An Algebra for Composing Access Control Policies

PIERO BONATTI
Università di Milano
SABRINA DE CAPITANI DI VIMERCATI
Università di Brescia
and
PIERANGELA SAMARATI
Università di Milano

Despite considerable advancements in the area of access control and authorization languages, current approaches to enforcing access control are all based on monolithic and complete specifications. This results limiting when restrictions to be enforced come from different input requirements, possibly under the control of different authorities, and where the specifics of some requirements may not even be known a priori. Turning individual specifications into a coherent policy to be fed into the access control system requires a nontrivial combination and translation process.

This paper addresses the problem of combining authorization specifications that may be independently stated, possibly in different languages and according to different policies. We propose an algebra of security policies together with its formal semantics and illustrate how to formulate complex policies in the algebra and reason about them. A translation of policy expressions into equivalent logic programs is illustrated, which provides the basis for the implementation of the algebra. The algebra's expressiveness is analyzed through a comparison with first-order logic.

Categories and Subject Descriptors: H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security, Design

Additional Key Words and Phrases: Access control, policy composition, algebra, logic programs

---

This paper extends the previous work by the authors appeared in the Proc. of the 7th ACM Conference on Computer and Communication Security (CCS 2000).
Authors' addresses: Piero Bonatti and Pierangela Samarati, Dipartimento di Tecnologie dell'Informazione, Università di Milano, Via Bramante, 65, 26013 Crema - Italy, email: bonatti@crema.unimi.it, samarati@dsi.unimi.it; Sabrina De Capitani di Vimercati, Dipartimento di Elettronica per l'Automazione, Università di Brescia, Via Branze, 38, 25123 Brescia - Italy, e-mail: decapita@ing.unibs.it

1. INTRODUCTION

Recent years have witnessed considerable work on access control models and languages. Many approaches have been proposed to increase expressiveness and flexibility of authorization languages by supporting multiple policies within a single framework [Bertino et al. 1999; Hosmer 1992; Jajodia et al. 2001; Li et al. 1999; Woo and Lam 1993]. All these proposals, while based on powerful languages able to express different policies, assume a single monolithic specification of the entire policy. Such an assumption does not fit many real world situations, where access control might need to combine restrictions independently stated that should be enforced as one. As an example, consider a data warehouse collecting data from different sources, where each data source may impose access restrictions on its data; access restrictions can be stated in different languages and with reference to different paradigms. Consider now a large organization composed of different departments and divisions, each of which can independently specify security policies; the global policy of the organization results from the combination of all these components. Another example is represented by "dynamic coalition" scenarios where different parties, coming together for a common goal for a limited time, need to merge their security requirements in a controlled way while retain their autonomy. A further example is provided by recent laws concerning privacy issues. In a modern information system, the security policy of the organization should combine internally specified constraints with externally imposed privacy regulations [Banisar and Davies 1999]. Finally, as security policies become more sophisticated, even within a single system ruled by one administrator it may be desirable to formulate the policy incrementally by assembling small, manageable, and independently conceived modules.

Existing frameworks represent these situations by translating and merging the different component policies into a single "program" in the adopted language. While existing languages are flexible enough to obtain the desired combined behavior, this method has several drawbacks. First, the translation process is far from being trivial; it must be done very carefully to avoid undesirable side effects due to interference between the component policies. Interference may result in the combined specifications not reflecting correctly the intended restrictions. Second, after translation it is not possible anymore to operate on the individual components and maintain them autonomously. Third, existing approaches cannot take into account incomplete policies, where some components are not (completely) known a priori (e.g., when somebody else is to provide that component).

This situation calls for a policy composition framework by which different component policies can be integrated while retaining their independence. In this paper, we propose an algebra for combining security policies together with its formal semantics. Complex policies are formulated as expressions of the algebra. Our framework results flexible and keeps compound specifications simple by organizing them into different levels of abstraction. The formal framework can be used to reason about properties of (possibly incomplete) specifications. We illustrate a translation of algebra expressions into equivalent logic programs, which provide the basis for the implementation of the language. The expressive power of the algebra will be analyzed from different points of view: by applying the algebra to some composition scenarios and to the specification of individual components, by evaluating the algebra against a list of desiderata for composition languages.

To our knowledge ours is the first proposal addressing composition of authorization specifications. Previous work on composition (e.g., [Abadi and Lamport 1992; Jaeger 1999]) focussed on the secure behavior of program modules. Closest work lies in proposals targeted to the development of a uniform framework to express possibly heterogeneous policies [Bertino et al. 1999; Jajodia et al. 2001; Li et al. 1999; Woo and Lam 1993]. However, as already discussed none of these proposals addressed composition. Our work is complementary to these proposals, which can be used to specify the individual component policies in our framework. A preliminary version of our work appeared in [Bonatti et al. 2000]. In this paper we extend the work with the expressiveness analysis of the algebra and with formal proofs of the correctness of the translation of the algebra expressions into logic programs.

2. CHARACTERISTICS OF A COMPOSITION FRAMEWORK

A first step in the definition of a framework for composing policies is the identification of the characteristics that it should have. In particular, we have identified the following:

1. *Heterogeneous policy support* The composition framework should be able to combine policies expressed in arbitrary languages and enforced by different mechanisms. For instance, a data warehouse may collect data from different data sources where the security restrictions autonomously stated by the sources and associated with the data may be stated with different specification languages, or refer to different paradigms (e.g., open vs closed policy).
2. *Support of unknown policies* It should be possible to account for policies which may be partially unknown, or be specified and enforced in external systems. These policies are like "black-boxes" for which no (complete) specification is provided, but that can be queried at access control time. Think, for example, of a situation where given accesses are subject to a policy P enforcing "central administration approval". While P can respond yes or no to each specific request, neither the description of P, nor the complete set of accesses that it allows might be available. Run-time evaluation is therefore the only possible option for P. In the context of a more complex and complete policy including P as a component, the specification could be partially compiled, leaving only P (and its possible consequences) to be evaluated at run time.
3. *Controlled interference* Policies cannot always be combined by simply merging their specifications (even if they are formulated in the same language), as this could have undesired side effects causing the accesses granted or denied to not correctly reflect the specifications. As a simple example, consider the combination of two systems P_{closed}, which applies a closed policy, based on rules of the form "*grant access if* $(s, o, +a)$", and P_{open} which applies an open policy, based on rules of the form "*grant access if* $\neg(s, o, -a)$". Merging the two specifications would cause the latter decision rule to derive all authorizations not blocked by P_{open}, regardless of the contents of P_{closed}. Similar problems may arise from uncontrolled interaction of the derivation rules of the two specifications. Besides, if the adopted language is a logic language with negation, the merged program

might not be stratified (which may lead to ambiguous or undefined semantics).

4. *Expressiveness* The language should be able to conveniently express a wide range of combinations (spanning from minimum privileges to maximum privileges, encompassing priority levels, overriding, confinement, refinement, etc.) in a uniform language. The different kinds of combinations must be expressed without changing the input specifications and without ad-hoc extensions to authorizations (like those introduced to support priorities). For instance, consider a department policy P_1 regulating access to documents and the central administration policy P_2. Assume that access to administrative documents can be granted only if authorized by both P_1 *and* P_2. This requisite can be expressed in existing approaches only by explicitly extending all the rules possibly referred to administrative documents to include the additional conditions specified by P_2. Among the drawbacks of this approach is the rule explosion that it would cause and the complex structure and loss of control over the two specifications which, in particular, cannot be maintained and managed autonomously anymore.
5. *Support of different abstraction levels* The composition language should highlight the different components and their interplay at different levels of abstraction. This feature is important to: *i)* facilitate specification analysis and design; *ii)* facilitate cooperative administration and agreement on global policies; *iii)* support incremental specification by refinement.
6. *Formal semantics* The composition language should be declarative, implementation independent, and based on a solid formal framework. An underlying formal framework is needed to: *i)* ensure non-ambiguous behavior and *ii)* reason about policy specifications and prove properties on them [Landwehr 1981].

