathcal{F}_\Sigma$, we say that $t$ rewrites to $u \in \mathcal{F}_\Sigma$ by $\mathcal{R}$ in $\mathcal{A}$, which is also denoted by $t \rightarrow_{\mathcal{A}}^c u$, iff there exists a rule $\text{lh}s \rightarrow_{\mathcal{R}} \text{rh}s$ in $\mathcal{R}$, a position $\omega \in \text{Pos}(t)$ and a ground substitution $\sigma$ such that $\sigma(\text{lh}s) = t_{\omega}, \mathcal{A} \models \varphi(\sigma)$ and $u = t \sigma(\text{rh}s)_{\omega}$. Indeed, this definition requires that $\text{hol}_{\mathcal{A}}(\varphi)$ is computable.

As usual, given a relation $\rightarrow$ (resp. $\rightarrow^*$, $\rightarrow^*$) denotes the transitive (resp. reflexive and transitive, symmetric) closure of $\rightarrow$.

Given a constrained term rewriting system defined on a signature $\Sigma$, we say that $f \in \mathcal{F}$ depends on $g \in \mathcal{F}$ if and only if there exists a rule $\text{lh}s \rightarrow_{\mathcal{R}} \text{rh}s$ such that the $\text{lh}s(e) = f$ and $g$ occurs in $\varphi$. In order to avoid some recursive definitions that could lead to non terminating function evaluation, we assume that the dependency graph of functions has no loop. In such cases, the CTRS $\mathcal{R}$ is said to be an unrecursive constrained term rewriting system (UCTRS).

**Definition 4.** [Confluence, normalization, convergence, normal form] Given a signature $\Sigma$ and a TRS $\mathcal{R}$, we say that a term $t \in \mathcal{F}_\Sigma$ is in normal form if and only if there exists no $t' \in \mathcal{F}_\Sigma$ such that $t \rightarrow_{\mathcal{R}} t'$. We also say that $\mathcal{R}$ is confluent iff for all $t, u, v \in \mathcal{F}_\Sigma$ such that $t \rightarrow_{\mathcal{R}} u$ and $t \rightarrow_{\mathcal{R}} v$, there exists a unique $t' \in \mathcal{F}_\Sigma$ such that $u \rightarrow_{\mathcal{R}} t'$ and $v \rightarrow_{\mathcal{R}} t'$, terminating iff there is no infinite chain $t_0 \rightarrow_{\mathcal{R}} t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} \ldots$ and convergent iff it is confluent and terminating. In the latter case, any $t \in \mathcal{F}_\Sigma$ can be associated to a unique $t' \in \mathcal{F}_\Sigma$ in normal form such that $t \rightarrow_{\mathcal{R}} t'$: we say that $t'$ is the $\mathcal{R}$-normal form of $t$ and we denote it by $t' = t \downarrow_{\mathcal{R}}$. All these notions are extended to CTRS by replacing $\rightarrow_{\mathcal{R}}$ by $\rightarrow_{\mathcal{A}}^c$ for any algebra $\mathcal{A}$. In this case, we talk about $\mathcal{A}$-confluence, $\mathcal{A}$-termination and $\mathcal{A}$-convergence. For any theory $\mathcal{T}$, $\mathcal{R}$ is said $\mathcal{T}$-confluent, $\mathcal{T}$-terminating and $\mathcal{T}$-convergent iff it is $\mathcal{A}$-confluent, $\mathcal{A}$-terminating and $\mathcal{A}$-convergent for any algebra $\mathcal{A} \in \text{Mod}(\mathcal{T})$.

Definitions and properties of constrained rewrite systems have been studied, for instance, in [KKR90].
We may assume given a partial order $>$ on rules of a TRS or CTRS $\mathcal{R}$ to express reduction priority. In this case, a term $t$ rewrites to $u$ with a rule $r$ only if it cannot be rewritten with another rule $r'$ such that $r' < r$. In what follows we always consider that the reductions w.r.t. TRSs and CTRSs are ordered by the order of presentation of rules. This gives the possibility to write a last rule with a variable as left-hand side, to handle all cases that do not match the previous patterns.

## 3 Security Policies over Systems

We propose here a framework where the security policies and the systems they are applied on are specified separately but using a common formalism.

We are interested in systems that change their state according to some sensitive actions. Since the notions of state and (restricted) transition are central, we first define transition systems on which policies can apply. We start from the classical notion of labelled transition systems (LTS) defined as tuples $(\mathcal{E}nv, \mathcal{E}nv_0, \mathcal{L}, \delta)$ where $\mathcal{E}nv$ is a set of states, $\mathcal{E}nv_0 \subseteq \mathcal{E}nv$ is a set of initial states, $\mathcal{L}$ is a set of action labels and $\delta \subseteq \mathcal{E}nv \times \mathcal{L} \times \mathcal{E}nv$ is a transition relation, and we make precise in the next sections the way these components are specified.

Since we are mainly interested in the behaviour of a system under a given policy, we focus on the sensitive actions concerned by the policy. When a (security) request is performed on a system in a given state, the new state of the system depends on the decision taken by the policy for the respective request. We can already point out that in the general case, the response to a request is not binary and the corresponding decision can be different from the usual accept/deny answer. The actions we are interested in are thus pairs query-decision that we call in what follows events.

A security policy depends partly on the information that characterize the states of a system but does not depend on the way the system evolves. We focus on this sensitive information and on all information that are relevant for the evolution of a system under a security policy and we group all these information in so-called environments that correspond to (part of) the states of a LTS. The systems are then characterized by transitions between environments and the security policies are specified as decisions for requests performed into an environment.

In this section, we first propose formal definitions for environments and events and, based on these definitions, we define systems and security policies and we formalize the application of a policy to a system. After giving these definitions in the general framework of first-order logic, we identify presentations of first-order theories where rewriting techniques can be used to get more operational concepts.

### 3.1 Specifications

The interactions between the policy and the system is applied on are performed through some events, which are exactly pairs consisting of a query and the associated decision. The formal specifications we consider should thus introduce the sorts $Q(\text{uery})$ and $D(\text{ecision})$.

**Definition 5.** (Specification) We call specification a 3-tuple $\mathcal{SP} = (\Sigma \cup \Sigma_{Ev}, T, \Delta)$ where $\Sigma, \Sigma_{Ev}$ are signatures such that the co-arity of every symbol in the event signature $\Sigma_{Ev}$ is one of the two $\Sigma_{Ev}$-sorts $Q$ or $D$, $T$ is a set of $\Sigma$-formulae called theory of $\mathcal{SP}$ and $\Delta$ is a finite subset of $F_{\mathcal{L}}$ called domain of $\mathcal{SP}$. We denote by $Q$ and $D$ the sets $F_{\Sigma, \Sigma_{Ev}}^Q$ (of queries) and $F_{\Sigma, \Sigma_{Ev}}^D$ (of decisions) respectively.
The previous definition differs from the standard notion of specification by the additional component \( \Delta \) whose purpose is to restrict the domain of the algebraic models of the specification, as follows:

An algebra \( \mathcal{A} \) is a model of a specification \( \mathcal{SP} = (\Sigma, \mathbf{T}, \Delta) \) iff \( \mathcal{A} \in \mathfrak{I}_{\mathcal{G}_E} \) such that \( |\mathcal{A}| = \Delta \) and is a model of each formula in \( \mathbf{T} \). We denote by \( \mathcal{M}_{\mathfrak{od}(\mathcal{SP})} \) the set of all models of \( \mathcal{SP} \).

**Example 6.** Along the lines of this paper, we consider the mandatory part of the Bell and LaPadula (BLP) access control policy [BL96a, BL73]. The BLP 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 \( (L, \inf) \)) associated with subjects and objects. Hence, we define the two following signatures:

- \( \Sigma_{BLP} = (\mathcal{I}_{BLP}, \mathcal{F}_{BLP}, \mathcal{P}_{BLP}) \) where:

\[
\mathcal{I}_{BLP} = \{S, O, A, L\} \quad \mathcal{F}_{BLP} = \left\{ \begin{array}{c} r : A \rightarrow A \\
q : A \rightarrow A \\
f_s : S \rightarrow L \\
f_o : O \rightarrow L \\
\end{array} \right\} \quad \mathcal{P}_{BLP} = \left\{ \begin{array}{c} \inf : L \rightarrow L \\
m : S, O, A \rightarrow \mathcal{L} \end{array} \right\}
\]

The functions \( f_s \) and \( f_o \) describe security levels associated with subjects and objects. The predicate \( m \) describes current accesses over objects by subjects: \( m(s, o, a) \) means that the subject \( s \) has an access over an object \( o \) according to the access mode \( a \).

