SMT-based Enforcement and Analysis of NATO Content-based Protection and Release Policies

Alessandro Armando
DIBRIS, U. of Genoa, Italy
and
Security and Trust Unit,
FBK-Irst, Trento, Italy
armando@fbk.eu

Silvio Ranise,
Riccardo Traverso
Security and Trust Unit,
FBK-Irst, Trento, Italy
ranise@fbk.eu
rtraverso@fbk.eu

Konrad Wrona
NATO Communications and
Information Agency
The Hague, Netherlands
konrad.wrona@ncia.nato.int

ABSTRACT

NATO is developing a new IT infrastructure that will enable automated information sharing between different information security domains and provide strong separation between different communities of interest while supporting dynamic and flexible enforcement of the need-to-know principle. In this context, the *Content-based Protection and Release* (*CPR*) model has been introduced to support the specification and enforcement of access control policies used in NATO and, more generally, in complex organizations. While the ability to support fine-grained security policies for a large variety of users, resources and devices is desirable, the definition, maintenance, and enforcement of these policies can be difficult, time-consuming, and error-prone. Thus, automated support for policy analysis to help designers in these activities is needed. In this paper we show that several policy-related analysis problems of practical interest can be reduced to SMT problems, we propose an effective enforcement mechanism relying on attribute-based encryption (ABE), and assess the scalability of our approach on an extensive set of synthetic benchmarks.

1. INTRODUCTION

The successful operation of NATO missions requires the effective and secure sharing of information not only among coalition partners, but also with external organizations such as the Red Cross. While making as much information as possible available to the participants involved in a mission, it is crucial to avoid the disclosure of sensitive details to users with insufficient clearance. Of course, balancing confidentiality and availability of information greatly complicates the task of information sharing: permitting access to digital resources to coalition partners is necessary to ensure their effective involvement in a mission, but disclosure to unintended recipients clearly must be avoided.

To meet these challenges, NATO is developing a new IT infrastructure that will enable automated information sharing between different information security domains while providing strong separation between different communities of interest and supporting dynamic and flexible enforcement of need-to-know principles [29]. In this context, the *Content-based Protection and Release* (*CPR*) model was introduced in [5] to support the specification and enforcement of access control policies used in NATO and, more generally, in complex organizations. The design goals of the CPR model include *(i)* supporting the specification of fine-grained security policies related to a large variety of users, resources and devices, possibly encompassing multiple security domains and *(ii)* supporting the seamless composition of policies possibly addressing different concerns. To meet goal *(i)*, CPR builds upon the Attribute-Based Access Control (ABAC) model [31], thereby overcoming the limitations of traditional access control models (e.g. RBAC [27]). Goal *(ii)* is achieved by supporting the composition of *release policies* (defining relationships between subjects and resources) and *protection policies* (describing the technical constraints, e.g. communication channels and terminals, under which the information stored in resources can be accessed). This simplifies policy specification and management, and enables more efficient implementation of the policies and procedures mandated by directives within NATO and other international/governmental organizations. Yet the task of defining, maintaining, and enforcing fine-grained policies for a complex organization is still daunting. It calls for automated support of a number of key activities occurring in the definition and management of CPR policies, such as:

- *Policy Debugging.* In order to evaluate if a newly created policy matches the intentions of the designers, scenarios should be constructed in which certain users can or cannot access resources according to some expectations.
- *Policy Analysis.* Before deploying a newly developed policy it is important to ensure that it has some fundamental (e.g. safety) properties. For instance, a group of users with a given clearance level should never be able to access resources with a given level of sensitivity.
- *Policy Enforcement.* To re-use available enforcement

ABAC'16, March 11 2016, New Orleans, LA, USA

ACM 978-1-4503-4079-3/16/03 ...$15.00
DOI: http://dx.doi.org/10.1145/2875491.2875493

mechanisms, CPR policies should be translated to common policy languages (XACML[1], ABE [9]).

CPR policies can be specified by means of a high-level policy language (*CPRL*) [5] that is currently being validated within NATO in several use-case scenarios. (A simplified version of one of these is described below.) CPRL is rooted in first-order logic and is therefore endowed with a simple yet formal semantics—as are many other logic-based access control languages [10]. CPRL policies are therefore amenable to formal analysis using automated reasoning techniques and tools. Moreover, many problems of practical interest (such as answering authorization queries and checking for subsumption between policies) can be reduced to *decidable* theorem-proving problems [5], called Satisfiability Modulo Theories (SMT) problems, which extend the Boolean satisfiability (SAT) problem by allowing for a much richer vocabulary to create formulae. Computationally, SMT problems are NP-hard since they subsume the SAT problem. In this paper, we show that automated support to the aforementioned activities can be achieved by taking advantage of state-of-the-art SMT technology. As a proof-of-concept we developed the *CPRL Tool* to help policy designers and administrators in their routine work. The tool provides automatic support to tackle a variety of problems, henceforth called *CPRL Problems*, ranging from answering queries to checking for (policy) subsumption. For a set of CPR policies and a question, the tool automatically computes an answer together with supporting evidence. For instance, if the question is "Does the CPRL policy satisfy the property that a group G of users can never access a set R of resources?" and the answer is "No", then the evidence is a scenario in which a user in group G is permitted to access a resource in R.

The three main contributions of this paper are: *(i)* we show that several policy-related problems of practical interest can be reduced to SMT problems (Section 3), *(ii)* we define an effective enforcement mechanism based on ABE, and *(iii)* we show that a scalable implementation of the above techniques can be obtained by making use of a state-of-the-art SMT solver.

The paper is organized as follows. Section 2 summarizes the main ideas underlying the CPR model and its first-order logic formalization. Section 3 discusses our approach to supporting the enforcement and analysis of CPRL policies. Section 4 discusses the prototype implementation of the CPRL Tool, and presents an experimental evaluation that shows the scalability of our approach. Related work and conclusions are presented in Section 5. Appendix A presents a use-case scenario inspired by a realistic situation.

2. THE CPR ACCESS CONTROL MODEL

Modern joint military missions rely on network-centric operations. The future of the NATO information sharing infrastructure [30] is built around an access control component that operates in an open and distributed environment. It has been observed that traditional access control models—such as discretionary (DAC), mandatory (MAC), and role-based (RBAC) models—are not always adequate in this environment [11]. The Attribute-Based Access Control (ABAC) model (see e.g. [21]) offers a powerful and unifying extension to these well-known models.

[1] `http://docs.oasis-open.org/xacml/3.0`

In ABAC, requesters are permitted or denied access to a resource based on the properties, called attributes, that may be associated to users, resources, and the context. Examples of attributes are: identity, role, and military rank of users; identifier and sensitivity of resources; and, for context, time of day and threat level. In ABAC, suitably defined attributes can represent security labels, clearances and classifications (for encoding MAC), identities and access control lists (for DAC), and roles (for RBAC). In this sense, ABAC supplements traditional access control models rather than supplanting them [21]. Policies in ABAC can be seen as conditions on the attribute values of the entities involved in an access decision. In other words, they are Boolean functions that map the attribute values of the user `u`, the resource `r`, and the context `c` to true ("permit") when `u` is entitled to get access to `r` in the context `c`, and false ("deny") otherwise.

The model underlying CPR policies refines ABAC mainly in two respects. First, in addition to the attributes of users, resources, and the context, those of terminals are considered, i.e. the capabilities of the device through which a user is trying to access a resource. Examples of terminal attributes are the hardware model, the type of encryption used to locally store data and the type of connection to the terminal (e.g. SSL). Second, the CPR (access control) policies are structured in two distinct sub-policies: a *release policy*, taking into account user, resource, and contextual attributes and a *protection policy*, taking into account resource, terminal, and contextual attributes. This enables separation of policy management roles and reflects the current procedures used within international and governmental organizations, e.g. NATO. For example, consider the situation in which a user wants to access NATO classified information. This requires, on the one hand, connecting to a network infrastructure used for processing NATO classified information. To do this, a terminal must satisfy a number of technical requirements related to hardware and software configuration that are precisely defined in NATO technical directives and guidance documents. On the other hand, the security policy governing user access to the documents stored in the network is defined in a separate set of directives and guidance documents. To summarize, a user `u` can access a resource `r` with a terminal `t` by checking if (i) the attributes of `u` and `r` satisfy the release policy and (ii) those of `r` and `t` satisfy the protection policy. If checks (i) and (ii) are both positive, "permit" is returned, otherwise the result is "deny".