3. AN ALGEBRA OF POLICIES

To make our approach generally applicable we do not make any assumption on the subjects, objects, or actions w.r.t. which accesses have to be controlled and, therefore, authorization specifications stated. We assume that sets S, O, and A denoting the subjects, objects, and actions, respectively, are given. Depending on the application context and the policy to be enforced, subjects could be users or groups thereof, as well as roles or applications; objects could be files, relations, XML documents, classes, and so on.

3.1 Preliminary concepts

We start by defining authorization terms as follows.

Definition Authorization term. *Authorization terms are triples of the form (s,o,a), where s is a constant in S or a variable over S, o is a constant in O or a variable over O, and a is a constant in A or a variable over A.*

At a semantic level, a policy is defined as a set of triples as follows.

Definition Policy. *A policy is a set of ground authorization terms.*

The triples in a policy P state the accesses permitted by P. Intuitively, a policy represents the outcome of an authorization specification, where, for composition purposes, it is irrelevant how specifications have been stated and their outcome computed.

The algebra (among other operations) allows policies to be restricted (by posing constraints on their authorizations) and closed w.r.t. inference rules. The model should be compatible with a variety of languages for constraining authorizations and formulating rules (e.g., [Jajodia et al. 2001; Li et al. 1999; Woo and Lam 1993]). For this purpose, we make our algebra parametric w.r.t. the following languages and their semantics:

(1) An *authorization constraint language* $\mathcal{L}_{acon}$ and a semantic relation $\mathsf{satisfy} \subseteq (\mathsf{S} \times \mathsf{O} \times \mathsf{A}) \times \mathcal{L}_{acon}$; the latter specifies for each ground authorization term (s, o, a) and constraint $c \in \mathcal{L}_{acon}$ whether (s, o, a) satisfies c.

(2) A *rule language* $\mathcal{L}_{rule}$ and a semantic function $\mathsf{closure} : \wp(\mathcal{L}_{rule}) \times \wp(\mathsf{S} \times \mathsf{O} \times \mathsf{A}) \to \wp(\mathsf{S} \times \mathsf{O} \times \mathsf{A})$; [1] the latter specifies for each set of rules R and ground authorizations P which authorizations are derived from P by R.

For simplicity, we consider a single authorization constraint language $\mathcal{L}_{acon}$ and a single rule language $\mathcal{L}_{rule}$. Our model can be straightforwardly extended to handle many such languages simultaneously, so that compound policies can be assembled using different tools.

To fix ideas and make concrete examples, in this paper we shall adopt the following simple languages for constraints and rules:

(1) $\mathcal{L}_{acon}$ contains constraints that can be modeled as *basic predicates* with at most three arguments taken from distinct basic domains (i.e., S, O, and A).[2] These predicates can evaluate properties associated with subjects, objects, and actions (e.g., membership of an object in a given constant set), or relationships between elements of different domains (e.g., ownership relationship between objects and subjects). A special case of unary predicates is represented by constraints of the form $(s\ \mathsf{op}\ s_0)$, $(o\ \mathsf{op}\ o_0)$, or $(a\ \mathsf{op}\ a_0)$, where *i)* s, o, a are variables ranging over S, O, and A, respectively; *ii)* op can be $\leq, \geq, <, >, =$, where disequalities model hierarchical relationships among subjects, objects, and actions (e.g., file/directory; user/group; role/superrole) [Jajodia et al. 2001], and *iii)* s_0, o_0, a_0 are members of S, O, and A, respectively. Note that these predicates are unary since one element in each simple constraint is always a constant.

(2) As a rule language we adopt simple Horn clauses, built from authorization terms and base predicates, of the form $(s, o, a) \leftarrow L_1 \wedge \ldots \wedge L_n$, where each L_i is either an authorization term or a basic predicate p. Here we regard authorization terms as logical atoms. For example, if a policy contains an authorization (s_1, o_1, a_1) and we close the policy under the rule $(s_2, o_2, a_2) \leftarrow (s_1, o_1, a_1)$, then the resulting policy will contain the derived authorization (s_2, o_2, a_2). The semantic function $\mathsf{closure}(R, P)$ is defined accordingly as the least Herbrand model of the logic program $\Pi = R \cup P \cup B$, where B is the definition of the base predicates. We recall that the least Herbrand model of Π can be expressed as $T_\Pi \uparrow \omega$, where T_Π is the *immediate consequence operator* associated with Π, and $T_\Pi \uparrow \omega$ is the limit of the monotonic infinite sequence

[1] For all sets X, $\wp(X)$ denotes the powerset of X.

[2] The reason for requiring the arguments in the predicates to belong to distinct domains is to limit $\mathcal{L}_{acon}$ to constraints that can be evaluated on each single authorization triple at a time (to determine whether an authorization satisfies or not given constraints)

$\emptyset, T_\Pi(\emptyset), T_\Pi(T_\Pi(\emptyset)), \ldots, T^i_\Pi(\emptyset), \ldots$, with i a natural number (see [Lloyd 1984] for more details).

These languages have been chosen with the goal of keeping presentation as simple as possible, focusing attention on policy composition, rather than authorization properties and inference rules.

3.2 Policy expressions

We are now ready to define our algebra. First, its syntax is introduced, then the meaning of each operator will be illustrated. We assume an infinite set of *policy identifiers* is given. Policy expression syntax is given by the following BNF grammar:

$$E ::= \mathbf{id} \mid E + E \mid E \& E \mid E - E \mid E\hat{\ }C \mid o(E,E,E) \mid E * R \mid T(E) \mid (E)$$
$$T ::= \tau\, \mathbf{id}.E$$

Here **id** is the token type of policy identifiers, E is the nonterminal describing *policy expressions*, T is a construct called *template*, that represents partially specified policies, C and R are the constructs describing $\mathcal{L}_{acon}$ and $\mathcal{L}_{rule}$, respectively (they are not specified here because the algebra is parametric w.r.t. $\mathcal{L}_{acon}$ and $\mathcal{L}_{rule}$). Note that templates are *not* policy expressions, only templates with actual parameters are. The above syntax is disambiguated by assigning suitable precedence and associativity to each operator:

op	precedence	associativity
τ	0	non-associative
.	1	non-associative
$+, \&, -$	2	left-associative
$*, \hat{\ }$	3	left-associative

We now discuss the semantics of the algebra. Formally, the semantics is a function that maps each policy expression onto a set of ground authorizations (that is, a policy), and each template onto a function over policies. The simplest possible expressions, namely identifiers, are bound to sets of triples by *environments*.[3]

Definition Environments. *An environment e is a partial mapping from policy identifiers to sets of ground authorizations. By $e[S/X]$ we denote a modification of environment e such that*

$$e[S/X](Y) = \begin{cases} S & \textit{if } Y = X \\ e(Y) & \textit{otherwise} \end{cases}$$

The binding can either be stated explicitly or generated by some engine. The policy can be seen in such a case as a "black box". In symbols, the semantics of an identifier X w.r.t. an environment e will be denoted by

$$[\![X]\!]_e \stackrel{\text{def}}{=} e(X)\,.$$

[3]Note that the following definition makes policy identifiers behave like the identifiers (or *names*) of functional languages.

Sometimes it will be convenient to use a distinguished policy identifier P_{all} to denote the set of all authorization triples. Accordingly, in the following, we shall restrict our attention to environments such that $e(P_{all}) = \mathsf{S} \times \mathsf{O} \times \mathsf{A}$.

Note that X might be undefined (not specified) in the environment; in that case $[\![X]\!]_e$ is undefined. Similarly, the semantics of compound expressions that use undefined identifiers is undefined.[4]

Compound policies can be obtained by combining policy identifiers through the algebra operators. Let the meta-variables P and P_i range over policy expressions.

Addition (+). It merges two policies by returning their union. Formally,

$$[\![P_1 + P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \cup [\![P_2]\!]_e.$$

For instance, in an organization composed of different divisions, access to the main gate can be authorized by any of the administrator of the divisions (each of them knows which users need access to reach their division). The totality of the accesses through the main gate to be authorized should then be the union of the statements of each division. Intuitively, additions can be applied in any situation where accesses can be authorized if allowed by any of the component policies.