- \( \Sigma_{Ev} = (\{Q, D\}, \mathcal{F}_{Ev}, \emptyset) \) where:

\[
\mathcal{F}_{Ev} = \left\{ \begin{array}{c} \text{ask} : S, O, A \rightarrow Q \\
\text{release} : S, O, A \rightarrow Q \\
\text{permit} : \rightarrow D \\
\text{deny} : \rightarrow D \\
\end{array} \right\}
\]

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 \( a \).

- \( \Delta_{BLP} \) contains all constants of \( \Sigma_{BLP} \), namely \( \{r, w\} \).

We are now in position to define the specification \( \mathcal{SP}_{BLP} = (\Sigma_{BLP} \cup \Sigma_{Ev}, \mathbf{T}_{BLP}, \Delta_{BLP}) \) where \( \mathbf{T}_{BLP} \) is the theory expressing that \( (L, \inf) \) is a lattice:

\[
\mathbf{T}_{BLP} = \left\{ \begin{array}{c}
\forall x \inf(x, x) \\
\forall x, y \inf(x, y) \land \inf(y, x) \Rightarrow x = y \\
\forall x, y, z \inf(x, y) \land \inf(y, x) \Rightarrow \inf(x, z) \\
\forall x, y, z \exists w (\inf(w, x) \land \inf(w, y) \land \forall w (\inf(w, x) \land \inf(w, y) \Rightarrow \inf(w, z))) \\
\forall x, y, z \exists w (\inf(x, z) \land \inf(y, z) \land \forall w (\inf(x, w) \land \inf(y, w) \Rightarrow \inf(x, w))) \\
\end{array} \right\}
\]

As illustrated in the previous examples, the specifications (of security policies) are in general defined for generic sets of subjects, objects, actions... However, in specific environments where they are applied, it is necessary to consider specific instances of these sets. The notion of extended specification defined below provides a formal way to consider such instances.

**Definition 7.** [Extended specification] Let \( \mathcal{SP} = (\Sigma, \mathbf{T}, \Delta) \) be a specification where \( \Sigma = (\mathcal{I}, \mathcal{F}, \mathcal{P}) \). For any pair \( \mathcal{E} = (\mathcal{F}_E, \Delta_E) \) where \( \mathcal{F}_E \) is a finite set of functional symbols whose arity and co-arity are in \( \mathcal{I} \) and \( \Delta_E \) a finite set of ground terms build from \( \Sigma \) and \( \mathcal{F}_E \), we call specification extended by \( \mathcal{E} \) from \( \mathcal{SP} \) the specification denoted by \( \mathcal{SP}[\mathcal{E}] \) and defined by \( (\Sigma[\mathcal{E}], \mathbf{T}, \Delta[\mathcal{E}]) \) where \( \Sigma[\mathcal{E}] = \{\mathcal{I}, \mathcal{F} \cup \mathcal{F}_E, \mathcal{P}\} \) and \( \Delta[\mathcal{E}] = \Delta \cup \Delta_E \).
**Example 8.** Extended specifications are useful to consider particular sets of constants involved in the considered policy. For example, if we want to deal with the following sets:

\[ \text{Subj} = \{ \text{Bob}, \text{Alice}, \text{Charlie} \} \quad \text{Obj} = \{ \text{File}_1, \text{File}_2, \text{File}_3 \} \quad \text{Levels} = \{ \text{Secret}, \text{Confidential}, \text{L}_1, \text{L}_2, \text{Public}, \text{Sanitized} \} \]

we can consider the extended specification \( \mathcal{SP}_{BLP} \) where \( \mathcal{F}_\mathcal{E} = \Delta_\mathcal{E} = \text{Subj} \cup \text{Obj} \cup \text{Levels} \).

### 3.2 Environments

As already mentioned, the information needed by a policy to take a decision with respect to a request or to define the evolution of a system are gathered in so-called environments. An environment can be thus viewed as (the significant fragment of) the current state of a system; it consists, for example, of the set of subjects, the set of objects and the set of current accesses for an access control policy.

It seems natural to use a first-order language for the definition of the various components of a system and of a security policy. In particular, the terms of a many-sorted signature could be (and often are) used to specify the various components of a system and of a policy applied on it. Nevertheless, this is somewhat restrictive since the different actors (like the subjects and objects in an access control policy) involved in such a specification should be made precise only when the system and policy are defined. An alternative approach consists in interpreting the symbols used in the definition of the policy into the signature that is effectively used. We define thus the environments as syntactic interpretations, while the queries and decisions (i.e. the actions) are simply first-order terms built on a given signature.

More precisely, we define an environment as the model of a set (i.e. of a conjunction) of first-order formulae called theory, which further constrains the interpretation by expressing, for instance, some static properties of the environment. For instance, in Example 6, these constraints are used to specify that the security levels of the Bell and LaPadula policy are organized in a finite lattice.

A crucial point in our approach is to be able to decide whether a given first-order formula expressed as a constraint, holds in the environment. As the information needed by a policy always depend on a finite set of entities (subjects, objects, resources and so on), we will consider for defining an environment, algebras with a finite interpretation domain composed only of ground terms. The following definition formalizes the construction of such algebras.

**Definition 9.** [Finite term generated \( \Sigma \)-algebra] Given a signature \( \Sigma = (\mathcal{I}, \mathcal{F}, \mathcal{P}) \), we call **finite term generated \( \Sigma \)-algebra** any \( \Sigma \)-algebra \( \mathcal{A} \) such that for all \( s \in \mathcal{I} \), \( \mathcal{A}^s \) is a finite subset of \( \mathcal{F}_\mathcal{L} \) sorted by \( s \) and such that any term \( t \) belonging to this subset is interpreted by itself: \( t_\mathcal{A} = t \). We denote by \( \text{FTG}\text{-}\mathcal{A}l_{\mathcal{G}} \) the set of all finite term generated \( \Sigma \)-algebras and by \( \text{FTG}\text{-}\mathcal{A}l_{\mathcal{D}} \) the set of all finite term generated \( \Sigma \)-algebras whose domain is \( \Delta \). The notation \( \text{FTG}\text{-}\mathcal{M} \text{od}(\mathcal{F}h) \) (resp. \( \text{FTG}\text{-}\mathcal{M} \text{od}^\Delta(\mathcal{F}h) \)) refers to an \( \text{FTG}\text{-}\mathcal{A}l_{\mathcal{G}} \) (resp. \( \text{FTG}\text{-}\mathcal{A}l_{\mathcal{D}} \)) model of \( \mathcal{F}h \) where \( \mathcal{F}h \) is a \( \Sigma \)-theory.

We are now ready to define the notions of environment and event, based on a given specification.

**Definition 10.** [Environment, Event] Given a specification \( \mathcal{SP} = (\Sigma \cup \Sigma_{Ev}, \mathcal{T}, \Delta) \) (extended or not), an **\( \mathcal{SP} \)-environment** \( \mathcal{E} \) is a \( \Delta \)-\( \text{FTG} \) \( \Sigma \)-algebra model of \( \mathcal{SP} \) (i.e. an element of \( \text{FTG}\text{-}\mathcal{M} \text{od}^\Delta(\mathcal{T}) \)). We will denote the set of all \( \mathcal{SP} \)-environments by \( \mathcal{E}_{\mathcal{SP}} \) (or simply \( \mathcal{E} \) when there is no ambiguity). An **\( \mathcal{SP} \)-event** (or simply **event**) is a pair \( \gamma = (q, d) \) where \( q \in \mathcal{D} \) is the query of \( \gamma \) and \( d \in \mathcal{D} \) its
**Example 11.** If we consider the \( \text{SP}_{BLP}[\mathcal{E}] \) specification of Example 8, the following algebra, denoted by \( e_{BLP} \), is an environment in \( \varepsilon_{\text{SP}} \text{SP}_{BLP}[\mathcal{E}] \):

- \( (e_{BLP})^S = \text{Subj} \quad (e_{BLP})^O = \text{Obj} \quad (e_{BLP})^T = \text{Levels} \)
- \( (f_{e_{BLP}}) = \{ \begin{array}{ll}
\text{Bob} & \rightarrow \text{Secret} \\
\text{Alice} & \rightarrow L_2 \\
\text{Charlie} & \rightarrow \text{Sanitized}
\end{array} \}
\)
- \( \text{inf}_{e_{BLP}} = \{ \begin{array}{l}
\text{(Secret,Secret), (Confidential,Confidential), (Confidential,Secret), (L_1, L_1), (L_1,\text{Confidential}), (L_1,\text{Secret}), (L_2,\text{L_2}), (L_2, \text{Confidential}), (L_2,\text{Secret}), (Public,Public), (Public,L_1), (Public,L_2),
\text{Public,Confidential}, (Public,\text{Secret}), (Sanitized,\text{Sanitized}), (Sanitized,Public), (Sanitized,L_1), (Sanitized,L_2), (Sanitized,\text{Confidential}), (Sanitized,\text{Secret})
\end{array} \}
\)
- \( m_{e_{BLP}} = \emptyset \)

Inspired by the construction of a minimal Herbrand model in first-order logic, we now define a minimal finite term generated model and give conditions under which such a minimal model exists. This definition also allows us to characterize any environment \( e \) by a theory \( \mathcal{T}h \) such that \( e \) is a minimal finite term generated model of \( \mathcal{T}h \). When an algebra satisfies an infinite set of ground atoms, the interest of such a characterization is obvious; when the set of ground atoms valid in the algebra is finite, then such a characterization allows us to define environments in a more “compact” way and gives the possibility to have more efficient algorithms to compute solutions of a formula in an environment. Moreover, we need a concrete way to express environments in order to compare or modify them.