2.1 Formal modelling

We regard subjects, terminals, resources, actions, and environments (called *entities*) as records whose fields are the attributes of the entities. An entity is uniquely identified by the values associated to its attributes. A *domain* is the set of values that an attribute can take. The semantics of a *CPR policy* α—regardless of the language in which it is written—is given by a collection of structures, each one composed of a universe—a non-empty set of values for the attributes—and a Boolean function (predicate) β over the values of the attributes expressing a relationship among a user `u`, a terminal `t`, a resource `r`, an action `a`, and an environment `e`. We say that user `u` can execute action `a` on resource `r` by using terminal `t` in environment `e` according to the CPR policy α iff the Boolean function β returns true when applied to the values associated to the attributes of `u`, `t`, `r`, `a`, and `e`.

We say that a structure in the semantics of a CPR pol-

icy *satisfies* the policy. We observe that structures in the semantics of a CPR policy correspond to standard interpretations in first-order logic (FOL) [13]. This allows us to use the language of FOL for writing CPR policies and standard logical techniques to compositionally express CPR policies by means of simpler logical expressions encoding protection and release policies. Formally, we will use the notion of logical theory $T = (\Sigma, \mathcal{M})$ that fixes the collection $\mathcal{M}$ of the possible interpretations of the symbols in a given set Σ. The idea is that, in certain situations, one is not interested in showing that a formula $x < y \wedge x \not< y + y$ is unsatisfiable for all possible interpretations of the symbols $<$ and $+$, but only for those interpretations in which $<$ is the usual ordering over the integers and $+$ is the addition function.

For CPRL policies, let $T_D = (\Sigma_D, \mathcal{M}_D)$ be a theory of the attributes domain characterizing their relevant algebraic properties.

We consider the attributes (fields) $a_1, ..., a_n$ of an entity (record) x as unary functions mapping x to the values of $a_1, ..., a_n$, respectively. Thus the operation of dereferencing the attribute a_i of the entity x is simply modelled as function application $a_i(x)$, also written as $x.a_i$.

From now on, we assume that $T_D = (\Sigma_D, \mathcal{M}_D)$ is given and all the formulae are built from symbols in $\Sigma_D \cup Att$, i.e. containing both the attributes Att associated to the entities and the algebraic operators Σ_D for the domains on which the attributes take values. The interpretation of the functions in Att is not constrained thus any possible value (in the appropriate domain) can be associated to each attribute of an entity, whereas the interpretation of the symbols in Σ_D is constrained by the models in $\mathcal{M}_D$. We define a *release policy expression* as a FOL formula $\rho(\mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e})$—intuitively saying that u can perform action a on resource r in environment e—in which it is possible to dereference at most the attributes of u, a, r or e (while it is prohibited to dereference the attributes of t). We define a *protection policy expression* as a FOL formula $\pi(\mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})$—intuitively saying that the action a can be performed on resource r by using terminal t in environment e—in which it is possible to dereference at most the attributes of t, a, r or e (while it is prohibited to dereference the attributes of u). A *CPR policy* $\alpha(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})$ is the conjunction of a release policy $\rho(\mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e})$ and a protection policy $\pi(\mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})$; in symbols,

$$\alpha(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \quad := \quad \rho(\mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \wedge \pi(\mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})\,. \tag{1}$$

Before defining queries, we introduce the notion of *abstraction of an entity* x as a formula $Abs(x)$ in which it is possible to dereference only the attributes of x. If entity x^* is such that $Abs(x^*)$, we say that x^* *belongs to* (or, *is in*) Abs. If x^* is the only entity such that $Abs(x^*)$ evaluates to true, we say that Abs is an *attribute assignment of* x^*. Typically, an attribute assignment has the following form:

$$x.a_1 = v_1 \wedge \cdots \wedge x.a_n = v_n \tag{2}$$

where $a_1, ..., a_n$ are *all* the attributes of the entity x and $v_1, ..., v_n$ are values in the attribute domains. An *abstract query* $Q(\mathsf{u}, \mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e}) := U(\mathsf{u}) \wedge T(\mathsf{t}) \wedge R(\mathsf{r}) \wedge A(\mathsf{a}) \wedge E(\mathsf{e})$ is a conjunction of an abstraction U of users, an abstraction T of terminals, an abstraction R of resources, an abstraction A of actions, and an abstraction E of environments. When all the abstractions in a query Q are attribute assignments, we say that Q is a *standard query*. Below, we use *query* to mean both an abstract and a standard query.

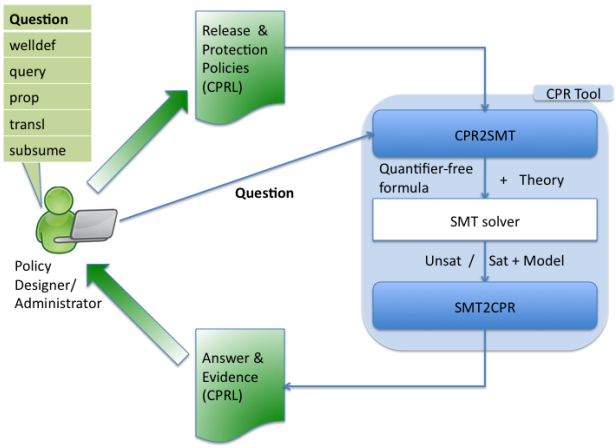

Figure 1: Enforcement and Analysis of CPR Policies

Let $\alpha(\mathsf{u}, \mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e})$ be a CPR policy and $Q(\mathsf{u}, \mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e})$ a query; a user u in U may execute an action a in A on a resource r in R using a terminal t in T in an environment e belonging to E iff

$$Q(\mathsf{u}, \mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e}) \wedge \alpha(\mathsf{u}, \mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e}) \tag{3}$$

is satisfiable modulo the theory $T_D = (\Sigma_D, \mathcal{M}_D)$ of domains, i.e. it evaluates to true in some of the interpretations fixing the meaning of the symbols in Σ_D according to the interpretations in $\mathcal{M}_D$. The satisfiability of (3) is equivalent to that of the following two formulae:

$$Q(\mathsf{u}, \mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e}) \wedge \rho(\mathsf{u}, \mathsf{r}, \mathsf{a}, \mathsf{e}) \tag{4}$$

$$Q(\mathsf{u}, \mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e}) \wedge \pi(\mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e}) \tag{5}$$

for $\alpha(\mathsf{u}, \mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e}) := \rho(\mathsf{u}, \mathsf{r}, \mathsf{a}, \mathsf{e}) \wedge \pi(\mathsf{t}, \mathsf{r}, \mathsf{a}, \mathsf{e})$. It is possible to show [5] that the above satisfiability checks are decidable when queries, protection and release policies are quantifier-free formulae, i.e. formulae obtained by combining atoms with the standard Boolean connectives.

3. SMT-BASED ANALYSIS AND ENFORCEMENT OF CPR POLICIES

Our approach to help policy designers and administrators in the definition, analysis and enforcement of CPR policies is depicted in Figure 1. Using *CPRL*, the designer declares the attributes of the entities, their domains, and the release and protection policies. Additionally, the designer selects a "Question" from those in the table in the top left corner of Figure 1 in order to form one of the following *CPRL (analysis) problems*.

- Single group (R, P) of release and protection policies: "*welldef*(R, P)" checks if there exists (at least) one query that is permitted and (at least) another one that is denied by R and P; "*query*(R, P, Q)" checks if the query Q is permitted or denied by R and P; "*prop*(R, P, O)" asks if property O is verified by R and P; and "*transl*(R, P)" asks to translate R and P to an XACML policy set.
- Two groups (R, P) and (R', P') of release and protection policies: "*query*(R, P, R', P', Q)" checks if query Q

is answered in the same way by group (R, P) and group (R', P'); "$prop(R, P, R', P', O)$" checks if property O is verified by group (R, P) and group (R', P'); and "$subsume(R, P, R', P')$" checks if group (R, P) subsumes group (R', P'), i.e. if using (R, P) automatically also fulfills (R', P').

These problems are solved by the CPR tool (on the right in Figure 1), which is composed of a pipeline of three modules: CPR2SMT, SMT Solver, and SMT2CPR. The first module takes as input the CPRL analysis problem and returns a quantifier-free FOL formula φ together with a theory T_D of the domains of the attributes such that the (un-)satisfiability of φ modulo T_D is equivalent to a positive (negative, respectively) answer to the given problem. The second module, SMT solver, takes as input a quantifier-free FOL formula φ with a theory T_D and returns as a result either unsat(isfiable), or sat(isfiable) respectively, together with a model making φ true. The third and last module, SMT2CPR, takes as input the output of the SMT solver and returns an answer to the original question. For instance, assume the CPRL analysis problem is "$query(R, P, Q)$" and "unsat" is returned by the SMT solver, then the answer computed by SMT2CPR is "Denied" meaning that the query Q is denied by R and P.