Conjunction (&). It merges two policies by returning their intersection. Formally,

$$[\![P_1 \& P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \cap [\![P_2]\!]_e.$$

While addition allows an access if any of the component policies allows it, conjunction requires all the component policies to agree on the fact that the access should be granted. Intuitively, while addition enforces maximum privilege, conjunction enforces minimum privilege. For instance, consider an organization in which divisions share certain documents (e.g., clinical folders of patients). An access to a document may be allowed only if all the authorities that have a say on the document agree on it. That is, if the corresponding authorization triple belongs to the intersection of their policies.

Subtraction (-). It restricts a policy by eliminating all the accesses in a second policy. The formal definition is

$$[\![P_1 - P_2]\!]_e \stackrel{\text{def}}{=} [\![P_1]\!]_e \setminus [\![P_2]\!]_e.$$

Intuitively, subtraction specifies exceptions to statements made by a policy and encompasses the functionality of negative authorizations in existing approaches. The advantages of subtraction over explicit denials include a simplification of the conflict resolution policies and a clearer semantics. In particular, the difference operation allows to clearly and unambiguously express the two different uses of negative authorizations, namely *exceptions to positive statements* (authorizations to be negated are removed only from the specific policy for which they represent exceptions) and *explicit prohibitions* (authorizations to be negated are removed as the

[4]Undefined identifiers will be needed for the correct translation of unknown policies into logic programs (cf. Figure 3). For semantic analysis and expressiveness issues, however, we shall focus only on environments defined over all free identifiers (see definition of template operator), as all policies should be defined at access control time.

last operation to produce the global policy), which are often confused or require explicit ad-hoc extensions to authorizations [Rabitti et al. 1991]. Subtraction can also be used to express different overriding/conflict resolution criteria as needed in each specific context, without affecting the form of the authorizations (c.f. Section 7).

Closure ($*$). It closes a policy under a set of inference (derivation) rules. The general definition is

$$[\![P * R]\!]_e \stackrel{\text{def}}{=} \mathsf{closure}(R, [\![P]\!]_e).$$

Derivation rules can, for example, enforce propagation of authorizations along hierarchies in the data system, or enforce more general forms of implication, related to the presence or absence of other authorizations, or depending on properties of the authorizations [Jajodia et al. 2001]. Intuitively, derivation rules can be thought of as logic rules whose head is the authorization to be derived and whose body is the condition under which the authorization can be derived. The closure of a policy P under a set of rules R produces a policy containing all the authorizations that can be derived by evaluating R against P according to a given semantics. Recall that, in the examples of this paper, we assume rules to be Horn clauses. The general definition is thus specialized to $[\![P * R]\!]_e = T_{R \cup [\![P]\!]_e \cup B} \uparrow \omega$.

Scoping restriction ($\hat{}$). It restricts the application of a policy to a given set of subjects, objects, and actions. Formally,

$$[\![P\hat{\ }c]\!]_e \stackrel{\text{def}}{=} \{(s,o,a)\theta \mid (s,o,a)\theta \in [\![P]\!]_e, (s,o,a)\theta \text{ satisfy } c\theta\},$$

where $c \in \mathcal{L}_{acon}$ and θ is a ground substitution for variables s, o, a. Scoping is particularly useful to "limit" the statements that can be established by a policy and to enforce authority confinement. Intuitively, all authorizations in the policy which do not satisfy the scoping restriction are ignored, and therefore result ineffective. For instance, an organization can attach to a policy P_{adm}, to be specified by the administration, a scoping restriction that limits the authorizations that P_{adm} can state to administrative documents, expressed as "$o \leq$ `adm_documents`" or, equivalently, "`adm_document`(o)". Similarly, the organization can attach to policy P_{lib}, to be specified by the librarian, a scoping restriction "$a =$ `read`" that limits statements to read-only actions. Scoping restrictions can be also used to select a portion of a policy, which may be subject to a different treatment than the rest of P, for example, being overridden as discussed below.

Overriding (o). It replaces part of a policy with a corresponding fragment of a second policy. The portion to be replaced is specified by means of a third policy. Formally,

$$[\![o(P_1, P_2, P_3)]\!]_e \stackrel{\text{def}}{=} [\![(P_1 - P_3) + (P_2 \& P_3)]\!]_e.$$

For instance, consider the case where users of a library who have passed the due date for returning a book cannot borrow the same book anymore *unless* the responsible librarian vouchers for (authorizes) the loan. The policy can be expressed as $o(P_{lib}, P_{vouch}, P_{block})$, where P_{lib} are the accesses authorized at the library, P_{block} is the "black-list" of accesses, and P_{vouch} are the accesses authorized by the responsible librarian. Beside being specified explicitly, the fragment of P_1 to be overridden can be specified by means of scoping restrictions selecting triples in P_1 that satisfy a given condition. For instance, consider a department where access to laboratories is regulated by policy P_{lab}. Suppose that to be admitted, non-US citizens need *also* the chair consent, stated by policy P_{chair}. In other words, the portion of P_{lab} referred to non-US citizens should be overridden by its intersection with P_{chair}. This policy is then specified as $o(P_{lab}, P_{chair}, P_{lab}\hat{}\,\mathsf{non{-}UScitizen}(s))$. Note the importance of substituting to the fragment the intersection of the two policies, meaning *both* of them must agree on the access. Cases in which the fragment should simply be substituted with the second policy can be achieved via difference (or scoping restriction) and addition. In the following, we will abbreviate expression $o(P_1, P_2, P_1\hat{}c)$ as $o(P_1, P_2, \hat{}c)$.

Template (τ). It defines a partially specified policy that can be completed by supplying the parameters. $[\![\tau X.P]\!]_e$ is a function over policies (ground authorization sets), such that for all policies S,

$$[\![\tau X.P]\!]_e(S) \stackrel{\text{def}}{=} [\![P]\!]_{e[S/X]}\,.$$

Templates can be instantiated by applying them to a policy expression. For all policy expressions P_1, $[\![(\tau X.P)(P_1)]\!]_e \stackrel{\text{def}}{=} [\![\tau X.P]\!]_e([\![P_1]\!]_e) = [\![P]\!]_{e[[\![P_1]\!]_e/X]}$. We say that all the occurrences of X in an expression $\tau X.P$ are *bound*. The *free* identifiers of a policy expression P are all the identifiers with non-bound occurrences in P. Clearly, $[\![P]\!]_e$ is defined iff all the free identifiers in P are defined in e.
Templates are useful for representing partially specified policies, where some component X is to be specified at a later stage. For instance, X might be the result of further policy refinement, or it might be specified by a different authority. When a specification P_1 for X is available, the corresponding global policy can be simply expressed as $(\tau X.P)(P_1)$. As an example, consider the case where physicians of a hospital can access clinical data of patients only if the central administration of the hospital authorizes the access and some privacy regulations imposed by an external `Data Protection Authority` are satisfied. The policy can be expressed as $\tau X.(P_{hadm}\&X)$, where P_{hadm} are the accesses authorized by the administration of the hospital and X denotes the (unknown) policy specified by the `Data Protection Authority`. The subsequent consideration of a specific privacy regulation policy P_{priv} will then produce the global policy $\tau X.(P_{hadm}\&X)(P_{priv})$.