**Definition 12.** [Minimal \( \Delta \)-FTG model] Given a signature \( \Sigma = (\mathcal{L}, \mathcal{F}, \mathcal{P}) \), a subset \( \Delta \) of \( \mathcal{F}_2 \) and a consistent definite Horn \( \Sigma \)-theory \( \mathcal{T}h \), we call a minimal \( \Delta \)-FTG model of \( \mathcal{T}h \) the \( \Delta \)-FTG algebra model of \( \mathcal{T}h \), denoted by \( \text{FTG-}\mathcal{M} \text{od}_{\Delta_{\text{min}}}^{\Delta} (\mathcal{T}h) \), such that there is no other \( \Delta \)-FTG model that satisfies only a strict subset of the ground atoms satisfied by \( \text{FTG-}\mathcal{M} \text{od}_{\Delta_{\text{min}}}^{\Delta} (\mathcal{T}h) \).

Such a minimal model exists if for any \( t \in \mathcal{F}_2 \), there exists \( t' \in \Delta \) such that \( \mathcal{T}h \models t = t' \). Indeed, if for any \( t \in \mathcal{F}_2 \), there exists \( t' \in \Delta \) such that \( \mathcal{T}h \models t = t' \) then there exists at least one \( \Delta \)-FTG algebra model of \( \mathcal{T}h \). Since each \( \Delta \)-FTG algebra has a finite domain of cardinality \( \text{card}(\Delta) \), there is a finite number of ground atoms valid in this algebra (we only consider ground atoms built with terms in \( \Delta \) knowing that if an atom contains a term \( t \notin \Delta \), it suffices to replace \( t \) by the term \( t' \) such that \( \mathcal{T}h \models t = t' \). More precisely, there exist \( \text{card}(\Delta) \times (\text{card}(\Delta) - 1) + \sum_{p \in \Phi} \text{card}(\Delta)^{ar(p)} \) \( \Delta \)-FTG models of \( \mathcal{T}h \). Thus, by representing each finite term generated model by the set of ground atoms it satisfies, it is easy to see that \( \mathcal{A}_{\text{min}} = \bigcap_{\mathcal{A} \in \text{FTG-}\mathcal{M} \text{od}_{\Delta_{\text{min}}}^{\Delta} (\mathcal{T}h)} \mathcal{A} \) is a \( \Delta \)-FTG algebra and that it is a model of \( \mathcal{T}h \), since \( \mathcal{T}h \) contains only definite Horn clauses. Such \( \mathcal{A}_{\text{min}} \) is the minimal model \( \text{FTG-}\mathcal{M} \text{od}_{\Delta_{\text{min}}}^{\Delta} (\mathcal{T}h) \).

**Definition 13.** [Presentation of a specification, of an environment] Given a specification \( \text{SP} = (\Sigma \cup \Sigma_E, \mathcal{T}, \Delta) \), we call a presentation of \( \text{SP} \) any tuple \( \langle \mathcal{SP} \rangle = (\Sigma \cup \Sigma_E, \mathcal{T}_{\text{con}} \cup \mathcal{T}_{\text{ded}}, \Delta) \) such that

- \( \mathcal{T}_{\text{con}} \) is a set of definite Horn clauses containing only atoms of the form \( p(t_1, \ldots, t_n) \) where for all \( i \in \{1..n\}, t_i \) is either a variable or a ground term,
Moreover, given a presentation \( \langle \mathcal{S} \mathcal{P} \rangle \) of \( \mathcal{SP} \), we call presentation \( \text{w.r.t} \ \langle \mathcal{S} \mathcal{P} \rangle \) of the \( \mathcal{SP} \)-environment \( e \) the pair:

\[
\langle e \rangle = (\mathcal{B}, \mathcal{R})
\]

such that

- \( \mathcal{B} \) is a set of ground atoms called \textit{base of facts},
- \( \mathcal{R} \) is a convergent UCTRS called \textit{interpretation of functions} whose normal forms belong to \( \Delta \),
- \( e = \text{FTG-od}_{\min}^\Delta (\mathcal{B} \cup \mathcal{T}_\text{cons} \cup \{ t = t' \mid t, t' \in \mathcal{F}_\Delta \text{ and } t' = t \downarrow_{\mathcal{S}} \}) \)

Just assuming that every function in the signature has a computable interpretation, every specification has indeed a presentation as above.

**Proposition 14.** [Existence of a presentation] Given a presentation \( \langle \mathcal{S} \mathcal{P} \rangle \) and an \( \mathcal{SP} \)-environment \( e \), there always exists a presentation of \( e \) \( \text{w.r.t} \ \langle \mathcal{S} \mathcal{P} \rangle \), provided that for any \( f \in \mathcal{F} \), \( f_e \) is computable.

**Proof.** Since \( e \) is finite and term generated, we can build \( \mathcal{B} \) as the finite set of all ground atoms valid in \( e \) in which only terms of \( \Delta \) occur, and \( \mathcal{R} = \{ (f(t_1, \ldots , t_n) \rightarrow t) \mid f_e(t_1, \ldots , t_n) = t, t_1, \ldots , t_n \in \Delta \} \).

The construction given above is rather naive and inefficient in the sense that it considers explicitly every ground atom built from \( \Delta \). The aim is to take into account \( \mathcal{T}_\text{cons} \) and to build a suitable \( \mathcal{R} \) in order to obtain a smaller set \( \mathcal{B} \). For example, given a relation \( r \) closed under a property (e.g. transitivity), it is more natural to express the smallest subset of \( r \) whose (transitive) closure is \( r \) as the base of facts and to add the closure property (transitivity) in \( \mathcal{T}_\text{cons} \), instead of enumerating all elements in relation by \( r \) in the base of facts.

**Example 15.** From the specification \( \mathcal{SP}_{BLP} \{ e \} \), we can define the presentation \( \langle \mathcal{SP}_{BLP} \{ e \} \rangle = (\Sigma_{BLP} \cup \Sigma_{Ev} \cup \mathcal{T}_{BLP}^{\text{cons}} \cup \mathcal{T}_{BLP}^{\text{ded}}, \Delta_{\mathcal{S}}) \) where:

\[
\begin{align*}
\mathcal{T}_{BLP}^{\text{cons}} &= \{ \\
&\forall x \inf f(x,x) \\
&\forall x, y, z \ inf f(x,y) \land \ inf f(y,z) \Rightarrow \ inf f(x,z) \\
&\forall x, y, z \ inf f(x,y) \land \ inf f(y,x) \Rightarrow x = y \\
&\forall x, y, z \exists w (\inf f(x,z) \land \inf f(y,z) \land \forall v (\inf f(w,x) \land \inf f(w,y) \Rightarrow \inf f(z,w))) \\
&\forall x, y, z \exists w (\inf f(x,z) \land \inf f(y,z) \land \forall v (\inf f(x,w) \land \inf f(y,w) \Rightarrow \inf f(z,w)))
\end{align*}
\]

\[
\begin{align*}
\mathcal{T}_{BLP}^{\text{ded}} &= \{ \\
&\inf f(\text{Confidential}, \text{Secret}), \inf f(L_1, \text{Confidential}), \inf f(L_2, \text{Confidential}), \\
&\inf f(\text{Public}, L_1), \inf f(\text{Public}, L_2), \inf f(\text{Sanitized}, \text{Public}) \\
&\{ f_1(\text{Bob}), \text{Secret} \} \\
&\{ f_1(\text{Alice}), L_2 \} \\
&\{ f_1(\text{Charlie}), \text{Sanitized} \} \\
&\{ f_0(\text{File}_1), \text{Confidential} \} \\
&\{ f_0(\text{File}_2), L_1 \} \\
&\{ f_0(\text{File}_3), L_2 \}
\end{align*}
\]