Below, we first (Section 3.1) present the syntax of CPRL for specifying a single group of release and protection policies, and define its semantics by a translation to first-order logic along the lines of Section 2.1. Then in Section 3.2, we describe the internal workings of the modules CPR2SMT and SMT2CPR in the CPR tool, i.e. how the CPRL analysis problems involving one group of release and protection policies listed above are reduced to SMT problems and how the answers to the logical problems returned by the SMT solver are translated back to CPRL. Concerning the SMT solver, we do not describe it because we use one of the off-the-shelf and publicly available solvers. In addition to the obvious capability of checking the satisfiability of quantifier-free formulae modulo the theory of attribute domains induced by CPRL declarations, we require the SMT solver to be able to return a model satisfying the formula (if applicable). As we will see, this capability—supported by many SMT solvers—is crucial for the construction of scenarios describing, for instance, why a property is not satisfied by a policy. Finally, in Section 3.3 we discuss the extension of CPRL to describe two groups of release and protection policies, policy semantics, and how the relevant CPRL analysis problems are reduced to SMT problems.

3.1 CPRL

Roughly, the core of CPRL contains constructs to define the theory T_D of domain attributes, the attributes *Att* defining the entities, and release and protection policies.

Syntax. For the definition of T_D, CPRL provides several constructs to introduce types that offer the syntactical characterization of the domains over which the attributes take values. There are four basic types: `nat` for naturals, `int` for integers, `bool` for Booleans, and `real` for reals. Additionally, it is possible to introduce the following two user-defined types. A *subrange* type S denoting a contiguous set of integers $\{a, ..., b\}$ with $a < b$ can be declared as follows:

type S = **subrange**$(a\,, b\,)$;

An *enumerated* type E denoting a finite set $\{e_1, ..., e_n\}$ of values (with $n \geq 1$) can be introduced by the construct:

type E = { e_1, ..., e_n };

For defining attributes and entities, CPRL provides the construct

entity E = [a_1 : t_1, ..., a_m : t_m];

where E can be `User`, `Resource`, `Action`, `Terminal`, or `Environment`, a_i is an attribute identifier, and t_i is a type identifier (for $i = 1, ..., m$ and $m \geq 1$).

Release and protection policies are introduced as follows:

release $R = F_R$; **protection** $P = F_P$;

where R and P are policy identifiers, whereas F_R and F_P are expressions built out of Boolean connectives (`!` for negation, `&` for conjunction, `|` for disjunction, `->` for implication, and `<->` for equivalence) and atoms of the forms: $e.a \bowtie v$ or $e.a \bowtie e'.a'$, where e, e' are entity identifiers, a, a' are attribute identifiers, v is a value, $\bowtie$ is either `=`, $>$, $\geq$, $<$, or $\leq$. We assume that atoms are well-typed, i.e. that a has the same type t as a' and that t is either `nat`, `int`, or `real` if $\bowtie$ is $>$, $\geq$, $<$, or $\leq$. Additionally, we assume that F_R may contain atoms with the entities `User`, `Action`, `Resource`, and `Environment` and F_P may contain atoms with the entities `Terminal`, `Action`, `Resource`, and `Environment`.

CPRL provides the following constructs to specify queries and properties:

query $Q = F_Q$; **property** $O = F_O$;

where Q and P are identifiers, whereas F_Q and F_O are expressions of the same forms as those in release and protection policy declarations with no restriction on the type of entities that atoms may contain.

Since CPRL adopts a very simple type system, we omit the description of the type-checking algorithm. Below, we assume that all CPRL expressions are well-typed.

Semantics. Along the lines of Section 2.1, we explain how the constructs of CPRL induce a theory T_D of domains and how it is possible to associate release and protection policy declarations together with queries and properties to quantifier-free formulae.

The theory of attribute domains. For the basic types, we assume the following theories: a theory $T_{\texttt{bool}}$ for `bool`eans with the type `bool` and two constants `true`, `false`, and whose models have a domain with only two elements; T_σ contains just the type $\sigma \in \{\texttt{nat}, \texttt{int}, \texttt{real}\}$, the numerals $0, 1, 2, ...$, the unary minus $-$, the binary addition $+$, and the orderings $>_\sigma$, $\geq_\sigma$, $<_\sigma$, and $\leq_\sigma$, and whose models are that of naturals, integers, or reals with the usual interpretation of the operations and relations according to σ.

For a subrange "`type` S = **subrange** (a, b)", we introduce a theory T_S with type S, the numerals $a, ..., b$, and whose models are those whose domains have cardinality $b - a + 1$. For an enumerated "`type` $E = \{e_1, ..., e_m\}$", we introduce a theory T_E with the type E, constants $e_1, ..., e_m$, and whose models are those whose domains have cardinality m.

Let $T_1, ..., T_k$ be the theories obtained by considering $T_{\texttt{bool}}$, $T_{\texttt{nat}}$, $T_{\texttt{int}}$, $T_{\texttt{real}}$, and all the theories associated to user-defined types (with $k \geq 4$). We define the theory $T_D = (\Sigma_D, \mathcal{M}_D)$ of the attribute domains to be the "combination" of all the theories above, i.e. Σ_D is the (disjoint) union of $T_1, ..., T_k$ and $\mathcal{M}_D$ is a collection of models such that

$$
\begin{aligned}
\langle v \rangle &:= v \text{ for } v \text{ a value} \\
\langle e.a \rangle &:= a(e) \text{ for } e \text{ an entity and } a \text{ an attribute} \\
\langle l = r \rangle &:= \langle l \rangle = \langle r \rangle \text{ for } l, r \text{ expressions of the same type} \\
\langle l \bowtie r \rangle &:= \langle l \rangle \bowtie_\sigma \langle r \rangle \text{ for } l, r \text{ of type } \sigma \in \{\texttt{nat}, \texttt{int}, \texttt{real}\} \\
&\qquad \text{and } \bowtie \in \{>, \geq, <, \leq\} \\
\langle !f \rangle &:= \neg \langle f \rangle \\
\langle f_1 \circ f_2 \rangle &:= \langle f_1 \rangle \bullet \langle f_2 \rangle \text{ where } (\circ, \bullet) \text{ is } (\texttt{\&}, \wedge),\ (\texttt{|}, \vee), \\
&\qquad (\texttt{->}, \Rightarrow), \text{ or } (\texttt{<->}, \Leftrightarrow) \\
&\qquad \text{for } f, f_1, f_2 \text{ CPRL expressions}
\end{aligned}
$$

Figure 2: Translation of CPRL expressions

$M \in \mathcal{M}_D$ iff the structure obtained from M by ignoring the interpretation of the symbols not in Σ_i is a model of T_i—i.e. is in the collection $\mathcal{M}_i$—where $T_i = (\Sigma_i, \mathcal{M}_i)$ and $i = 1, ..., k$.

Entities and attributes. We extend the signature of T_D with the types `User`, `Action`, `Resource`, `Terminal`, `Environment` for the entities together with the set Att of unary function symbols $a_i : E \rightarrow t_i$ extracted from all entity declarations "**entity** $E = [\, a_1 : t_1, ..., a_m : t_m \,]$" for $i = 1, ..., m$. Without loss of generality, we assume that sets of attributes associated to entities are pairwise disjoint.

Release and protection policies. The two declarations

$$\textbf{release}\ \ R = F_R\,; \qquad \textbf{protection}\ \ P = F_P\,;$$

for release and protection policies are translated to the following two first-order formulae:

$$\forall \mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e}.R(\mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \Leftrightarrow \langle F_R \rangle \tag{6}$$

$$\forall \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}.P(\mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \Leftrightarrow \langle F_P \rangle \tag{7}$$

where $\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}$ are variables of type `User`, `Terminal`, `Action`, `Resource`, and `Environment`, respectively, with the mapping $\langle \cdot \rangle$ recursively defined on the structure of expressions as shown in Figure 2. The formulae (6) and (7) can be seen as macro definitions: $R(\mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e})$ ($P(\mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})$) abbreviates the formula $\langle F_R \rangle$ ($\langle F_P \rangle$, respectively).

Queries and properties. The two declarations

$$\textbf{query}\ \ Q = F_Q\,; \qquad \textbf{property}\ \ O = F_O\,;$$

for queries and properties are translated to the following two first-order formulae:

$$\forall \mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}.Q(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \Leftrightarrow \langle F_Q \rangle \tag{8}$$

$$\forall \mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}.O(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \Leftrightarrow \langle F_O \rangle \tag{9}$$

where $\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}$ are variables of type `User`, `Terminal`, `Action`, `Resource`, and `Environment`, respectively. As for release and protection policies, the formulae (8) and (9) can be seen as macro definitions: $Q(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})$ ($O(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})$) abbreviates the formula $\langle F_Q \rangle$ ($\langle F_O \rangle$, respectively).