Templates with multiple parameters can be expressed and applied using the following abbreviation:

$$(\tau X_1, \ldots, X_n.P)(P_1, \ldots, P_n) = \tau X_1, (\tau X_2.(\ldots(\tau X_n.P)(P_n)\ldots)(P_2))(P_1)\,.$$

Figure 1 summarizes the operators of our algebra and their semantics and illustrates two possible graphical representations of algebraic operations. Basically, each policy is represented as a box containing the policy expression or the policy identifier. In the first representation, operators are represented as circles labeled with the operator. In the second representation, operators are represented as labels associated with arcs. Moreover, closure of a policy P under rules R is represented as a flag labeled R attached to P's box; the application of a scoping restriction c to a policy P is represented as a small box attached to P's box; and the overriding $o(P_1, P_2, P_3)$ is represented as an arc from P_1 to P_2, where the arc has attached P_3 (for simplicity, when P_3 is $P_1\hat{}c$ only an oval labeled c is attached to the arc). The

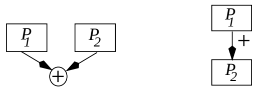
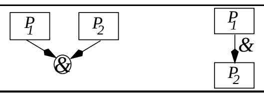
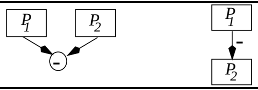
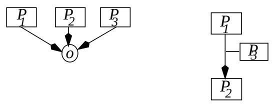
Operator	Semantics $[\![\ ]\!]_e$	Graphical representation
$P_1 + P_2$	$[\![P_1]\!]_e \cup [\![P_2]\!]_e$	
$P_1 \& P_2$	$[\![P_1]\!]_e \cap [\![P_2]\!]_e$	
$P_1 - P_2$	$[\![P_1]\!]_e \setminus [\![P_2]\!]_e$	
$P * R$	$T_{R \cup [\![P]\!]_e \cup B} \uparrow \omega$	
$P\hat{\ }c$	$\{t \in [\![P]\!]_e \mid t \text{ satisfy c}\}$	
$o(P_1, P_2, P_3)$	$[\![(P_1 - P_3) + (P_2 \& P_3)]\!]_e$	

Fig. 1. Operators of the algebra and their graphical representation

two representations can be used interchangeably as they better suit for the clarity of the resulting picture.

4. EXAMPLE SCENARIOS

We illustrate some examples of expressions stating protection requirements by composing policy statements through different operators.

4.1 Example 1: Hospital

Consider a hospital composed of three departments, namely `Radiology`, `Surgery`, and `Medicine`. Each of the departments is responsible for granting access to data under their (possibly overlapping) authority domains, where domains are specified by a scoping restriction. The statements made by the departments are then unioned, meaning the hospital considers an access as authorized if any of the department policies states so. For privacy regulations, however, the hospital will not allow any access (even if authorized by the departments) to `lab_tests` data unless there is patient consent for that, stated by policy $P_{consents}$. In terms of the algebra, the hospital policy can be represented as $o(P_{rad}\hat{\ }[o \leq \texttt{rad}] + P_{surg}\hat{\ }[o \leq \texttt{surg}] + P_{med}\hat{\ }[o \leq \texttt{med}], P_{consents}, \hat{\ }[o \leq \texttt{lab\_tests}])$. Accordingly, `lab_tests` data will be released only if *both* the hospital authorizes the release and the interested patient consents to it. As an example of component policies, let us zoom into $P_{consents}$ and P_{med}.

$P_{consents}$ reports accesses to laboratory tests for which there is patient consents.

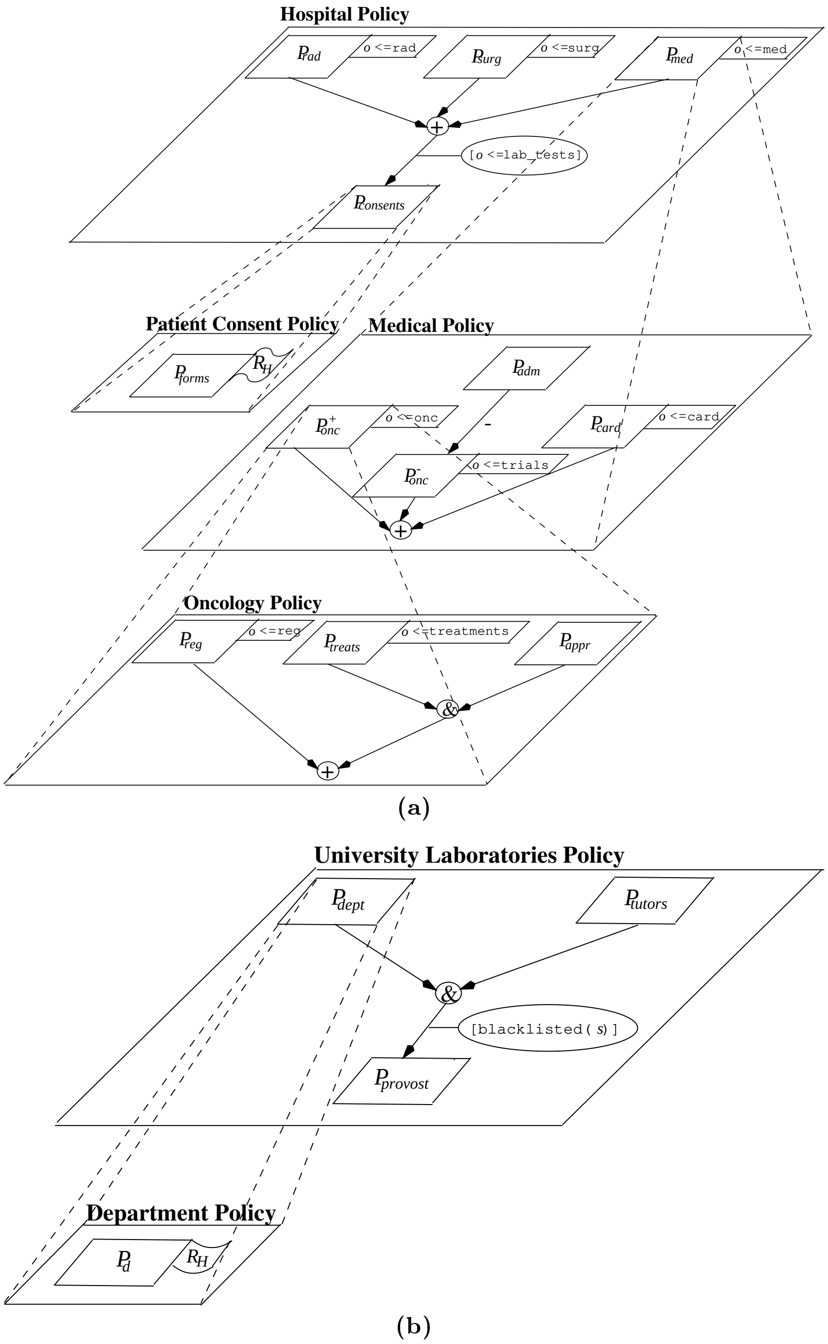


Fig. 2. A policy regulating access to the hospital data (a) and to the university laboratories (b)

Authorizations in $P_{consents}$ are collected by the hospital administration by means of forms that patients fill in when admitted. Patients' consents can refer to single individuals (e.g., `John Doe` can individually point out that his daughter `jane.doe` can access his tests) as well as to subject classes (e.g., `research_labs` and `hospitals`), and can refer to single documents or to classes of them. Authorizations specified for subject/object classes are propagated to individual users and documents by classical hierarchy-based derivation rules (see Section 7). Denoting such rules with R_H, we can express $P_{consents}$ as $P_{forms} * R_H$.

Policy P_{med} of the medical department is composed of the policies of its two divisions, `Cardiology` and `Oncology`, and of a policy P_{adm} specified by the central administration of the department. The `Oncology` division can revoke authorizations in P_{adm} regarding data related to clinic trials, by listing them in P^{-}_{onc}. In addition, each of the divisions can specify further authorizations (policies P^{+}_{onc} and P_{card}), whose scope is restricted to objects in their respective domains. The medical department's policy P_{med} can be expressed as $P_{adm} - P^{-}_{onc}\hat{}[o \leq \texttt{trials}] + P^{+}_{onc}\hat{}[o \leq \texttt{onc}] + P_{card}\hat{}[o \leq \texttt{card}]$. Let us take a closer look at the component policy P^{+}_{onc}. P^{+}_{onc} consists of two separate policies: P_{reg}, regulating access to the hospital cancer register; and P_{treats}, regulating access to experimental cancer treatments. In addition, access to experimental cancer treatments can be allowed only if the `Cancer Clinical Trials Office` (`CCTO`) has approved testing the treatments on patients. By representing approvals as policy P_{appr}, we can write policy P^{+}_{onc} as $P_{reg}\hat{}[o \leq \texttt{reg}] + (P_{treats}\hat{}[o \leq \texttt{treatments}] \& P_{appr})$.