Now, if we consider the environment \( e_{BLP} \) introduced in Example 11, its presentation can be defined as \( \langle e_{BLP} \rangle = (\mathcal{B}_{e_{BLP}}, \mathcal{R}_{e_{BLP}}) \) where:

\[
\begin{align*}
\mathcal{B}_{e_{BLP}} &= \{ \\
&\inf f(\text{Confidential}, \text{Secret}), \inf f(L_1, \text{Confidential}), \inf f(L_2, \text{Confidential}), \\
&\inf f(\text{Public}, L_1), \inf f(\text{Public}, L_2), \inf f(\text{Sanitized}, \text{Public}) \\
&\{ f_1(\text{Bob}), \text{Secret} \} \\
&\{ f_1(\text{Alice}), L_2 \} \\
&\{ f_1(\text{Charlie}), \text{Sanitized} \} \\
&\{ f_0(\text{File}_1), \text{Confidential} \} \\
&\{ f_0(\text{File}_2), L_1 \} \\
&\{ f_0(\text{File}_3), L_2 \}
\end{align*}
\]

To show the usefulness of using a UCTRS to interpret function symbols, let us assume that the security level of a subject depends on all objects it has read: for example, it is the infimum of all these
objects (provided there is at least one object). In such a case, we add to \( \mathcal{R}_{\text{ref}} \) the following rule:

\[
\begin{align*}
(f_s(s), \varphi) &= \left( \exists o, m(s, o, r) \\
&\land \forall o, (m(s, o, r) \Rightarrow \inf(l_{\text{inf}}, f_o(o))) \\
&\land \forall l, (\forall o, m(s, o, r) \Rightarrow \inf(l, f_o(o))) \Rightarrow \inf(l, l_{\text{inf}}) \right), l_{\text{inf}}
\end{align*}
\]

This rule must be applied in priority, so is added as the first rule of the ordered UCTRS \( \mathcal{R}_{\text{ref}} \). To ensure that \( \mathcal{R}_{\text{ref}} \) is still convergent, it is sufficient to prove that for any \( e \in \mathcal{E}_{\text{SP}} \), the set of e-solutions of \( \varphi \) contains at most one element. The formula \( \forall x, y \exists z \left( \inf(z, x) \land \inf(z, y) \land \forall w \left( \inf(w, x) \land \inf(w, y) \Rightarrow \inf(w, z) \right) \right) \) belonging to \( \mathcal{B}_{\text{BLP}} \) allow us to do such a proof.

**Proposition 16.** Given an SP-environment \( e \), if there exists a presentation \( \langle e \rangle = (B, B) \) of \( e \) w.r.t. a presentation \( \langle \Sigma \rangle = (\Sigma \cup \Sigma_{\text{Ev}}, \mathcal{T}_{\text{cons}} \cup \mathcal{T}_{\text{ded}}, \Delta) \), then the set of all e-solutions of \( \varphi \) is computable for all \( \varphi \in \mathcal{F}_{\text{or}} \).

**Proof.** We give a proof by induction over \( \mathcal{F}_{\text{or}} \).

- \( e \models \sigma(p(t_1, \ldots, t_n)) \) iff \( \sigma(p(t_1, \ldots, t_n)) \downarrow^e \in B \) or there exists \( (l_1 \land \ldots \land l_n) \Rightarrow l_{n+1} \in \mathcal{T}_{\text{cons}} \) such that \( e \models \sigma(l_i) \) for all \( i \in [1..n] \) and \( \sigma(p(t_1, \ldots, t_n)) \downarrow^e = \sigma(l_{n+1}) \downarrow^e \). Thus, if \( \varphi \) is an atom, then \( \mathcal{O}l_c(\varphi) \) is computable since \( \downarrow^e \) is convergent and the set of possible substitutions is finite (there are \( \text{card}(\Delta) \text{card}(\text{Var}(\varphi)) \) possible substitutions).

- \( e \models \sigma(\neg \varphi) \) iff \( e \not\models \sigma(\varphi) \). Thus, if \( \mathcal{O}l_c(\varphi) \) is computable, then \( \mathcal{O}l_c(\neg \varphi) \) is computable too and equals to the complement of \( \mathcal{O}l_c(\varphi) \).

- If \( \mathcal{O}l_c(\varphi_1) \) and \( \mathcal{O}l_c(\varphi_2) \) are computable, then \( \mathcal{O}l_c(\varphi_1 \land \varphi_2) \) and \( \mathcal{O}l_c(\varphi_1 \lor \varphi_2) \) are obviously computable and respectively equal to \( \mathcal{O}l_c(\varphi_1) \cap \mathcal{O}l_c(\varphi_2) \) and \( \mathcal{O}l_c(\varphi_1) \cup \mathcal{O}l_c(\varphi_2) \).

- Finally, we have

\[
\mathcal{O}l_c(\forall x \varphi) = \bigcap_{t \in \Delta} \mathcal{O}l_c(\{ x \rightarrow t \}(\varphi))
\]

and

\[
\mathcal{O}l_c(\exists x \varphi) = \bigcup_{t \in \Delta} \mathcal{O}l_c(\{ x \rightarrow t \}(\varphi))
\]

### 3.3 Systems

A system is defined by a specification, a set of initial environments and a transition relation that defines the way environments evolve.

**Definition 17.** A system is a tuple \( \mathcal{S} = (\Sigma, \mathcal{S}_{\text{ini}}, \Delta) \) where:

- \( \Sigma \) is a set of \( \Sigma \)-formulae called initial conditions of \( \mathcal{S} \).
- \( \Delta \) is a functional and total relation over \( \mathcal{E}_{\text{SP}} \times \mathcal{L}_{\text{SP}} \times \mathcal{E}_{\text{SP}} \) called transition. We note \( (e, (q, d), e') \in \Delta \) and \( e' = \Delta(e, (q, d)) \) or \( e \xrightarrow{(q,d)} e' \).

In this notion of system, we capture the evolution of a set of initial environments when a query is asked and a decision is taken. The set of initial conditions \( \mathcal{S}_{\text{ini}} \) induces the set \( \mathcal{E}_{\text{SP}} \) of initial environments which are all environments models of \( \mathcal{S}_{\text{ini}} \), that is to say \( \mathcal{E}_{\text{SP}} = \mathcal{E}_{\text{SP}} \cap \mathcal{M}_{\text{od}}(\mathcal{S}_{\text{ini}}) \).

---

1. \( \forall e, e', e'' \in \mathcal{E}_{\text{SP}}, (q, d) \in \mathcal{L}_{\text{SP}}, [(e, (q, d), e') \in \Delta \land (e, (q, d), e'') \in \Delta] \Rightarrow e' = e'' \)
2. \( \forall (e, (q, d)) \in \mathcal{E}_{\text{SP}} \times \mathcal{L}_{\text{SP}}, \exists e' \in \mathcal{E}_{\text{SP}}, (e, (q, d), e') \in \Delta \)
From any initial environment \( e_0 \in \mathcal{E} \), the directed labelled graph starting from \( e_0 \) and whose labelled edges are steps \( e \xrightarrow{\gamma} e' \), \( e' \) represents all possible evolutions of the environment of the system. We note \( \delta^* \) the natural extension of \( \delta \) to \( \mathcal{E} \times \mathcal{L}_{\mathcal{SP}}^* \times \mathcal{E} \mathcal{SP}^* \). From the set \( \mathcal{E} \mathcal{SP} \) of all possible initial environments and a system \( \mathcal{S} \), we define the set of reachable environments as:

\[
\Gamma(\mathcal{S}) = \{ e \in \mathcal{E} \mathcal{SP} \mid \exists (e_0, \gamma) \in \mathcal{E} \mathcal{SP} \times \mathcal{L}_{\mathcal{SP}}^*, (e_0, \gamma, e) \in \delta^* \}.
\]

We dispose so far of a way to express a specification and corresponding environments and we need now a way to express the transition relation. For this, we introduce a relation between algebras which can be seen as a closeness relation.

**Definition 18.** [Distance between environments] Given a specification \( \mathcal{SP} \), we define a distance \( \mu \) between \( \mathcal{SP} \) environments as follows:

\[
\mu(e, e') = \{ \varphi \text{ ground atom} \mid (e \models \varphi \text{ and } e' \not\models \varphi) \text{ or } (e \not\models \varphi \text{ and } e' \models \varphi) \}
\]

for any \( e, e' \in \mathcal{E} \mathcal{SP} \).