Putting things together. Assume a given CPRL specification containing declarations for types, attributes and entities, release and protection policies. Let T_D be the theory of attribute domains corresponding to the type declarations and Att be the set of unary functions corresponding to the attributes specified declared in entity declarations. Let $R_1, ..., R_r$ and $P_1, ..., P_p$ be the predicates defined by the formulae of the form (6) and (7), respectively, with $r \geq 1$ and $p \geq 1$. Let $Q_1, ...Q_q$ and $O_1, ...O_o$ be the predicates defined by the formulae of the form (8) and (9), respectively, with $q \geq 0$ and $o \geq 0$. The *global* release and protection policies are thus

$$R(\mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e}) := \bigvee_{i=1}^{r} R_i(\mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \quad P(\mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) := \bigvee_{i=1}^{p} P_i(\mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})\,,$$

respectively. Thus, according to (1), the *global* access control policy is

$$AC(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) := R(\mathsf{u}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \wedge P(\mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})\,. \tag{10}$$

A query Q_j is permitted (denied) iff the formula

$$Q_j(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \wedge AC(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \tag{11}$$

is satisfiable (unsatisfiable, respectively) modulo the theory T_D of attribute domains ($j = 1, ..., q$). A property P_j is verified (or not) iff the formula

$$AC(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \Rightarrow P_j(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e})$$

is valid (or invalid, respectively) modulo the theory T_D of attribute domains ($j = 1, ..., q$), i.e. every model T_D makes the formula above true. Since the validity of a formula is equivalent to the unsatisfiability of its negation [13], we can rephrase the condition for verifying a property as follows: P_j is verified iff the formula

$$AC(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \wedge \neg P_j(\mathsf{u}, \mathsf{t}, \mathsf{a}, \mathsf{r}, \mathsf{e}) \tag{12}$$

is satisfiable modulo T_D ($j = 1, ..., q$).

THEOREM 1. *Answering queries and proving properties of CPR policies specified in CPRL is decidable and NP-complete.*

PROOF. (*Sketch*) Decidability is a corollary of Theorem 1 in [5]. NP-hardness follows from the fact that propositional formulae are embedded in the formulae to which CPRL expressions are translated (cf. Figure 2). To show membership in NP, it is sufficient to observe that (a) in nondeterministic polynomial time in the size of the formula φ encoding the problem of answering queries or proving properties of CPR policies, we can guess a conjunction Γ of literals such that φ is satisfiable modulo T_D iff Γ is so and (b) we can check whether Γ is satisfiable modulo T_D in nondeterministic polynomial time in the size of Γ. □

Notice that answering arbitrary queries in CPRL is NP-complete since they may have arbitrary Boolean structure. For some classes of queries (e.g. similar to those considered in [3]), one may obtain better complexity results by devising *ad hoc* algorithms. We do not do this in this paper as we prefer to rely on ABE for the enforcement. In previous work [5] we also translated CPRL to XACML to reuse existing monitor-based mechanisms. Thanks to ABE however, enforcement is easier even when resources are stored in cloud platforms.

3.2 The modules SMT2CPR and CPR2SMT

Assume a given CPRL specification containing declarations for types, attributes and entities, and release and protection policies. Let T_D be the induced theory of attribute domains, Att be the set of unary functions associated to the attributes, and $R_1, ..., R_r$ and $P_1, ..., P_p$ be the predicates in (6) and (7) associated to release and protection policy declarations, respectively (with $r \geq 1$ and $p \geq 1$).

Pre-processing. The CPRL tool verifies the "compatibility" of each release policy R_i with each protection policy P_j by checking the satisfiability of their conjunction, i.e. of the formula:

$$AC_{i,j}(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \quad := \quad R_i(\mathsf{u},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge P_j(\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$$

for $i = 1,...,r$, $j = 1,...,p$, and $i \neq j$. The motivation for this is that the formula

$$\bigvee_{(i,j)\in Sat} AC_{i,j}(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \tag{13}$$

is a more compact but equivalent version of (10), where $Sat := \{(i,j) | AC_{i,j}(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \text{ is satisfiable modulo } T_D\}$.

PROPERTY 1. *The formulae (10) and (13) are logically equivalent modulo T_D.*

PROOF. Recall that (10) is the formula:

$$(\bigvee_{i=1}^{r} R_i(\mathsf{u},\mathsf{a},\mathsf{r},\mathsf{e})) \wedge (\bigvee_{j=1}^{p} P_j(\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}))\,,$$

which, by simple logical manipulations, is equivalent to

$$\bigvee_{i=1,...,r;\ j=1,...,p} (R_i(\mathsf{u},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge P_j(\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}))\,.$$

Observe that any disjunct $R_i(\mathsf{u},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge P_j(\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ that is unsatisfiable modulo T_D can be deleted without loss of generality when considering only the models of T_D, as is the case for computing answers to queries and verifying properties. The formula obtained in this way is (13). □

Because of this property, during the pre-processing phase, the module CPR2SMT in Figure 1 forms all possible formulae the form $R_i \wedge P_j$ (for $i = 1,...,r$, $j = 1,...,p$, $i \neq j$) and it will then let $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ denote the disjunction of all the formulae that are found to be satisfiable modulo T_D. The satisfiability checks are performed by an available SMT solver, which takes as input T_D and all the formulae $R_i \wedge P_j$. Notice that a satisfiable formula $R_i \wedge P_j$ denotes a pair of release and protection policy expressions that are "compatible", i.e. that have the potential to answer "permit" to some queries. Indeed, the module SMT2CPR in Figure 1 returns a list of all compatible pairs of release and protection policies and warns the user if a release (protection) policy is not compatible with any of the protection (release, respectively) policies. Finally, it raises an alarm if the list of compatible policies is empty: in this case, the policy denies all access requests!

CPRL analysis problem: "*welldef*(R,P)". A policy that permits or denies every query is not useful. We say that a policy is *well-defined* iff there exist two queries such that one is permitted and the other one is denied. To verify if there exists a query Q_g (Q_d) that is permitted (denied, respectively) by AC, it is sufficient to check if $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ (its negation $\neg AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$, respectively) is satisfiable modulo T_D.

PROPERTY 2. *If both $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ and its negation, $\neg AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$, are satisfiable modulo the theory T_D, then there must exist attribute assignments C_x^u, C_x^t, C_x^a, C_x^r, and C_x^e of the form (2) such that*

- $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge Q_g(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ *is satisfiable modulo T_D*
- $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge Q_d(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ *is not so,*

where $x = g,d$ and $\forall \mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}.Q_x(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \Leftrightarrow (C_x^u(\mathsf{u}) \wedge C_x^t(\mathsf{t}) \wedge C_x^a(\mathsf{a}) \wedge C_x^r(\mathsf{r}) \wedge C_x^e(\mathsf{e}))$.

PROOF. By assumption, $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ is satisfiable modulo T_D. This implies that there exists a model $\mathcal{M}_g$ of T_D in which AC evaluates to *true*. This means that there exist entities $\mathcal{M}_g[\mathsf{u}]$, $\mathcal{M}_g[\mathsf{t}]$, $\mathcal{M}_g[\mathsf{a}]$, $\mathcal{M}_g[\mathsf{r}]$, $\mathcal{M}_g[\mathsf{e}]$ assigned to the variables $\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}$, respectively, such that $\mathcal{M}_g$ interprets each function a_e representing an attribute a_e in Att of the entity e to given values, i.e. $\mathcal{M}_g[a](\mathcal{M}_g[\mathsf{u}])$, $\mathcal{M}_g[a](\mathcal{M}_g[\mathsf{t}])$, $\mathcal{M}_g[a](\mathcal{M}_g[\mathsf{a}])$, $\mathcal{M}_g[a](\mathcal{M}_g[\mathsf{r}])$, and $\mathcal{M}_g[a](\mathcal{M}_g[\mathsf{e}])$, respectively. Let $C_g^e(e)$ be the conjunction of the atoms of the form $a_e(e) = \mathcal{M}_g[a_e](\mathcal{M}_g[e])$ for every attribute a_e associated to entity $e \in \{\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}\}$. By construction, it is clear that $C_g^e(e)$ evaluates to *true* in $\mathcal{M}_g$. As a consequence, the formula $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge \bigwedge_{e\in\{\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}\}} C_g^e(e)$ also evaluates to *true* in $\mathcal{M}_g$. This implies that $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge Q_d(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ is satisfiable modulo the theory T_D (as $\mathcal{M}_g$ is a model of T_D).