Figure 2(a) illustrates the global policy regulating access to the hospital data and the content of the component policies discussed.

4.2 Example 2: University Laboratories

Consider a policy regulating access to university laboratories. To use machines, students must be authorized by both the laboratory tutors and the department administration. The tutors and the administration can specify authorizations at different levels of details. In particular, policy P_{tutors}, specified by the tutors, can state permissions on a single user-single machine basis, with statements like `(jim.smith,machine1,login)`. The department policy can state permissions P_d with reference to groups of students and machines, with statements like `(cs101,cs-lab,login)` which, closed under classical propagation rules R_H implies a permission for all students enrolled in class `cs101` to use machines in the `cs-lab`. Authorized accesses are defined as a conjunction of the two policies (intuitively, students should have permission to use the laboratory by the department and machine assignment by the tutor). In addition, access to any laboratory resource is forbidden to students blacklisted for infraction to rules (e.g., honor code); only an explicit permission by the provost can override such a restriction. The overall policy can thus be expressed as $o(P_{tutors} \& P_{dept}, P_{provost}, \hat{}[\texttt{blacklisted}(s)])$ whose graphical representation is illustrated in Figure 2(b).

5. PROPERTIES

The formal semantics on which the algebra is based allows us to reason about policy specifications and their properties, meaning correctness requirements that the resulting access control process should satisfy. For instance, consider the hospital policy above, some examples of correctness statements that need to be guaranteed may be:

(1) *patient awareness and hospital authorization:* nobody can access `lab_tests` data if there are not both the patient consent and the hospital authorization for it.
(2) *obedience to explicit denials:* if the oncology department has stated that an access should not be allowed, the access will not be.
(3) *law enforcement:* nobody can access new cancer treatments without the approval of the `CCTO`.

These correctness criteria must be satisfied regardless of the contents of the component policies. For instance, property 1 above must be satisfied regardless of which patients gave consents, what accesses the medical department wishes to permit, or how the hospital policy is formulated. Correctness criteria like the ones above can be easily proved exploiting the set theoretic semantics of the algebra. Intuitively, the correctness statements establish that given a template, whatever the structure or content of the specific policies in it, certain conditions are satisfied. The proofs use the formal semantics of expressions to determine whether or not the conditions will be satisfied. In particular, Propositions 5.1 through 5.3 below ensure us, by a simple analysis on the template, that the policy in Figure 2(a) satisfies properties 1, 2, and 3 above.

PROPOSITION 5.1. *Let $T = \tau(X\ Y).o(X, Y, \hat{}c)$ be a partially specified policy (template). For all policy expressions P_1 and P_2, environments e, and authorizations (s, o, a) satisfying c, $(s, o, a) \in [\![T(P_1, P_2)]\!]_e$ if and only if $(s, o, a) \in [\![P_1]\!]_e$ and $(s, o, a) \in [\![P_2]\!]_e$.*

PROOF. By definition, $[\![T(P_1, P_2)]\!]_e = ([\![P_1]\!]_e \setminus [\![P_1 \hat{}c]\!]_e) \cup ([\![P_2]\!]_e \cap [\![P_1 \hat{}c]\!]_e)$. Since (s, o, a) satisfies c (by hypothesis), we have $(s, o, a) \notin [\![P_1]\!]_e \setminus [\![P_1 \hat{}c]\!]_e$, either because $(s, o, a) \notin [\![P_1]\!]_e$, or because $(s, o, a) \in [\![P_1 \hat{}c]\!]_e$. It follows that $(s, o, a) \in T(P_1, P_2)$ iff $(s, o, a) \in [\![P_2]\!]_e \cap [\![P_1 \hat{}c]\!]_e$; in turn, since (s, o, a) satisfies c, this is equivalent to say that $(s, o, a) \in [\![P_2]\!]_e$ and $(s, o, a) \in [\![P_1]\!]_e$. $\square$

PROPOSITION 5.2. *Let $T = \tau(X\ Y\ Z\ W).(X \hat{}c_1 + (Y - Z \hat{}c_2) + W \hat{}c_3)$ be a partially specified policy such that $c_1 \wedge c_2$ and $c_2 \wedge c_3$ are not satisfiable. For all policy expressions P_1, P_2, P_3, P_4, environments e, and authorizations (s, o, a) satisfying c_2, if $(s, o, a) \in [\![P_3]\!]_e$, then $(s, o, a) \notin [\![T(P_1, P_2, P_3, P_4)]\!]_e$.*

PROOF. By definition, $[\![T(P_1, P_2, P_3, P_4)]\!]_e = [\![P_1 \hat{}c_1]\!]_e \cup ([\![P_2]\!]_e \setminus [\![P_3 \hat{}c_2]\!]_e) \cup [\![P_4 \hat{}c_3]\!]_e$. Suppose (s, o, a) satisfies c_2. Then, by hypothesis $(s, o, a) \notin [\![P_1 \hat{}c_1]\!]_e \cup [\![P_4 \hat{}c_3]\!]_e$. Moreover, if $(s, o, a) \in [\![P_3]\!]_e$, then by definition $(s, o, a) \notin ([\![P_2]\!]_e \setminus [\![P_3 \hat{}c_2]\!]_e)$. It follows that $(s, o, a) \notin [\![T(P_1, P_2, P_3, P_4)]\!]_e$. $\square$

PROPOSITION 5.3. *Let $T = \tau(X\ Y\ Z).(X \hat{}c_1 + (Y \hat{}c_2 \& Z))$ be a partially specified policy such that $c_1 \wedge c_2$ is not satisfiable. For all policy expressions P_1, P_2, P_3, environments e, and authorizations (s, o, a) satisfying c_2, $(s, o, a) \in [\![T(P_1, P_2, P_3)]\!]_e$ only if $(s, o, a) \in [\![P_3]\!]_e$.*

PROOF. By def., $[\![T(P_1, P_2, P_3)]\!]_e = [\![P_1 \hat{}c_1]\!]_e \cup ([\![P_2 \hat{}c_2]\!]_e \cap [\![P_3]\!]_e)$. Suppose that (s, o, a) satisfies c_2 and $(s, o, a) \in [\![T(P_1, P_2, P_3)]\!]_e$. Then, $(s, o, a) \notin [\![P_1 \hat{}c_1]\!]_e$, otherwise (s,o,a) would satisfy also c_1 contradicting the hypothesis. It follows that $(s,o,a) \in [\![P_2\hat{\ }c_2]\!]_e \cap [\![P_3]\!]_e \subseteq [\![P_3]\!]_e$. □

Note that had the scoping restriction attached to P_{reg} not been there, it would have not been possible to prove Proposition 5.3 and, indeed, property 3 would have not been satisfied anymore as stated by the following proposition.

PROPOSITION 5.4. *Let $T = \tau(X\ Y\ Z).(X+(Y\hat{\ }c\&Z))$ be a partially specified policy. There exist policy expressions P_1, P_2, P_3, an environment e, and an authorization (s,o,a) satisfying c such that $(s,o,a) \in [\![T(P_1,P_2,P_3)]\!]_e$ and $(s,o,a) \notin [\![P_3]\!]_e$.*

PROOF. Take three distinct policy identifiers as P_1, P_2, and P_3, and define $e(P_1) = \{(s,o,a)\}$, and $e(P_2) = e(P_3) = \emptyset$. By definition, $[\![T(P_1,P_2,P_3)]\!]_e = [\![P_1]\!]_e \cup [\![P_2\hat{\ }c\&P_3]\!]_e \supseteq [\![P_1]\!]_e = \{(s,o,a)\}$. Then, clearly, $(s,o,a) \in [\![T(P_1,P_2,P_3)]\!]_e$ and $(s,o,a) \notin P_3$. □

In our framework policies can also be analyzed to point out inherent inconsistencies. For instance, a bad formulation of a policy can always cause the result to be *empty*, whatever the structure or contents of its components. This is, for example, the case of a policy of the form $\tau(X\ Y).o(X\&Y, X-Y, Y)$, as stated by the following proposition.