Indeed, it is natural to call \( \mu \) a distance because for any \( e, e', e'' \in \mathcal{E} \mathcal{SP} \): \( \mu(e, e') = \mu(e', e) \), \( \mu(e, e'') \subseteq \mu(e, e') \cup \mu(e', e'') \) and \( \mu(e, e') = \emptyset \) iff \( e = e' \).

**Definition 19.** [\( \varphi \)-closest environment] Given a specification \( \mathcal{SP} \) and a formula \( \varphi \), we call set of \( \varphi \)-closest environments to \( e \) the set of environments \( e \varphi = \{ e_1, \ldots, e_n \} \) such that:

- \( \forall i \in [1..n], e_i \models \varphi \)
- \( \forall e' \in \mathcal{E} \mathcal{SP} \text{ such that } e' \models \varphi, \exists i \in [1..n] \text{ such that } \mu(e, e') \supseteq \mu(e_i, e') \)

We can obviously see that if \( \varphi \) is semantically equivalent to a formula of the form \( \varphi_{\text{pre}} \Rightarrow \varphi_{\text{goal}} \) where \( \varphi_{\text{goal}} \) is a literal containing no quantified variable, then there exists \( e' \in e \varphi \) such that \( e' \) is exactly the algebra such that:

- if \( \varphi_{\text{goal}} = p(t_1, \ldots, t_n) \), then \( \forall f \in \mathcal{F}, f \models f_e, \forall q \in \mathcal{P} \setminus \{ p \}, q_e = q_e' \text{ and } p_e = p_e \cup \left( \bigcup_{\sigma \in \mathcal{OL}(\varphi_{\text{pre}})} (\sigma(t_1), \ldots, \sigma(t_n)) \right) \)
- if \( \varphi_{\text{goal}} = \neg p(t_1, \ldots, t_n) \), then \( \forall f \in \mathcal{F}, f \models f_e, \forall q \in \mathcal{P} \setminus \{ p \}, q_e = q_e' \text{ and } p_e = p_e \setminus \left( \bigcup_{\sigma \in \mathcal{OL}(\varphi_{\text{pre}})} (\sigma(t_1), \ldots, \sigma(t_n)) \right) \)
- if \( \varphi_{\text{goal}} \) is of the form \( f(t_1, \ldots, t_n) = t \), then \( \forall q \in \mathcal{P} \), \( q_e = q_e' \), \( \forall g \in \mathcal{P} \setminus \{ f \}, g_e = g_e' \text{, } f_e \text{ is the function which associates } \sigma(t_e) \text{ to } (\sigma(t_1), \ldots, \sigma(t_n)) \text{ for any } \sigma \in \mathcal{OL}(\varphi_{\text{pre}}) \text{, and } f_e(t_1', \ldots, t_n') \text{ to other tuple } (t_1', \ldots, t_n') \in \Delta^n \).

Such an algebra \( e' \) will be denoted by \( e(\varphi_{\text{pre}} \Rightarrow \varphi_{\text{goal}}) \) and is said to be the \( \varphi \)-closest environment to \( e \) headed for \( \varphi_{\text{goal}} \).

**Example 20.** From the extended specification \( \mathcal{SP}_{\text{BLP}}[\mathcal{E}] \) and an arbitrary set \( \mathcal{S} \text{it}_{\text{BLP}} \) of axioms, we can define the system \( \mathcal{S}_{\text{BLP}} = (\mathcal{SP}_{\text{BLP}}[\mathcal{E}], \mathcal{S} \text{it}_{\text{BLP}}, \delta_{\text{BLP}}) \) where \( \delta_{\text{BLP}} \) is defined as follows:

\[
\forall e \in \mathcal{E} \mathcal{SP},
\begin{align*}
  e \xrightarrow{\text{ask}(s,o,a), \text{permit}} & \mathcal{E} \mathcal{SP} e[\text{m}(s,o,a)] \\
  e \xrightarrow{\text{release}(s,o,a), \text{permit}} & \mathcal{E} \mathcal{SP} e[-\text{m}(s,o,a)] \\
  e \xrightarrow{\text{deny}} & \mathcal{E} \mathcal{SP} e
\end{align*}
\]

Note that \( \delta_{\text{BLP}} \) is a functional and total relation. For \( \mathcal{S} \text{it}_{\text{BLP}} \) we can take for example the following set of formulae expressing that initially there are no accesses, and specifying the lattice of security...
levels:

\[ \mathcal{S} \text{init}_{BLP} = \left\{ \begin{array}{l}
\forall x_s, x_o, x_a \neg m(x_s, x_o, x_a) \\
\inf_{\text{Confidential, Secret}} \in f(L_1, \text{Confidential}), \inf_{L_1, \text{Confidential}}, \\
\inf_{\text{Public}, L_1}, \inf_{\text{Public}, L_2}, \inf_{\text{Sanitized, Public}} 
\end{array} \right\} \]

3.4 Security Policies

As we said, a policy defines the decision that answers a request performed into a given environment. These decisions depend on the corresponding request and on the environment in which the request is performed. Indeed, the environments could be rather complex; in particular the subjects and objects that are usually involved in an access control policy are part of the environment.

**Definition 21.** [Security policy] A security policy is a pair \( \mathcal{P} = (SP, Rules) \) consisting of a specification \( SP = (\Sigma \cup \Sigma_{Ev}, \top, \Delta) \) and a relation \( Rules \) over \( For_{\Sigma} \times L \times D \).

Given a policy \( \mathcal{P} = (SP, Rules) \), for each tuple \( r = (\varphi_{pre}, q, d) \in Rules \), \( \varphi_{pre} \) is called precondition of \( r \). For any query \( q \), we denote by \( Pre(q) \) the set

\[ \{ \varphi_{pre} \in For_{\Sigma} \mid \exists d, (\varphi_{pre}, q, d) \in Rules \} \]

**Example 22.** The mandatory part of the BLP policy \( \mathcal{P}_{BLP} = (SP_{BLP}[\mathcal{E}], Rules_{BLP}) \) is defined as follows:

- Every subject can read an object whose level of security is less than its level of security iff it does not write into an object of a lower security level:

  \( (\psi_1, \text{ask}(s, o, \text{read}), \text{permit}) \in Rules_{BLP} \) and
  \( (-\psi_1, \text{ask}(s, o, \text{read}), \text{deny}) \in Rules_{BLP} \)

  where \( \psi_1 = \inf_{f_o(o), f_s(s)} \wedge [\forall o' m(s, o', \text{write}) \Rightarrow \inf_{f_o(o'), f_o(o')}] \).

- Every subject can write into an object if it does not read another object with a higher security level.

  \( (\psi_3, \text{ask}(s, o, \text{write}), \text{permit}) \in Rules_{BLP} \) and
  \( (-\psi_3, \text{ask}(s, o, \text{write}), \text{deny}) \in Rules_{BLP} \)

  where \( \psi_3 = \forall o' m(s, o', \text{read}) \Rightarrow \inf_{f_o(o'), f_o(o')} \).

- Every subject can release any of its accesses.

  \( (\psi_5, \text{release}(s, o, a), \text{permit}) \in Rules_{BLP} \) and
  \( (-\psi_5, \text{release}(s, o, a), \text{deny}) \in Rules_{BLP} \)

  where \( \psi_5 = m(s, o, a) \).

Independently of the system a policy is applied on, we generally require that for every query, the policy eventually gives a decision and only one.
DEFINITION 23. A security policy \( \varphi = (\mathcal{S}P, \mathcal{R}ules) \) is said to be **total** iff for any \( \mathcal{S}P \)-environment \( e \) and any query \( q \), there always exists at least one decision \( d \) such as there is an element \( (\varphi_{pre}, q, d) \) in \( \mathcal{R}ules \) such that \( e \models \varphi_{pre} \). This definition can be expressed as follows: \( \forall q \in \mathcal{D}_{Ev}, \mathcal{T} \models \mathcal{V}_{e \in \mathcal{P}re(q)} \varphi \). Moreover, \( \varphi \) is said **consistent** iff for any \( \mathcal{S}P \)-environment \( e \) and any query \( q \), if \( (\varphi_{pre}, q, d) \) and \( (\varphi_{pre}', q, d') \) are in \( \mathcal{R}ules \) and \( e \) satisfies \( \varphi_{pre} \) and \( \varphi_{pre}' \), then \( d = d' \).