The other situation, i.e. when $\neg AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ is unsatisfiable in T_D, is similar and is therefore omitted. □

By this property, the CPR2SMT module in Figure 1 passes both AC and its negation $\neg AC$ to the available SMT solver, which is assumed to return, in addition to sat/unsat, a model of the input formula in case of satisfiability. The SMT2CPR module in Figure 1 then warns the user that the policy is not well-defined if one of the two input formulae is found unsatisfiable (in this case, every query will be always permitted or denied) or it extracts a query (the attribute assignments C^u, C^t, C^a, C^r, and C^e, which are then translated back to the CPRL language). (This task is easy since the CPRL constructs devoted to express policies and queries can be seen as a concrete syntax for formulae of first-order logic.)

CPRL analysis problem: "*query*(R,P,Q)". Recall that a query "**query** Q = F_Q" is translated to a formula of the form (8). By definition, a query is permitted iff the formula $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge Q(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ is satisfiable modulo T_D. Because of the pre-processing phase, we know that AC is a disjunction of conjunctions of the form $R_i \wedge P_j$ (for $(i,j) \in Sat$) and thus, it is sufficient to find the pair $(i,j) \in Sat$ such that

$$R_i(\mathsf{u},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge P_j(\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge Q(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \tag{14}$$

is satisfiable module T_D so as to permit access to user u to perform action a on resource r using terminal t in environment e; otherwise, i.e. if for all pairs in Sat, the formula (14) is unsatisfiable modulo T_D, the query is denied. The module CPR2SMT in Figure 1 sends all formulae of the form (14) to the SMT solver, which establishes their satisfiability or unsatisfiability modulo T_D. In case all formulae are unsatisfiable, the module SMT2CPR in Figure 1 signals that the query Q is denied; otherwise, for each pair of release and protection policies that permits the query, attribute assignments for users, terminals, actions, resources, and environments satisfying the query are extracted from the model satisfying (14) returned by the SMT solver and shown to the user. Reporting all compatible pairs (not just one) of release and protection policies that positively answer a query allows for a deeper understanding of which parts of the access control policy permit a certain access. Showing the attribute assignments of the entities that satisfy the query again helps the user to build concrete authorization scenarios from abstract queries that consider sets of values for certain attributes or do not constrain others.

CPRL analysis problem: "*prop*(R, P, O)". Recall that a property "**property** P = F_P" is translated to a formula of the form (9). Also recall that a property is verified iff the formula $AC(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge \neg O(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e})$ is unsatisfiable modulo T_D. Because of the pre-processing phase, we know that AC is a disjunction of the form $R_i \wedge P_j$ (for $(i,j) \in Sat$) and thus we must check that for all pairs $(i,j) \in Sat$, the formula

$$R_i(\mathsf{u},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge P_j(\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \wedge \neg O(\mathsf{u},\mathsf{t},\mathsf{a},\mathsf{r},\mathsf{e}) \qquad (15)$$

is unsatisfiable modulo T_D; otherwise, i.e. if there exists a pair in Sat such that the formula (15) is satisfiable modulo T_D, the property is not verified. The module CPR2SMT sends all formulae of the form (15) to the SMT solver, which establishes their satisfiability or unsatisfiability modulo T_D^{Att}. When all formulae are unsatisfiable, the module SMT2CPR signals that the property O is verified; otherwise, an attribute assignment for users, terminals, actions, resources, and environments satisfying the policy but not O is extracted from the model satisfying (15) returned by the SMT solver and shown to the user. Such an attribute assignment is intended to provide a concrete authorization scenario in which the property is violated and to help the user understand why the property is not verified.

CPRL analysis problem: "*transl*(R, P, r)". In traditional public-key cryptography, a message is encrypted for a specific receiver using the receiver's public key. ABE re-defines the notion of identity as a set of attributes. In the ABE variant we use, Ciphertext-Policy ABE (CP-ABE) [15], a user's private key is associated with a set of attributes and a ciphertext specifies an access policy, which is a Boolean combination of constraints over the universe of attributes. Decryption succeeds iff the user's attibutes satisfy the ABE policy associated to the ciphertext.

In order to translate CPR into ABE policies, the idea is to compile a release and protection policy into two ABE policies for a particular resource r, use the two ABE policies to encrypt r, and then distribute appropriate keys to users and terminals whose attributes satisfy the ABE policies. For uniformity with CPR policies, we denote ABE policies with Java-like Boolean expressions and write $\alpha(u)$ and $\gamma(t)$ when these constrain the values of attributes on users u and terminals t, respectively. For simplicity we ignore action a (since here we account for only reading actions) and environment e. The problem of translating a CPR policy (R, P) to ABE policies can be abstractly characterized as follows: given resource r, derive two ABE policies α and γ characterizing the set of users and the set of terminals, respectively, that are entitled to access r according to R and P, respectively. This problem can be reduced to that of partially evaluating the expressions $R(u, r)$ and $P(t, r)$. We use the SMT solver to perform the symbolic manipulations so as to obtain expressions $\alpha(u)$ and $\gamma(t)$, which are precisely the ABE policies we are looking for. Note that $\alpha(u)$ and $R(u, r)$ evaluate to the same Boolean value for every user u and environment e. The same holds for $\gamma(t)$ and $P(t, r)$ for every terminal t and environment e. From these observations, it is straightforward to show the correctness of the translation under the assumption that a trusted third party generates and distributes keys to users and terminals according to their attributes: a monitor would permit user u to access resource r using terminal t in environment e iff u equipped with t can decrypt resource r.

Once α and γ have been derived, it is possible to use an available CP-ABE implementation [1, 2] to encrypt the resource r. In particular, r is first encrypted with the public key of the system and policy α; the result is encrypted again with the same public key together with policy γ. When a user requests access from a particular terminal, both keys are applied to the encrypted document in reverse order. The decryption will succeed only if both user and terminal private keys satisfy the CP-ABE policies.

THEOREM 2. *The CPRL analysis problems welldef*(R, P)*, query*(R, P, Q)*, prop*(R, P, O)*, and transl*(R, P, r) *are decidable and NP-complete.*

This follows from Theorem 1 and Properties 1 and 2.

3.3 Handling two groups of policies

The part of the CPRL language considered so far together with the CPRL analysis problems that the CPRL Tool is capable of solving are intended to help policy designers match their intentions with the specification of a single group of release and protection policies. Usually, policies change over time to take into account evolving authorization requirements or to correct/refine existing access control rules. CPRL also supports the specification of two groups (corresponding to two versions) of a policy by using the following constructs:

release R {**group** g} = F_R;
protection P {**group** g} = F_P;

where R, P, F_R, and F_P are as before and g is an identifier of a version of a policy. These constructs are translated to first-order formulae (6) and (7), respectively, by decorating the predicate symbols R and P with the identifier of the group g. For simplicity, we assume that only two groups of release and protection policies can be declared in a CPRL specification. Thus, two global release and protection policies together with two global access control policies AC and AC' are defined as in (10). It is then possible to solve the CPRL analysis problems "*query*(R, P, R', P', Q)" and "*prop*(R, P, R', P', O)" by using the same approach described above for a single group of release and protection policies. Notice that since groups of policies share the same type and entity declarations in a specification, the formulae for answering queries and verifying properties are checked for satisfiability modulo the same theory T_D of attribute domains. Additionally, the module SMT2CPR will highlight the situations in which the answers to the query Q are the same or different for the two groups of policies and when the property O is verified by one group of release and protection policies but not by the other.

The CPRL analysis problem "*subsume*(R, P, R', P')" aims to establish if the set of queries permitted by (R, P) is a subset of those permitted by (R', P'). Notice that every model of AC—representing the global policy of group (R, P)—can be uniquely associated to attribute assignments C^u, C^t, C^a, C^r, and C^e such that $AC \wedge C^u \wedge C^t \wedge C^a \wedge C^r \wedge C^e$ is satisfiable modulo the theory of the attributes (as explained in the proof of Property 2); similarly for the global policy AC' associated to group (R', P'). From this, it is clear that checking if the set of queries permitted by group (R, P) of policies is a sub-set of those permitted by the other group (R', P') can be reduced to checking if the class of models satisfying AC (AC') is contained in that satisfying AC' (AC, respectively). By definition, this amounts to checking if the formulae $AC \Rightarrow AC'$ and $AC' \Rightarrow AC$ are valid modulo the theory

T_D of attribute domains, or, equivalently, if $AC \wedge \neg AC'$ and $AC' \wedge \neg AC$ are unsatisfiable modulo T_D. The last two formulae are generated by the module CPR2SMT in Figure 1 and are passed to the SMT solver. The module SMT2CPR reports the results returned by the SMT solver for each formula and in case one is found satisfiable, it extracts a query from the returned model that is permitted by one group of policies and denied by the other. This yields a concrete authorization scenario from which the user can understand the similarities and the differences between the two versions of the policy.