PROPOSITION 5.5. *Let $T = \tau(X\ Y).o(X\&Y, X-Y, Y)$ be a partially specified policy. For all policy expressions P_1, P_2, and environments e, $[\![T(P_1,P_2)]\!]_e$ is an empty set.*

PROOF. By definition $[\![T(P_1,P_2)]\!]_e = (([\![P_1]\!]_e \cap [\![P_2]\!]_e) \setminus [\![P_2]\!]_e) \cup (([\![P_1]\!]_e \setminus [\![P_2]\!]_e) \cap [\![P_2]\!]_e)$. First note that $([\![P_1]\!]_e \cap [\![P_2]\!]_e) \setminus [\![P_2]\!]_e$ is empty, as no authorization can be in $[\![P_1]\!]_e \cap [\![P_2]\!]_e$ without being also in $[\![P_2]\!]_e$. Second, note that $([\![P_1]\!]_e \setminus [\![P_2]\!]_e) \cap [\![P_2]\!]_e$ is empty, as no authorization can be in $[\![P_1]\!]_e \setminus [\![P_2]\!]_e$ and $[\![P_2]\!]_e$ at the same time. The proposition immediately follows. □

Note that while the proofs are straightforward (and this is the beauty of the framework), the arguments are not. As authorization languages get more expressive and complete, it is not easy to ensure correctness. Think, for example, of recent logic-based authorization languages, where insertion of a rule or fact can cause an authorization to be derived without the security administrator be aware of it. Being able to state and prove correctness requirements in a very simple way is therefore a great advantage. The simplicity of the correctness statements and proofs is also due to the component-based view supported by the algebra, which can be exploited to reason at different levels of abstraction, considering only the relevant details.

6. EVALUATING POLICY EXPRESSIONS

The resolution of the expression defining a policy P determines a set of ground authorization terms corresponding to the accesses allowed by P. Different strategies can be used to evaluate expressions for enforcing access control. A possible strategy consists in completely resolving the expression (i.e., compiling the policy) and materializing the result as the set of allowed triples. The materialization, against which access control can be efficiently evaluated, will only need to be updated upon changes to the policy.[5] An alternative strategy consists in enforcing a run-time evaluation of each request (access triple) against the policy expression to determine whether the triple belongs to the result (i.e., whether the access should be allowed). Although this does not bear any cost for the complete resolution, it clearly makes the (much more frequent) process of controlling access requests more expensive. Between these two extremes, possibly combining the advantages of them, there are partial evaluation approaches, which can enforce different degrees of computation/materialization. Partial evaluation is particularly appealing as it can combine the advantages of the two solutions: relatively static and known policies can be precomputed and materialized, while more dynamic or unknown policies (like `CCTO` and Provost permissions in Section 4) can be evaluated at run-time.

Before describing access control in more details we illustrate a translation of algebraic expressions into equivalent logic programs, then used for access control enforcement. The main reason for using a logic-based approach is that logic programs provide executable specifications compatible with different evaluation strategies (e.g., Datalog bottom-up engines [Lloyd 1984], Prolog top-down evaluation [Lloyd 1984], XSB delayed evaluation and tabling [Sagonas et al. 2000], Hermes [Subrahmanian et al.]). In particular, partial evaluation techniques permit compilation of the static parts of the policies, thereby improving efficiency. Formal results on logic programs guarantee that the partial evaluation steps preserve the program semantics on the authorization predicates being compiled.

6.1 Translating algebra expressions into logic programs

We present a translation *pe2lp* from policy expressions into logic programs. *pe2lp* creates a distinct predicate symbol for each policy identifier and for each internal node in the syntax tree of the given algebraic expression. For this purpose, a means is needed to denote each operator occurrence (corresponding to different internal nodes); this is accomplished by labeling each such occurrence with a distinct integer, as in $P +_{\mathbf{3}} Q \&_{\mathbf{5}} S -_{\mathbf{2}} R$. Formally, such extended expressions will be called *labeled policy expressions.*

Definition Canonical Labeling. *The* canonical labeling *of a policy expression E is the labeled policy expression obtained by numbering the operators in E from left to right with contiguous integers, starting from* 0.

Definition Main Label. *The* main label *of a labeled policy expression E is the label of the outermost operator of E, that is, the label of the root of the syntax tree of E. If E is simply a policy identifier, then the main label of E is E itself.*

For instance, the canonical labeling of $P+o(Q\hat{\ }c, S, R)$ is $P+_{\mathbf{0}}o_{\mathbf{1}}(Q\hat{\ }_{\mathbf{2}}c, S, R)$. The main label of this formula is 0. Roughly speaking, the main label of E corresponds to the last operator evaluated, and hence to the "output" of E.

Translation *pe2lp* takes a labeled expression and an environment as input, and returns a logic program "equivalent to" the given expression, in a sense that will be formally specified later on. For each policy identifier P, a predicate $\mathtt{auth}_P$ is

[5]Incremental approaches can be applied to minimize the recomputation of the policy [Subrahmanian et al.].

E	$pe2lp(E^\ell, e)$
P	$\{\mathtt{auth}_P(s,o,a) \mid (s,o,a) \in e(P)\}$ if $e(P)$ is defined, $\emptyset$ otherwise.
$F +_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z),\ \mathtt{auth}_i(x,y,z) \leftarrow mainp_G(x,y,z)\}$ $\cup pe2lp(F,e) \cup pe2lp(G,e)$.
$F \&_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge mainp_G(x,y,z)\} \cup pe2lp(F,e) \cup pe2lp(G,e)$.
$F -_i G$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge \neg mainp_G(x,y,z)\} \cup pe2lp(F,e) \cup pe2lp(G,e)$.
$F \hat{}_i c$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge c\} \cup pe2lp(F,e)$
$o_i(F,G,M)$	$\{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge \neg mainp_M(x,y,z),$ $\mathtt{auth}_i(x,y,z) \leftarrow mainp_G(x,y,z) \wedge mainp_M(x,y,z)\}$ $\cup\, pe2lp(F,e) \cup pe2lp(G,e) \cup pe2lp(M,e)$.
$(\tau_i X.F)(G)$	$\{\mathtt{auth}_X(x,y,z) \leftarrow mainp_G(x,y,z)\} \quad \cup pe2lp(F,e) \cup pe2lp(G,e)$.
$F *_i R$	$\{\mathtt{auth}_i(s,o,a) \leftarrow \mathtt{trans}(L_1,i) \wedge \ldots \wedge \mathtt{trans}(L_n,i) \mid ((s,o,a) \leftarrow L_1 \wedge \ldots \wedge L_n) \in R\}$ $\cup \{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z)\} \cup pe2lp(F,e)$.

L	$\mathtt{trans}(L,i)$
(s',o',a')	$\mathtt{auth}_i(s',o',a')$
$p(x_1,\ldots,x_n)$	$p(x_1,\ldots,x_n)$

Fig. 3. Translation *pe2lp* : from policy expressions to logic programs

defined. For each labeled operator op_i, a predicate $\mathtt{auth}_i$ is created. All these predicates have three arguments, a subject, an object, and an action. The translation, for an expression E and an environment e, is recursively defined by the table in Figure 3, where P ranges over policy identifiers, and F, G, and M range over policy expressions. By $mainp_F$ we shall denote the predicate $\mathtt{auth}_\ell$, where ℓ is the main label of F.