One can notice that the conditions used in the assertions of the policy defined in Example 22 are both exhaustive (for all \( q \in \mathcal{D}_{Ev}, \mathcal{V}_{e \in \mathcal{P}re(q)} \varphi \) is a theorem, i.e. always valid) and exclusive (for any \( q \in \mathcal{D}_{Ev}, \forall \varphi \neq \varphi' \in \mathcal{P}re(q), \{ \varphi, \varphi' \} \) is contradictory); thus it can be proved that \( \varphi_{BLP} \) is total and consistent.

As for environments, we define now a presentation of a policy in which the evaluation of queries into decisions is performed using constraint rewriting.

DEFINITION 24. **[Presentation of a policy]** Given a presentation of a specification \( (\mathcal{S}P) \), we call **presentation** of a policy \( \varphi = (\mathcal{S}P, \mathcal{R}ules) \) w.r.t \( (\mathcal{S}P) \) any UCTRS denoted by \( (\varphi) \) such that:

- the set of normal forms of \( (\varphi) \) is exactly \( \mathcal{D} \)
- any rule of \( (\varphi) \) is of the form \( \varphi \rightarrow \text{rhs} \) such that \( \varphi \in \mathcal{F}or_{\Sigma} \)
- for any \( e \in \mathcal{D} \cap (\mathcal{S}P) \), \( q \rightarrow_{(\varphi)} d \) iff there exists \( (\varphi_{pre}, q, d) \) in \( \mathcal{R}ules \) such that \( e \models \varphi_{pre} \).

EXAMPLE 25. The presentation of the policy \( \varphi_{BLP} = (\mathcal{S}P_{BLP}, \mathcal{R}ules_{BLP}) \) of Example 22 w.r.t. the presentation \( (\mathcal{S}P_{BLP}[\mathcal{E}]) \) given in Example 15 can be defined as follows:

\[
\langle \varphi \rangle_{BLP} = \begin{cases} 
\text{ask}(s,o,r) & \frac{\inf(f_o(a),f_r(s)) (m(s,o',x) \Rightarrow \inf(f_o(a),f_r(o')))}{x_q} \quad \text{permit} \\
\text{ask}(s,o,w) & \frac{\langle \alpha' \rangle m(s,o',x) \Rightarrow \inf(f_o(a),f_r(o'))}{m(s,o,a)} \quad \text{permit} \\
\text{release}(s,o,a) & m(s,o,a) \quad \text{deny} \\
\end{cases}
\]

where \( x_q \) is a variable of sort \( Q \). This example is quite simple in that the constraint of a rewrite rule \( q \rightarrow \hat{\text{lhs}} \) \( d \) corresponds exactly to a precondition in \( (\varphi_{pre}, q, d) \in \mathcal{R}ules \). This is not always the case. Indeed, the definition of a presentation just requires that the relation over \( \mathcal{D} \times \mathcal{D} \) induced by \( \vdash_{(\varphi)} \) corresponds to the relation “\( \forall q \in \mathcal{P}re(q) \) such that \( e \models \varphi \) and \( (\varphi, q, d) \in \mathcal{R}ules \)”. In the previous example it is obvious, but we could have a more complicated situation as the following one: let us consider that a writing access is allowed only if the corresponding reading access is permitted. In such a case, we would have a rule of the form \( \text{ask}(s,o,w) \rightarrow \text{ask}(s,o,r) \).

As in [DKKS07], the properties of totality and consistency of a policy \( \varphi = (\mathcal{S}P, \mathcal{R}ules) \) should be expressed as properties of termination, completeness and confluence of its presentation \( (\varphi) \). We leave that for further work.

### 3.5 Policy over system

A system only specifies what happens when an action is performed (according to a request and a decision). A policy on such a system imposes restrictions on the reachable environments, exactly as strategies impose restrictions on reachable elements in a derivation tree or a labelled transition system. Of course, a policy cannot be applied on any system but only on those whose specification
is “more general” than that of the policy. We define thus the notion of specification subsumption and use it to characterize policies compatible with a given systems.

**Definition 26.** Given two specifications \( \mathcal{SP} = (\Sigma \cup \Sigma_{Ev}, T, \Delta) \) and \( \mathcal{SP}' = (\Sigma' \cup \Sigma_{Ev}', T', \Delta') \), we say that \( \mathcal{SP} \) subsumes \( \mathcal{SP}' \) and we note \( \mathcal{SP} \preceq \mathcal{SP}' \) iff:

\[
\Sigma' \subseteq \Sigma, \text{ and } \Sigma_{Ev}' \subseteq \Sigma_{Ev}, \text{ and } T' \subseteq T \text{ (or more generally } T \models T'), \text{ and } \Delta' \subseteq \Delta.
\]

We say that a system \( \mathcal{S} = (\mathcal{SP}, \mathcal{S}nit, \delta) \) and a policy \( \varphi = (\mathcal{SP}_\varphi, \mathcal{Rules}_\varphi) \) are compatible iff \( \mathcal{SP} \preceq \mathcal{SP}_\varphi \). In such a case, the \( \varphi \)-secure system build from \( \mathcal{S} \) is obtained by restricting the transition relation of \( \mathcal{S} \).

We can now define systems constrained by policies.

**Definition 27.** For each compatible system \( \mathcal{S} = (\mathcal{SP}_\mathcal{S}, \mathcal{S}nit, \delta) \) and policy \( \varphi = (\mathcal{SP}_\varphi, \mathcal{Rules}_\varphi) \) the \( \varphi \)-secure system build from \( \mathcal{S} \) is defined as follows:

\[
\mathcal{S}_{\mid \varphi} = (\mathcal{SP}_\mathcal{S}, \mathcal{S}nit, \delta_{\varphi})
\]

where \( \delta_{\varphi} \) is the relation containing each \( (e, \gamma, e') \in \delta \) such that

\[
\begin{align*}
\gamma &= (q, d) \in \mathcal{SP}_\mathcal{S} \setminus \mathcal{SP}_\varphi \text{ or else} \\
\exists \phi_{\text{pre}}(q, d) \in \mathcal{Rules} \text{ such that } e \models \phi_{\text{pre}}
\end{align*}
\]

**Example 28.** If we consider again our running example, we can define the secure system \( \mathcal{S}_{\text{BLP}} = (\mathcal{SP}_{\text{BLP}}, \mathcal{S}nit_{\text{BLP}}, \delta_{\text{BLP}}) \).

**Definition 29.** Given a system \( \mathcal{S} = (\mathcal{SP}_\mathcal{S}, \mathcal{S}nit, \delta) \), a compatible policy \( \varphi = (\mathcal{SP}_\varphi, \mathcal{Rules}_\varphi) \), and a presentation of all considered objects, we define the transition relation of the \( \varphi \)-secure system build from \( \mathcal{S} \) as follows:

\[
(e, (q, d), e') \in \delta_{\varphi} \text{ iff } (e, (q, d), e') \in \delta \text{ and } q \rightarrow_{\varphi} d \text{ where } \tilde{\mathcal{R}} = \mathcal{R} \cup \{ \varphi \} \text{ and } (e) = (\mathcal{B}, \mathcal{B}).
\]

**Example 30.** Let us consider an example proposed in [SdO08, KKSdO08] and coming from the NetFilter\(^4\) documentation. More precisely, we consider a firewall whose aim is to block any traffic coming from the public network to the private network. As usual, we denote by ppp0 the interface associated to Internet connections. The private network consists of two local machines whose IP addresses are 10.1.1.1 and 10.1.1.2 and are rewritten into a same external IP address 123.123.123.1. This network address translation is part of the system which also contains the set of all connections established. Thus, we have the specification presentation \( \langle \mathcal{SP} \rangle = (\Sigma \cup \Sigma_{Ev}, \varnothing, \Delta) \) with:

\[
\mathcal{F} = \begin{cases}
\text{pkt} & : \text{Address} \times \text{Address} \rightarrow \text{Packet} \\
\text{eth0, ppp0} & : \rightarrow \text{Address} \\
[0,255] \times [0,255] \times [0,255] \times [0,255] & : \rightarrow \text{Address}
\end{cases}
\]

\[
\varphi = \begin{cases}
\text{ Established } & : \text{Packet} \\
\mathcal{F}_{Ev} = \begin{cases}
\text{filter} & : \text{Packet} \rightarrow Q \\
\text{drop,accept} & : \rightarrow D
\end{cases}
\end{cases}
\]

\(^1\)The set \( \mathcal{R}'' = \mathcal{R} \cup \{ \varphi \} \) is ordered such that rules belonging to \( \mathcal{R} \) are always “applied” before rules of \( \langle \varphi \rangle \).