THEOREM 3. *The CPRL analysis problems* $query(R, P, R', P', Q)$, $prop(R, P, R', P', O)$, *and* $subsume(R, P, R', P')$ *are decidable and NP-complete.*

This also follows from Theorem 1 and Properties 1 and 2.

4. IMPLEMENTATION, USE, AND EVALUATION OF THE CPR TOOL

We implemented the CPR Tool as described in Section 3. The prototype is written in Java, uses ANTLR[2] for parsing, and Yices[3] as the backend SMT solver. Among the many available SMT solvers, we chose Yices v. 1.0.40 for its user-friendly input language, which greatly facilitated the generation of the various proof obligations; however, many other SMT solvers can be used as a backend for the CPRL Tool since they support reasoning in the theory of attribute domains induced by the type declaration of CPRL.

We modelled a scenario of use of the CPR Tool with a realistic policy for information-sharing in a NATO rescue mission and then studied the scalability of the tool by means of synthetic policies. The policy for the NATO rescue mission is inspired by a realistic situation but has been greatly simplified; its description was moved to Appendix A to meet space limitations. Despite its simplification, we believe it gives an idea of the type of insights that policy designers can gain by interacting with the CPR Tool. In the following section, the synthetic benchmarks are used to evaluate the scalability of the tool; their use is common practice in the literature on policy analysis; see, e.g., [3].

4.1 Synthetic benchmarks

In order to evaluate the scalability of the CPR Tool, we designed and implemented a generator of synthetic policies, queries, and properties. The generator takes as input several parameters such as the number r of release policies, the number p of protection policies, and the average number a of atoms per release, protection, query, and property expression. We generated policies with an equal number of release r and protection p policies with $r = p = 25, 50, 75$ and an increasing number of atoms for $a = 5, 10, 15, 20$. Figure 3 shows the percentage of useful and compatible policies plotted with respect the average number a of atoms (we do not specify the number of release r and protection p policies since the percentages do not change significantly for varying values of $r = p = 25, 50, 75$): a release (protection, respectively) policy is *useful* when there exists at least one protection (release, respectively) policy forming a compatible pair, i.e. a pair in the set *Sat* computed by the pre-processing step

[2] `http://www.antlr.org`
[3] `http://yices.csl.sri.com`

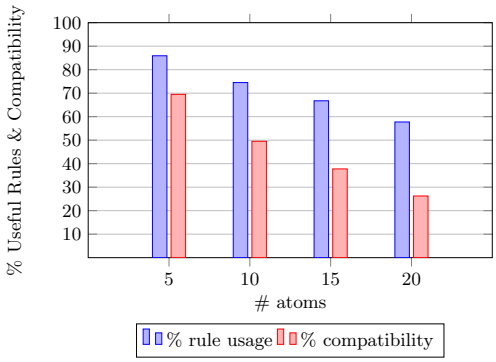

Figure 3: Characteristics of synthetic benchmarks

of Section 3.2. Notice that the cardinality of *Sat* can be at most r^2 (for $r = p$); thus, when $a = 5$, 70% compatible release and protection policies means that the cardinality of *Sat* is about 438 for $r = 25$, 1,750 for $r = 50$, and 3,938 for $r = 75$. As expected, the compatibility decreases from 70% for $a = 5$ to around 25% for $a = 20$ since each release and protection policy is more and more constrained and finding a compatible release or protection policy is less likely. In our experience (such as with the Passive Missile Defence scenario of Appendix A), the degree of compatibility between release and protection policies is usually lower than 20% with policy expressions containing on average between 5 and 7 atoms. As a consequence, the formulae generated by the CPR2SMT module in Figure 1 would be smaller on average than those considered in these benchmarks. We regard the policies considered here to be those of a worst-case scenario for the CPR Tool.

We ran the CPR Tool to solve the following CPRL analysis problems: $welldef(R, P)$, $query(R, P, Q)$, $prop(R, P, O)$, and $subsume(R, P, R', P')$. We did not consider $query(R, P, R', P', Q)$ and $prop(R, P, R', P', O)$ since they consist of two instances of problems involving a single group of policies, namely $query(R, P, Q)$ with $query(R', P', Q)$ and $prop(R, P, O)$ with $prop(R', P', O)$. All experiments were conducted on an Intel QuadCore (3.6 Ghz) CPU with 16 GB of RAM running Ubuntu 11.10. Figures 4 and 5 show the plots of the time taken by the CPR Tool to solve instances of *welldef* (upper half) and *query* (lower half) and *prop* (upper half) and *subsume* (lower half): the x-axis reports the increasing values of $r = p = 25, 50, 75$, the y-axis records the average timings (in seconds) over 10 policies randomly generated for the same value of $r = p$. The legend refers to the average number a of atoms per expression in the policies considered. For solving each of the 10 instances of every problem, the CPR tool was given a time-out of 15 minutes for *welldef*, of 15 minutes for *query* with 8 queries, of 30 minutes for *prop* with 10 properties, and of 20 minutes for *subsume*. We experienced no time-outs on *welldef*, no time-outs on *query* for any value of $r = p$, about 30% time-outs on *subsume*, and an average of 35% time-outs on *prop* for $r = p = 25, 50, 75$ with $a = 5, 10$ and more than 60% for $a = 15, 20$. Notice that, since the computation of the compatibility between release and protection policies is preliminary to the solution of any other problem (cf. the pre-processing in Section 3.2), the timings for *query*, *prop*, and *subsume* include that for *welldef*.

Despite difficult problem instances introduced by our synthetic policies, the behavior of the CPR Tool on the syn-

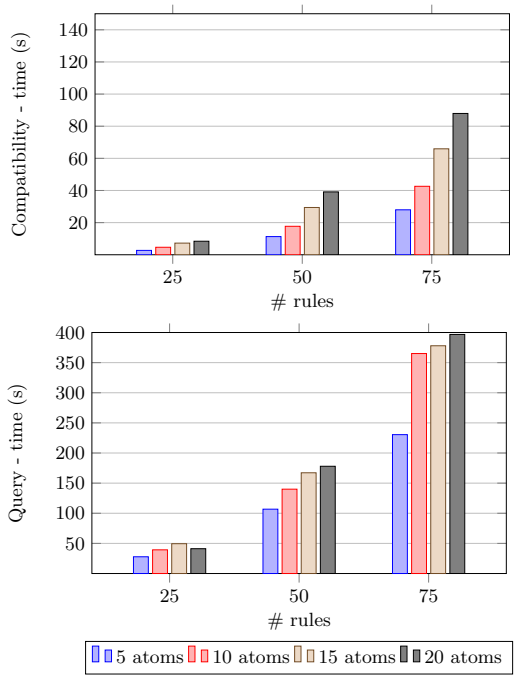

Figure 4: Results for *welldef* and *query* problems

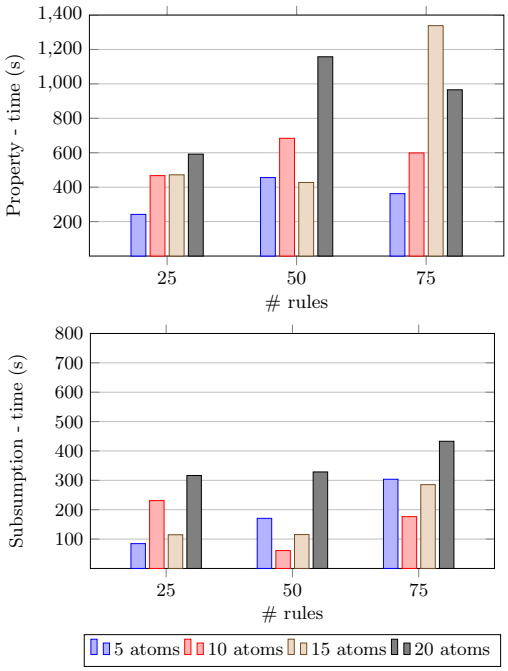

Figure 5: Results for *prop* and *subsume* problems

thetic benchmarks considered here is quadratic in the number $r = p$ of policy expressions for almost all problems considered above. An interesting area of future work would be to split the SMT problems generated by the CPR2SMT module into sub-problems and exploit the incremental satisfiability check capabilities available in most state-of-the-art SMT solvers (such as Yices). This would be particularly interesting for improving the performances on the *prop* problem instances.