Definition Canonical translation. *The* canonical translation *of a policy expression E w.r.t. an environment e is $pe2lp(E^\ell, e)$, where E^ℓ is the canonical labeling of E.*

Example 6.1. *Let $E = P + o(Q,S,R)$, and let e_0 be the environment mapping P to $\{(\mathsf{s}',\mathsf{o}',\mathsf{a}'),(\mathsf{s}'',\mathsf{o}'',\mathsf{a}'')\}$. Let Q,S,R be all undefined in e_0. The canonical translation of E w.r.t. e_0 is:*

$$
\begin{array}{l}
\mathtt{auth}_P(\mathsf{s}',\mathsf{o}',\mathsf{a}') \\
\mathtt{auth}_P(\mathsf{s}'',\mathsf{o}'',\mathsf{a}'') \\
\mathtt{auth}_0(x,y,z) \leftarrow \mathtt{auth}_P(x,y,z) \\
\mathtt{auth}_0(x,y,z) \leftarrow \mathtt{auth}_1(x,y,z) \\
\mathtt{auth}_1(x,y,z) \leftarrow \mathtt{auth}_Q(x,y,z) \wedge \neg\mathtt{auth}_R(x,y,z) \\
\mathtt{auth}_1(x,y,z) \leftarrow \mathtt{auth}_S(x,y,z) \wedge \mathtt{auth}_R(x,y,z)
\end{array}
$$

The above translation works correctly if the formal parameters of the templates occurring in E are all distinct. Formally, we say that E is *clash-free* if for all distinct sub-expressions $\tau X.P$, $\tau Y.Q$ of E, it holds that $X \neq Y$. Note that formal parameter names do not affect template semantics, so we can always rename parameters uniformly to avoid name clashes in the translation process, and the following proposition holds.

Proposition 6.1. *For each expression E there exists a clash-free expression E' such that E and E' are equivalent, that is, for all environments e, $[\![E]\!]_e = [\![E']\!]_e$.*

The following theorem tells us that *pe2lp* is semantics preserving. The logic program resulting from the translation must be interpreted according to the *stable model semantics* [Gelfond and Lifschitz 1988] (recalled in the appendix), or any other semantics (such as the well-founded semantics or the perfect model semantics) equivalent to the stable model semantics on *stratified programs* (i.e., programs that contain no recursion through negation), such as those produced by the translation. In the following the unique stable model of a stratified program P will be denoted by $\mathsf{sm}(P)$.

THEOREM 6.1. *Let B be a set of ground atoms defining the basic predicates and operators of $\mathcal{L}_{acon}$ and $\mathcal{L}_{rule}$. For all clash-free expressions E and all environments e defining all free identifiers in E,*

$$mainp_E(t,u,v) \in \mathsf{sm}(pe2lp(E^\ell, e) \cup B) \text{ iff } (t,u,v) \in [\![E]\!]_e \,.$$

In other words, in order to decide whether an authorization triple (t,u,v) is allowed by E, it suffices to evaluate the goal $mainp_E(t,u,v)$ in the program $pe2lp(E^\ell, e) \cup B$. The proof of this theorem, together with some preliminary concepts and notation, can be found in the appendix.

6.2 Access control enforcement

Before illustrating the use of logic programs to enforce access control, we need to specify how to treat "foreign" policies, that is, policies that may be expressed in a different language or stored at other sites. For each foreign policy, a wrapper should be provided [Subrahmanian et al.], that allows our logic programs to query the policy. This can be done with existing logic-based mediator techniques. For instance, using a HERMES-like syntax [Subrahmanian et al.], we may implement the link to an external policy P as $\mathtt{auth}_P(s,o,a) \leftarrow \mathbf{in}((s,o,a), P : \mathtt{grant}())$. Similarly, if the external policy specifies negative authorizations, we may write $\mathtt{auth}_{P^-}(s,o,a) \leftarrow \mathbf{in}((s,o,a), P : \mathtt{deny}())$. In the following, for each policy expression E and environment e, we denote by $\Pi_{E,e}$ the logic program consisting of the canonical translation $pe2lp(E^\ell, e)$ extended with the above wrapper rules for each foreign policy P.

We are now ready to discuss access control and related techniques for partial and complete materialization of policy expressions. In the discussion, we shall assume a given set of policy identifiers, Pol_{dyn}, containing the identifiers of policies that should not be materialized and the base predicates that cannot be evaluated at materialization time.

Partial materialization is accomplished by applying standard *partial evaluation* [Sterling and Shapiro 1997] techniques to the logic program $\Pi_{E,e}$. We recall that partial evaluation transforms program rules by iteratively applying *unfolding steps* of the form:

$$(A \leftarrow B_1, \ldots, B_n) \Rightarrow (A \leftarrow B_1, \ldots, B_{i-1}, C_1, \ldots, C_m, B_{i+1}, \ldots, B_n)\theta$$

where $H \leftarrow C_1, \ldots, C_m$ is a program clause whose variables are renamed with fresh variables, and θ is the *most general unifier* [Lloyd 1984] of B_i and H. The following predicates are not unfolded:

—Predicates $\mathbf{in}(\ldots, P : \ldots)$ such that $P \in \mathsf{Pol}_{dyn}$.

—Predicates of the form $\mathtt{auth}_P$ such that $P \in \mathsf{Pol}_{dyn}$, unless $\mathtt{auth}_P$ is implemented by a wrapper.

—Base predicates in Pol_{dyn}.

Eventually, rule bodies contain only predicates of this kind, and partial evaluation terminates. By well-known logic programming results, the following proposition holds ([Sterling and Shapiro 1997]):

PROPOSITION 6.2. *Let $\mathit{PartEv}(\Pi_{E,e})$ be the result of the partial evaluation of $\Pi_{E,e}$, and let $mainp_E$ be the main predicate of E's canonical translation. Then, for all ground authorizations (s, o, a), $\mathit{PartEv}(\Pi_{E,e}) \models mainp_E(s, o, a)$ iff $\Pi_{E,e} \models mainp_E(s, o, a)$.*

Intuitively, this proposition says that partial evaluation preserves the meaning of the original logic program. As a corollary of this proposition and Theorem 6.1, the partially evaluated (or *partially materialized*[6] captures exactly the meaning of the policy expression E in environment e.

COROLLARY 6.1. *For all ground authorizations (s, o, a), $\mathit{PartEv}(\Pi_{E,e}) \models mainp_E(s, o, a)$ iff $(s, o, a) \in [\![E]\!]_e$.*

EXAMPLE 6.2. *Consider the University Laboratory Policy in Figure 2(b). Assume that, at materialization time, the policies P_{tutors} and P_{dept} are known and consist of $\{(s_1, o_1, a_1), \ldots, (s_n, o_n, a_n)\}$ and $\{(s_1, o_1, a_1), \ldots, (s_k, o_k, a_k)\}$ $(k > n)$, respectively; while $P_{provost}$ policy and predicate* `blacklisted` *are unknown, meaning $\mathit{Pol}_{dyn} = \{P_{provost}, \mathtt{blacklisted}\}$. The canonical translation of the policy, w.r.t. the environment e binding P_{tutor} and P_{dept} to the above triples and leaving policy $P_{provost}$ and predicate* `blacklisted` *undefined, is:*

$$
\begin{array}{l}
\mathtt{auth}_0(x,y,z) \leftarrow \mathtt{auth}_1(x,y,z) \land \neg\mathtt{auth}_3(x,y,z) \\
\mathtt{auth}_0(x,y,z) \leftarrow \mathtt{auth}_{provost}(x,y,z) \land \mathtt{auth}_3(x,y,z) \\
\mathtt{auth}_1(x,y,z) \leftarrow \mathtt{auth}_{tutors}(x,y,z) \land \mathtt{auth}_{dept}(x,y,z) \\
\mathtt{auth}_2(x,y,z) \leftarrow \mathtt{auth}_{tutors}(x,y,z) \land \mathtt{auth}_{dept}(x,y,z) \\
\mathtt{auth}_3(x,y,z) \leftarrow \mathtt{auth}_2(x,y,z) \land \mathtt{blacklisted}(x) \\
\mathtt{auth}_{tutor}(s_i,o_i,a_i) \quad (i = 1,\ldots,n) \\
\mathtt{auth}_{dept}(s_i,o_i,a_i) \quad (i = 1,\ldots,k)
\end{array}
$$

We extend this program with the wrapper rule

$$\mathtt{auth}_{provost}(x,y,z) \leftarrow \mathbf{in}((x,y,z), P_{provost}\colon \mathtt{grant}()).$$

Then, we partially evaluate this program, obtaining:

$$
\begin{array}{l}
\mathtt{auth}_0(s_i,o_i,a_i) \leftarrow \neg\mathtt{blacklisted}(s_i) \\
\mathtt{auth}_0(s_i,o_i,a_i) \leftarrow \mathbf{in}((s_i,o_i,a_i), P_{provost}\colon \mathtt{grant}()) \ \land \mathtt{blacklisted}(s_i) \\
\mathtt{auth}_3(s_i,o_i,a_i) \leftarrow \mathtt{blacklisted}(s_i)
\end{array}
$$

where $i = 1, \ldots, n$.
Note that several intermediate predicate calls have been removed from the partially evaluated (or materialized) program.