\(^4\)http://www.netfilter.org
and $\Delta$ is the finite set of ground terms of sort Packet. The system $\mathcal{S}$ is given by $\mathcal{SP}$, the initial environment

$$
(e_0) = \langle \emptyset, \mathcal{R}_0 = \left\{ \begin{array}{l}
\text{pkt}(10.1.1.1, ppp0), \text{pkt}(123.123.1.1, ppp0) \\
\text{pkt}(10.1.1.2, ppp0), \text{pkt}(123.123.1.1, ppp0)
\end{array} \right\} \rangle
$$

consisting of the initial network address translations and specifying the fact that initially no connection is established, as well as the following transition relation:

\[
\begin{align*}
& e \xrightarrow{(\text{filter}(x), \text{accept})} e' \quad e' \xrightarrow{\text{Established}(x)} \\
& e \xrightarrow{(\text{filter}(x), \text{drop})} e \\
\end{align*}
\]

The following presentation represents the simple policy described above:

\[
(\wp) = \left\{ \begin{array}{l}
\text{filter}(\text{pkt}(\text{src}, \text{dst})) \quad \text{Established}(\text{pkt}(\text{src}, \text{dst})) \rightarrow \text{accept} \\
\text{filter}(\text{pkt}(\text{eth0}, \text{dst})) \quad \neg \text{Established}(\text{pkt}(\text{eth0}, \text{dst})) \rightarrow \text{accept} \\
\text{filter}(\text{pkt}(\text{ppp}0, \text{dst})) \quad \neg \text{Established}(\text{pkt}(\text{ppp}0, \text{dst})) \rightarrow \text{drop} \\
\text{filter}(\text{pkt}(123.123.1.1, \text{dst})) \quad \neg \text{Established}(\text{pkt}(123.123.1.1, \text{dst})) \rightarrow \text{accept} \\
\text{filter}(\text{other}) \\
\end{array} \right\}
\]

Thus, the $\wp$-secure system built from $\mathcal{S}$ contains the following transitions:

\[
\begin{align*}
& e_0 \xrightarrow{\text{filter}(\text{pkt}(10.1.1.1, \text{ppp}0), \text{accept})} e_1 \\
& e_1 \xrightarrow{\text{filter}(\text{ppp}0, \text{pkt}(192.168.2.124)), \text{drop}} e_1
\end{align*}
\]

with $(e_1) = (\text{Established}(\text{pkt}(10.1.1.1), \text{ppp}0), \mathcal{R}_0)$ because

\[
\text{filter}(\text{pkt}(10.1.1.1), \text{ppp}0) \xrightarrow{e_1}_{\mathcal{R}_0} \text{filter}(\text{pkt}(123.123.1.1), \text{ppp}0) \xrightarrow{e_1}_{(\wp)} \text{accept}
\]

and $\text{filter}(\text{ppp}0, \text{pkt}(192.168.2.124)) \xrightarrow{e_1} \text{drop}$.

We have shown so far how to represent in a formal way all objects related to the specification of a security policy and of the systems it can be applied on. Such a formalism allows us to reason about all these objects, but we are also interested in studying some properties which cannot be expressed only with these objects. The next section proposes a framework for performing this kind of extra reasoning.

### 4 Transformation of systems

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 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). This link is formalized in this section through a transformation of environments, whose aim is to translate an environment into another one dealing with the entities we are interested in. This transformation itself is based on the notion of morphism between two signatures.
functions such that $S$ based on different specifications. Indeed, let $E$ environments to $SP$ and suppose we want to prove a property $\theta$ we extend $\hat{\theta}$ transformation is especially interesting in access control where the system and the policy manipulate $e$ is no reachable state to check the desired properties over reachable states of $SP$ in which $\Gamma(S\hat{\theta}) = \Gamma(S\theta)$ such that $\text{an environment transformation } (\theta, \rightarrow) \text{ induces a relation } \theta \rightarrow \text{ over } E\text{nv}_{SP} \times E\text{nv}_{SP}$ defined as follows:

$e_1 \rightarrow^{\theta} e_2 \iff \forall \varphi_1, \varphi_2 \text{ s.t. } \varphi_1 \rightarrow \varphi_2, \text{ if } \alpha \in \mathcal{I} ol_{e_1}(\varphi_1) \text{ then } \theta \circ \alpha \in \mathcal{I} ol_{e_2}(\varphi_2)$

In order to obtain a functional relation $\rightarrow$, we adapt the previous one with respect to a presentation of specifications. Given two presentations $\langle SP_1 \rangle = (\Sigma_1 \cup \Sigma_{Ev_1}, T_1, \Delta_1)$ and $\langle SP_2 \rangle = (\Sigma_2 \cup \Sigma_{Ev_2}, T_2, \Delta_2)$ of $SP_1$ and $SP_2$, an environment transformation $(\theta, \rightarrow)$ whose image contains only definite Horn clauses, induces a relation $\theta \rightarrow \text{ w.r.t. } \langle SP_1 \rangle$ and $\langle SP_2 \rangle$ such that for any $e_1 \in E\text{nv}_{SP_1}$, there exists only one $e_2 \in E\text{nv}_{SP_2}$ such that $e_1 \rightarrow e_2$ and which is equal to

$$FTG_{\rightarrow}^{\Delta_2} \cap \{ e \rightarrow e' \mid \varphi_1 \rightarrow \varphi_2 \text{ and } \alpha \in \mathcal{I} ol_{e_2}(\varphi_2) \} \}$$

Environment transformations are useful to check the same properties over several secure systems based on different specifications. Indeed, let $\mathcal{S} = (SP, \exists \text{init}, \delta)$ be a system based on a specification SP and suppose we want to prove a property $\Omega$ over reachable states of $\mathcal{S}$ which cannot be expressed in $SP$. For this, we can use another specification $SP'$ in which $\Omega$ can be expressed. Then, in order to check the desired properties over reachable states of $\mathcal{S}$, it suffices to build a transformation of $SP'$-environments to $E\text{nv}_{SP'} (\theta, \rightarrow)$ where $\theta$ is a signature morphism from $\Sigma$ to $\Sigma'$ and to check that there is no reachable state $e \in \Gamma(\mathcal{S})$ such that an $SP'$-environment $e'$ verifying $e \rightarrow e'$ satisfies $\neg \Omega$. Such a transformation is especially interesting in access control where the system and the policy manipulate subjects and objects and where the aim of the policy is to control the information flow induced by accesses. In such a case it is useful to avoid the mix between what is really manipulated (accesses to files) and what one wants to reason about (information contained in files).
Example 33. In this example we illustrate environment transformations in order to deal with information flow properties of access control policies. First, we introduce the “generic” signature $Σ_{Flow} = (Ӏ_{Flow}, Φ_{Flow}, P_{Flow})$ where:

$$Φ_{Flow} = \{\text{Actor, Information}\} \quad P_{Flow} = \begin{cases} \text{Get : Actor, Information,} \\ \text{Put : Actor, Information,} \\ \text{MoveTo : Information, Information} \\ \text{Eligible : Actor, Information} \\ \text{Trustworthy : Actor, Information} \\ \text{Gflow : Information, Information} \end{cases}$$

and where $Φ_{Flow}$ 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 he 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$(s, i)$ 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 to describe flows that happen while the predicates Eligible, Trustworthy, and Gflow are used to specify flow policies (respectively a confidentiality policy, an integrity policy and a confinement policy). If we consider $Φ_{Flow}$ as the set $Σ_δ$ used to build the extended specification $SP_{BLP}[δ]$, we can define the signature morphism $θ_{BLP} = (θ_{Φ}^{BLP}, θ_{P}^{BLP})$ from $Σ_{BLP}[δ]$ to $Σ_{Flow}[δ] = (Ӏ_{Flow}, δ, P_{Flow})$ where:

- $D_{obj}(θ_{Φ}^{BLP}) = \{s, o\}$ with $θ_{Φ}^{BLP}(s) = \text{Actor}$ and $θ_{Φ}^{BLP}(o) = \text{Information}$
- $θ_{P}^{BLP}$ is the identity function:

$$\forall o \in \text{Obj} θ_{Φ}^{BLP}(o) = o : \text{Information} \quad \forall s \in \text{Subj} θ_{P}^{BLP}(s) = s : \text{Actor}$$