5. DISCUSSION

Like many other recent approaches to specifying access control policies (see, e.g., [10] for an overview), CPRL is rooted in first-order logic augmented with theories. The main advantage of this approach is the possibility to strike the best trade-off between the expressivity of the language and the efficiency of answering queries and solving policy analysis problems. CPRL aims to provide enough flexibility to support different access control models and their combinations by using attributes (as suggested in [21]) while supporting the design of fully automated techniques for a variety of analysis problems. For instance, [5] develops the decidability and complexity result of which Theorems 1, 2, and 3 above are corollaries.

Most other logic-based access control models available in the literature are based on some variant of Datalog [20, 12, 23, 7, 16]. The idea underlying this type of work is to build mechanisms to answer authorization queries by reducing them to logical problems that can be efficiently (e.g. in linear time) solved. Datalog-based policy languages suffer some shortcomings related to expressivity (as discussed in [17]) and certain policy analysis problems are impossible to solve automatically; for instance, subsumption is shown to be undecidable for policies expressed in Datalog in [8]. In contrast, CPRL has been designed to recast a wide range of policy analysis problems as theorem-proving problems in first-order logic that can be automatically solved by available state-of-the-art SMT solvers. We have demonstrated the efficiency and scalability of the CPR Tool approach to several analysis problems with the extensive experimental evaluation of Section 4.1. Rather than using SMT solvers for answering authorization queries, we prefer to translate CPRL specifications into XACML policies and then exploit the abundance of available and well-engineered tools for answering queries. Since the translation is faithful, XACML policies are guaranteed to behave and to satisfy the same properties as the original ones expressed in CPRL. This is similar in spirit to [32] with two main differences. First, CPRL is more expressive than the policy language proposed in [32]. Second, the analysis problems are solved by SMT solving while those in [32] by model checking. Since CPRL does not allow expression of state-change actions, there is no need to use a model checker to solve the resulting analysis problems. The main goal of a related paper [6] is to automatically detect inconsistencies and redundancies in rules for role provision expressed as conditions on attributes by a reduction to SMT problems. The work more closely related to ours with respect to the use of SMT solvers to mechanize policy analysis for ABAC policies is [3]. The main difference is the fact that we re-use available policy enforcement mechanisms by translating CPRL to XACML instead of proposing an *ad hoc* policy enforcement algorithm as in [3].

Several tools [14, 18, 25, 19, 24] reduce policy analysis to Boolean satisfiability so that state-of-the-art SAT solvers can be used to discharge the resulting proof obligations. In particular, the work presented in [24, 26] combines model checking and SAT-solver based techniques on pre-processed XACML policies in order to provide the user with an environment for analysing and integrating sets of policies. One may wonder if SMT tools are really needed to solve the logi-

cal problems resulting from CPRL policy analysis problems since reductions from the theories (e.g. $T_{\texttt{int}}$) used in CPRL to the Boolean satisfiability problem are available. Differentiating between SMT and SAT solvers invoked after reductions is not meaningful from a conceptual point of view as pointed out in [28], which identifies *lazy* and *eager* strategies to SMT solving: the former is based on interleaving SAT and theory solving while the latter applies theory reasoning first to derive an equisatisfiable Boolean formula and then invokes the SAT solver. The upfront reduction to Boolean satisfiability of expressive theories (such as $T_{\texttt{int}}$) is computationally demanding since it requires encoding, as additional Boolean formulae, all implicit relations implied by the theory. With the lazy approach, theory reasoning is applied when needed to refine Boolean reasoning. Since interleaving the reasoning activities leads to substantial speed-ups in practice, most available SMT solvers implement the lazy strategy since this has emerged as the most efficient of the two. This is the reason we have chosen SMT instead of SAT solvers to design and implement our policy analysis tool.

Finally, we mention the use of reasoning tools for description logics in the policy analysis tool of [22]; the main drawback of this approach is the lack of scalability.

6. REFERENCES

[1] `http://acsc.cs.utexas.edu/cpabe/`.

[2] J. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2), 2013.

[3] K. Arkoudas, S. Loeb, R. Chadha, J. Chiang, and K. Whittaker. Automated Policy Analysis. In *IEEE Int. Symp. on POLICY*, pages 1–8, July 2012.

[4] A. Armando, M. Grasso, S. Oudkerk, S. Ranise, and K. Wrona. Content-based Information Protection and Release in NATO Operations. In *SACMAT*, 2013.

[5] A. Armando, S. Oudkerk, S. Ranise, and K. Wrona. Formal Modelling of Content-Based Protection and Release for Access Control in NATO Operations. In *FPS 2013*, volume 8352 of *LNCS*, pages 227–244, 2014.

[6] A. Armando and S. Ranise. Automated and Efficient Analysis of Role-Based Access Control with Attributes. In *DBSec*, number 7371 in LNCS, 2012.

[7] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and Semantics of a Decentralized Authorization Language. *JCS*, 18(4):597–643, 2010.

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

[9] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-Policy Attribute-Based Encryption. In *28th IEEE Symp. on Security and Privacy*, 2006.

[10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and P. Samarati. Access control policies and languages. *JCSE*, 3(2):94–102, 2007.

[11] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and P. Samarati. Access control policies and languages in open environments. In *Secure Data Management in Decentralized Systems*. Springer, 2007.

[12] J. DeTreville. Binder, a logic-based security language. In *IEEE Symposium on Security and Privacy*, 2002.

[13] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, Inc., 1972.

[14] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access control policies. In *ICSE*, pages 196–206, 2005.

[15] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *ACM Conf. on Computer and Communications Security (CCS), Alexandria, VA*, 2006.

[16] Y. Gurevich and I. Neeman. Dkal: Distributed-knowledge authorization language. In *Proceedings of CSF*, pages 149–162, 2008.

[17] J. Y. Halpern and V. Weissman. Using First-Order Logic to Reason about Policies. *ACM TISSEc*, 11(4):21:1–21:41, July 2008.

[18] H. Hu, G.-J. Ahn, and K. Kulkarni. Discovery and Resolution of Anomalies in Web Access Control Policies. *IEEE TDSeC*, 10(6):341–354, 2013.

[19] G. Hughes and T. Bultan. Automated verification of access control policies using a SAT solver. *STTT*, 10(6):503–520, 2008.

[20] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. *Security and Privacy, IEEE Symposium on*, 1997.

[21] X. Jin, R. Krishnan, and R. Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *DBSec*, number 7371 in LNCS, pages 41–55, 2012.

[22] V. Kolovski, J. Hendler, and B. Parsia. Analyzing web access control policies. In *WWW*, pages 677–686, 2007.

[23] N. Li and J. C. Mitchell. Datalog with constraints: a foundation for trust management langauges. In *PADL'03*, LNCS 2562, pages 58–73. Springer, 2003.

[24] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. EXAM: a comprehensive environment for the analysis of access control policies. *Int. J. Inf. Sec.*, 9(4):253–273, 2010.

[25] M. Mankai and L. Logrippo. Access Control Policies: Modeling and Validation. In *5th NOTERE Conf.*, pages 85–91, 2005.

[26] P. Rao, G. Ghinita, E. Bertino, and J. Lobo. Visualization for access control policy analysis results using multi-level grids. In *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*, pages 25–28. IEEE, 2009.

[27] R. Sandhu, E. Coyne, H. Feinstein, and C. Youmann. Role-Based Access Control Models. *IEEE Computer*, 2(29):38–47, 1996.

[28] R. Sebastiani. Lazy Satisfiability Modulo Theories. *JSAT*, 3:141–224, 2007.

[29] K. Wrona and G. Hallingstad. Controlled information sharing in NATO operations. In *IEEE Military Communications Conference (MILCOM)*, 2011.

[30] K. Wrona and G. Hallingstad. Development of High Assurance Guards for NATO. In *MCC*, 2012.

[31] E. Yuan and T. Jin. Attribute-Based Access Control (ABAC) for Web Services. In *IEEE Int, Conf. on Web Services*, pages 561–569, 2005.

[32] N. Zhang, D. P. Guelev, and M. Ryan. Synthesising

Verified Access Control Systems through Model Checking. *JCS*, 16(1):1–61, 2007.