Now we introduce the presentation $(SP_{Flow}[δ]) = (Σ_{Flow}[δ], T_{Flow}^{cons} \cup T_{Flow}^{ded}, Δ_δ)$ of a specification where:

$$T_{Flow}^{cons} = \begin{cases} \forall o \text{MoveTo}(o, o) \\ \forall s, o, o' \text{Get}(s, o) \wedge \text{Put}(s, o') \Rightarrow \text{MoveTo}(o, o') \\ \forall s, o, o' \text{MoveTo}(o, o') \wedge \text{Get}(s, o') \Rightarrow \text{Get}(s, o) \\ \forall s, o, o' \text{MoveTo}(o, o') \wedge \text{Put}(s, o) \Rightarrow \text{Put}(s, o') \\ \forall o, o', o'' \text{MoveTo}(o, o') \wedge \text{MoveTo}(o', o'') \Rightarrow \text{MoveTo}(o, o'') \end{cases}$$

$$T_{Flow}^{ded} = \emptyset$$

Here, $T_{Flow}^{cons}$ allows to consider the transitive closure of information flows. Within such a specification, it is possible to define, in a generic way, confidentiality, integrity and confinement security properties as follows:

- Confidentiality $ψ_{conf}$: Get$(s, o) \Rightarrow \text{Eligible}(s, o)$
- Integrity $ψ_{int}$: Put$(s, o) \Rightarrow \text{Trustworthy}(s, o)$
- Confinement $ψ_{info}$: MoveTo$(o, o') \Rightarrow \text{Gflow}(o, o')$

In order to analyse information flows for the BLP policy, we define the environment transformation $(θ_{BLP}, \xrightarrow{BLP})$ where:

$$\begin{align*}
\{ m(s, o, \text{read}) & \xrightarrow{BLP} \text{Get}(s, o) \\
\text{inf}(f_o(o), f_i(s)) & \xrightarrow{BLP} \text{Eligible}(s, o) \\
m(s, o, \text{write}) & \xrightarrow{BLP} \text{Put}(s, o) \\
isf(f_o(o), f_o(o')) & \xrightarrow{BLP} \text{Gflow}(o, o')
\end{align*}$$
Such an environment transformation provides a framework allowing to check that the BLP policy ensures confidentiality and confinement. Indeed, it can be proved that:

\[ \forall e \in \Gamma(\mathcal{S}_{\text{BLP}}) \quad e \xrightarrow{\mathcal{B}L \mathcal{P}} e' \Rightarrow \models \psi_{\text{conf}} \land \psi_{\text{info}} \]

In the original paper [BL96a, BL73], the authors define the mandatory part of their policy as the two properties defined as follows:

\[ \Omega_{\text{BLP}} = \left\{ \begin{array}{ll}
\text{MAC-property} & \forall o, s \; m(s, o, \text{read}) \Rightarrow \text{inf}(f_o(o), f_s(s)) \\
\text{MAC*-property} & \forall o_1, o_2, s \; (m(s, o_1, \text{read}) \land m(s, o_2, \text{write})) \Rightarrow \text{inf}(f_o(o_1), f_o(o_2))
\end{array} \right\} \]

It can be noticed that:

\[ e \models \Omega_{\text{BLP}} \land e \xrightarrow{\mathcal{B}L \mathcal{P}} e' \Rightarrow \models \psi_{\text{conf}} \land \psi_{\text{info}} \]

Hence, \( \psi_{\text{conf}} \) and \( \psi_{\text{info}} \) can be viewed as an abstraction of MAC and MAC* that can be used to analyse other access control policies (e.g. the Chinese Wall policy). In the particular case of the BLP policy, we obtain a result similar to the well-known basic security theorem [BL96a, BL73] asserting that each environment \( e \in \Gamma(\mathcal{S}_{\text{BLP}}) \) is such that \( e \models \Omega_{\text{BLP}} \).

5 Conclusions

In this work, we have proposed an abstract formalism for the definition of systems constrained by security policies that allows a clear separation between the evolution of the system and the enforcement of the policy in the corresponding execution environment. Environments are formalized as algebras with finite domains and a system is defined as a labelled transition system transforming environments. The pairs (query, decision) used as labels of the corresponding transition systems describe the interaction between the system and the policy that defines the decisions to be taken for a request in a given environment. We specified the conditions a policy should satisfy to be applied to a system and we showed how systems that are secure with respect to a policy can be defined.

The environments as well as the security policies can be defined in an abstract way or using a more concrete presentation. More precisely, rewrite systems are used in a standard way to perform function evaluation in the considered environments. Moreover, given a system and a compatible policy, constrained term rewriting systems are used to combine the necessary check for values in the current environment (performed through constraint solving) with the computation of a decision (performed through rewriting). This approach clarifies and improves previous works by a more appropriate treatment of the environment in which a security policy is applied.

The formalism we have proposed provides a unified representation of all objects related to the specification of a security policy and of the systems it can be applied on. We also propose a transformation of environments that allows the translation of an environment into another one and thus the possibility to study some properties which cannot be expressed only within the initial presentation. We have shown, in particular, how confidentiality, integrity and confinement can be expressed for the BLP policy that does not deal explicitly with information flows but only with objects containing tractable information. This provides some evidence that the proposed framework is also adequate to handle a large class of policies and to formalize a certain comparison between policies.
5.1 Related work

An important part of security is brought by access control whose aim is to prevent resources from unauthorized accesses. An access control policy is usually presented as a set of rules specifying that a given subject is allowed (or denied) to perform an action on a given object. Different access control models have been proposed to specify the authorization mechanism: discretionary [HRU76] and mandatory [Bib75, BL96b] access control models, role-based access control (RBAC) [SCFY96], organization-based access control (OrBAC) [KBB03], Chinese-Wall policy [BN89]... Such models are useful to analyze whether some security property (such as confidentiality or separation of duties) is satisfied in all possible states of the system.

Even in more general contexts than access control, policies are often expressed by rules in natural language: if some conditions are satisfied, then, grant or deny some request. Not surprisingly, their specification and verification can rely on various logic-based formalisms. The existing approaches have adopted different semantic frameworks such as first-order logic [HW03, JSSS01, HJM08, JM06], Datalog [BS02, LM03, BO05, DFK06], temporal logic [BBFS98, CCBS05], deontic logic [CC97, KBB03], or defeasible logic [MAG04]. In open and distributed property-based access control systems, tasks are delegated and agents must rely on others to perform actions that they themselves cannot do. The concept of trust has been formalized and models of trust together with logical formalisms have been proposed, such as for instance Wooldridge’s Logic Of Rational Agents [JJ06]. These formalisms provide different degrees of expressiveness for authoring a policy.

In recent years, term rewriting theory has been applied for the specification and analysis of security policy in [DKKS07, BF08b, BF08a, SdO08, KKSdO08]. In a rewrite-based specification, policies are expressed by rules close to natural language: if some conditions are satisfied, then, a given request is evaluated into a decision; for instance, it may be granted or denied. Moreover strategic rewriting is used to express control on the rules and to handle priorities or choices between possible decisions. The rewrite-based approach provides executable specifications for security policies, which can be independently designed, verified, and then anchored on programs [dOWKK07] using a modular discipline. Once the policies are specified as rewriting systems, some properties can be checked, such as consistency of a policy that generally corresponds to the confluence of the underlying rewriting system while the completeness depends on the termination of the corresponding rewriting system. In [CMO08], the rewriting approach has been used to explore the information flow over the reachable states of a system and detect information leakage.

5.2 Further work

The proposed framework and especially the use of constrained term rewriting systems opens the way to further research problems related to the verification of properties of a security policy and of a secure system w.r.t. the policy. For instance, we expect to give some sufficient syntactic conditions, easy to check, to ensure totality and consistency of a security policy. We also need to further explore the power of environment transformation for proving properties over reachable states in the secure system. Moreover, we are currently working on an implementation of the framework.

A challenge in the domain of security policies is to dynamically integrate different security policies in a modular way. The policies and resources of a system may be modeled, built or owned by different unrelated parties, at different times, and under different environments. Hence, in order to avoid severe interference in their individual developments, it is better for a policy to adopt
a component-based architecture. To build a complex global policy, they are integrated via various composition operators.

A first goal is to define in our formalism a composition mechanism and to identify its general properties. A second objective is to study and to formalize several ways to compose policies and systems, to compare them and to express in a mathematical setting the properties that these composition mechanisms should satisfy. We also plan to define operators over policies allowing to solve conflicts or unspecified cases that may happen during composition and to classify these operators in order to characterize operators leading to a more restrictive policy or to a less restrictive policy, by considering comparison mechanism over policies and systems. The ultimate goal is to provide a formal definition of composition in order to specify and analyze the composition of security policies, and thus to give some theoretical foundations to policies composition.

References


[KKSdO08] C. Kirchner, H. Kirchner, and A. Santana de Oliveira. Analysis of rewrite-based access control policies. In D. Dougherty and S. Escobar, editors, Security and Rewriting Techniques, 3rd International Workshop Secret’08, volume informal proceedings, sub-