APPENDIX

A. SHARING MAPS COMPUTED BY THE PASSIVE MISSILE DEFENCE SYSTEM

The goal of the NATO Passive Missile Defence (PMD) system is to minimize the adverse effects of defence against missile attacks; to this end, simulations are run in specific geographic areas, taking into account several parameters (e.g. the type of missile and weather conditions). The result of a simulation is a map of the predicted enemy missile trajectory and debris impact area, annotated with the consequences of the impact at several locations, hazard areas with risk analysis, the trajectories of intercepting missiles, sub-munition locations and descriptions, etc. Maps generated by the PMD system can be used in NATO missions for crisis-response planning, disaster preparation and rescue, and medical operations, including those that require the coordination of NATO coalition partners with civilian organizations such as the Red Cross. Indeed, the various parts of a computed map are subject to different release conditions. For example, a NATO user may see both missile trajectories and public information about the zones of operation, while a Red Cross member must not see the former (because he may be able to infer the location from which the intercepting missile was fired) yet should be allowed to access the public information. In addition, protection requirements should be enforced to guarantee that accessed information can be handled with an adequate level of technical and operational support; e.g. data should be downloaded using a TLS protocol and stored in encrypted form on the laptop of the user. For the sake of brevity, we omit environments.

Figure 6 shows release conditions and protection requirements expressed in CPRL. The first block of declarations (lines 1-11) specifies the attributes of users, resources, and terminals. In particular, the user attributes are `organization` and `clearance` (line 8): the former contains the name of the organization to which the user belongs, and for simplicity, only two values are considered (line 7); the latter is the user clearance level (line 1). The terminal attributes are `confidentiality` and `mgauthority` (lines 10-11): the former gives the level of strength of the protection mechanisms to store the data offered by the terminal (line 2) while the latter specifies the name of the organization managing the terminal (line 7). The resource attributes are `category` and `topic` (line 9): the value of the former is the content category associated to the graphical objects in the map—namely `ScenarioDescription`, `COIMetrics` (metrics related to the consequences of the impact between the threatening and intercepting missiles), or `PublicInformation` (line 3); the value of the latter is the content topic, i.e. `HighValueAssetsOrLists`, `ThreatOperatingAreas`, `ThreatAndInterceptorTrajectoryDetails`, `GeneralHazardAreaLocation`), or `SubmunitionAreaLocation` (lines 4–6).

The second block of declarations (lines 13–28) lists release policies and the third block (lines 30–49) the protection policies. To illustrate the meaning of the specification in Figure 6, let us consider some of the policy declarations. The release policy expression `rP1` (line 13) says that any user can view objects containing public information: the fact that the keyword **user** is not even mentioned in the expression implies that no constraint is imposed on the values of user attributes. Similarly, the protection policy expression `pP1` (line 30) says that objects containing public information can be accessed via any terminal. The more complex release policy expression `rP2` (lines 14–19) can be interpreted as follows: a user belonging to NATO with clearance level `Secret` can access an object (`resource`) with category `ScenarioDescription` and whose topic can be either `HighValueAssetsOrLists`, `ThreatOperatingAreas`, or `ThreatAndInterceptorTrajectoryDetails`.

We can analyze the policy specification in Figure 6 with the CPRL Tool as follows. First, we check if the policy is well-defined. The answer from the CPR Tool is "Yes" together with the following permitted query (below, `clr`, `cat`, `top`, `con`, and `mga` stand for `clearance`, `category`, `topic`, `confidentiality`, and `mgauthority`, respectively):

```
user.clr=Restricted & user.org=NATO_Org &
resource.cat=COIMetrics &
resource.topic=SubmunitionAreaLocation &
terminal.con=High & terminal.mga=NATO_Org
```

meaning that a NATO user with clearance `Restricted` can access a resource with category `COIMetric` and topic `SubmunitionAreaLocation` using a `High` confidentiality terminal managed by NATO, and the following denied query:

```
user.clr=Unclassified & user.org=NATO_Org &
resource.cat=ScenarioDescriptions &
resource.top=GeneralHazardAreaLocation &
terminal.con=High & terminal.mga=NATO_Org
```

meaning that a NATO user with clearance `Unclassifed` cannot access a resource with category `ScenarioDescriptions` and topic `GeneralHazardAreaLocation` using a terminal managed by NATO with `High` confidentiality. We then ask the tool to to answer the following authorization query:

```
query q1= user.organization = NATO_Org &
terminal.mgauthority = NATO_Org &
resource.topic = GeneralHazardAreaLocation
```

encoding the intuition that a resource whose topic is `GeneralHazardAreaLocation` can be released to NATO users when they use NATO terminals. The tool confirms that query `q1` is permitted by the policy in Figure 6 and adds that it is so by using either release policy expression `rP1` with the protection policy `pP1` or release policy expression `rP3` with the protection policy `pP3` (reporting that no other combination of release and protection policy expressions permit the query). Additionally, the CPR Tool "completes" the assignment of values to attributes for users, terminals, and resources that are not mentioned by query `q1`. So, for example, together with the fact that `rP1` with `pP1` permits the query, the following attribute assignments are returned:

```
user.clr=Unclassified & user.org=NATO_Org &
resource.cat=PublicInformation &
resource.top=GeneralHazardAreaLocation &
terminal.con=NoInfo & terminal.mga=NATO_Org.
```

A closer look at these assignments gives us a hint about a possible under-specification of the policy since a resource whose topic is `GeneralHazardAreaLocation` cannot have category `PublicInfo`. This suggests adding the constraint

```
! (resource.topic = GeneralHazardAreaLocation)
```

```
type Clearances = { None, Unclassified, Restricted, Confidential, Secret };
type Confidentialities = { NoInfo, Basic, Standard, Enhanced, High };
type Categories = { PublicInformation, ScenarioDescriptions, COIMetrics };
type Topics = { HighValuesAssetsOrLists, ThreatOperatingAreas,
                ThreatAndInterceptorTrajectoryDetails,
                GeneralHazardAreaLocation, SubmunitionAreaLocation };
type Organizations = { NATO_Org, Red_Cross };
entity User = [ clearance : Clearances, organization : Organizations ];
entity Resource = [ category : Categories, topic : Topics ];
entity Terminal = [ confidentiality : Confidentialities,
    mgauthority : Organizations ];

release rP1 = (resource.category = PublicInformation);
release rP2 = (user.clearance = Secret) &
    (user.organization = NATO_Org) &
    (resource.category = ScenarioDescriptions) &
    ((resource.topic = HighValuesAssetsOrLists) |
        (resource.topic = ThreatOperatingAreas) |
        (resource.topic = ThreatAndInterceptorTrajectoryDetails));
release rP3 = (user.organization = NATO_Org) &
    (resource.category = COIMetrics) &
    (resource.topic = GeneralHazardAreaLocation);
release rP4 = (user.organization = NATO_Org) &
    (resource.category = COIMetrics) &
    (resource.topic = SubmunitionAreaLocation) &
    ((user.clearance = Secret)       |
    (user.clearance = Confidential) |
    (user.clearance = Restricted)    );

protection pP1 = (resource.category = PublicInformation);
protection pP2 = (resource.category = ScenarioDescriptions) &
    (terminal.mgauthority = NATO_Org) &
    ((resource.topic = HighValuesAssetsOrLists) |
        (resource.topic = ThreatOperatingAreas) |
        (resource.topic = ThreatAndInterceptorTrajectoryDetails)) &
    ((terminal.confidentiality = High) |
        (terminal.confidentiality = Enhanced));
protection pP3 = (resource.category = COIMetrics) &
    (resource.topic = GeneralHazardAreaLocation) &
    ((terminal.confidentiality = High)        |
        (terminal.confidentiality = Enhanced) |
        (terminal.confidentiality = Standard) |
        (terminal.confidentiality = Basic)    |
        (terminal.confidentiality = NoInfo)) &
    (terminal.mgauthority = NATO_Org);
protection pP4 = (resource.category = COIMetrics) &
    (resource.topic = SubmunitionAreaLocation) &
    !(terminal.confidentiality = NoInfo) &
    (terminal.mgauthority = NATO_Org);

query q1 = user.organization = NATO_Org &
    terminal.mgauthority = NATO_Org &
    resource.topic = GeneralHazardAreaLocation;
```

Figure 6: PMD scenario: release and protection policy expressions

to both `rP1` and `pP1`. If after this change to the policy we again ask the CPR Tool query `q1` above, we discover that only `rP3` with `pP3` permits it.

This is only a partial account of the use of the CPR Tool in the PMD scenario. It can also be used, for example, to translate the policy specification in Figure 6 to XACML so that available policy enforcement mechanisms can be used. This has been exploited to map the full PMD policy specification to XACML for use in the CPR access control system described in [4]. Further details on the use of the CPR Tool in the PMD scenario are given in [5].