[6]Complete materialization is a special case of partial evaluation, where Pol_{dyn} is empty and every predicate is unfolded.

7. ELEMENTARY POLICY SPECIFICATION

Our algebra can combine policies stated in different languages and through different paradigms. The algebra is therefore not substitutive of authorization languages, but complements them by allowing different specifications to be merged together, and combined according to different options. The independence from and support for the coexistence of different authorization languages and control mechanisms is a considerable advantage of our approach.

We note however, that when no other authorization language is being applied already, the constructs of our algebra can be also used to specify elementary policies.

In particular, the traditional *closed* policy can be expressed as a policy "P" listing the authorization triples corresponding to the accesses to be allowed; while the *open* policy can be expressed as a policy "$P_{all} - P$", where P_{all} is bound to the set of all possible authorizations and P contains the accesses to be denied. Authorizations for user groups and data types that propagate to their members can be simply expressed in our framework through rules that enforce authorization propagation along the hierarchy. The authorization specifications, stated with respect to individual elements (e.g., users or documents) or classes thereof (e.g., user groups or directories), would then be closed by the set of derivation rules $R = \{(s,o,a) \leftarrow (s',o,a), s \leq s', (s,o,a) \leftarrow (s,o',a), o \leq o', (s,o,a) \leftarrow (s,o,a'), a \leq a'\}$, where $\leq$ reflects the order defined on the different dimensions [Jajodia et al. 2001]. Derivation of authorizations according to criteria other than hierarchical relationships can be enforced in an analogous way.

Recent authorization models support both positive and negative authorizations. Positive authorizations state permissions whereas negative authorizations state denials [Lunt 1989]. The way positive and negative authorizations interplay is established by overriding/conflict resolution rules that may depend on the hierarchical relationships of the authorization elements and/or on priorities (or types) associated with the authorizations. Although we do not explicitly support negative authorizations, our framework can indeed express denials through the subtraction operator (negative authorizations appear as a policy to be removed). For instance, a specification supporting both permissions and denials and enforcing a *denial takes precedence* policy can be represented by expression "$P^+ - P^-$" where P^+ are the positive authorization terms and P^- are the authorization terms to be negated. Conflict resolution policies based on the hierarchies of the data system can be supported in an analogous way. As an example, the *most specific takes precedence* policy w.r.t. a hierarchy is obtained by computing, for each node the sum of its positive statements minus the negative statements for all its descendants, and summing up all the triples returned for each node. Here, by statements we mean the authorization triples closed under the propagation rules for the considered hierarchy. Formally, let i be the different nodes in the hierarchy w.r.t. which the policy is stated[7], P_i^+ and P_i^- the authorization triples corresponding to permissions and denials referred to i, and R_H the hierarchy-based propagation rules. Then, the most specific takes precedence policy w.r.t. hierarchy H is de-

[7]The hierarchy can be the hierarchy of subjects, objects, or actions or a combination of them [Jajodia et al. 2001].

fined as $\sum_{i \in H}(P_i^+ * R_H - \sum_{j \leq i} P_j^- * R_H)$.[8] Alternatively, the most specific takes precedence principle can be achieved by closing the given authorizations under suitable propagation rules enforcing the criteria [Jajodia et al. 2001]. As another example, consider the Orion authorization model [Rabitti et al. 1991], where authorizations propagate down the hierarchies and are classified as *strong* or *weak*. Strong authorizations (guaranteed to be free of conflict among themselves) override weak authorizations. Conflicts between weak authorizations are solved according to the most specific takes precedence policy. The overall policy can be stated as "$P_{weak} + P_{strong}^+ * R_H - P_{strong}^- * R_H$", where P_{strong}^+ and P_{strong}^- are the positive and negative strong authorizations, respectively, R_H are the hierarchy-based propagation rules, and P_{weak} is the set of triples resulting from applying the most specific takes precedence policy to weak authorizations as stated above.

Approaches enforcing further overriding criteria, for example, inclusion of organizational-level vs site-level authorizations [Damiani et al. 2000], or explicit priorities, such as the order in which authorizations are listed [Shen and Dewan 1992], can be expressed in a similar way. Intuitively, in the expressions enforcing authorizations w.r.t. a given criteria, triples denoting permissions appear as policies to be added, while triples denoting denials appear as policies to be subtracted. The order in which policies appear in the expression determines which policy override which. Our framework is therefore able to support and combine different approaches existing in the literature. In this respect, algebraic expressions turn out to be very flexible: a new dimension/criterion to be taken care of is simply reflected in the introduction of one (or two, if negative authorizations are supported) operands in the expression. This has also advantages in terms of clarity and readability of the specifications.

8. EXPRESSIVENESS ANALYSIS W.R.T. FIRST-ORDER LOGIC

A policy expression defines a mapping from a set of input policies (the policy identifiers occurring in the expression) to an output policy (the expression's value). It is interesting to investigate which classes of mappings can be expressed within the composition algebra.

Similar expressiveness analyses have already been carried out for database query mappings. The first investigations of this kind characterized the relationships between relational expressions and first-order logic (FOL) (cf. [Kanellakis 1990]). In the following a similar expressiveness analysis is carried out for the policy composition algebra. It will turn out that the basic core of our algebra captures only a strict subset of FOL (while the relational calculus is equivalent to it). This is mainly due to the fact that policy expressions operate on a fixed relation schema (corresponding to authorization triples). An advantage of this feature is that certain important decision problems are decidable for policy expressions, as discussed later on.

We start by introducing the logical framework. Let $P_{all}, P_1, P_2, \ldots$ be policy

---

[8] In case of incomparable conflicts, this expression resolves in favor of positive authorizations. A denial takes precedence principle could be enforced by subtracting from the result the sum of the nonoverridden negative statements, obtained with the dual expression $\sum_{i \in H}(P_i^- * R_H - \sum_{j<i} P_j^+ * R_H)$.

identifiers, where P_{all} is bound to the set of all possible authorizations, and let $C_1, C_2, \ldots$ be unary constraint predicates. Constraint predicates model elementary properties of authorization triples, such as the scoping restrictions used in the examples. For instance, some C_i may represent the constraint $[o \leq \mathtt{surg}]$. Then $C_i(x)$ is satisfied by all authorization triples x whose object field is dominated by `surg` in the object hierarchy. Similarly, each policy identifier P_i can be identified with a unary predicate such that $P_i(x)$ holds iff the triple x belongs to policy P_i.

Now, from the basic domains S, O, and A, from the interpretation of constraint predicates, $\mathsf{satisfy}$, and from an environment e, one can obtain interpretation structures of the form

$$\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle$$

for the monadic [9] first-order language $\mathcal{L}$ induced by predicates $\{P_{all}, P_1, P_2, \ldots\}$ and $\{C_1, C_2, \ldots\}$. More precisely, the truth of closed[10] sentences is defined inductively as follows, where t ranges over ground authorization triples, F and G range over arbitrary closed formulae and H is a formula with one free variable:

$$\begin{aligned}
\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models P_i(t) \ &\text{ iff } \ t \in e(P_i) && (1)\\
\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models C_i(t) \ &\text{ iff } \ t \ \mathsf{satisfy} \ C_i && (2)\\
\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F \wedge G \ &\text{ iff } \ \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F \text{ and} && (3)\\
&\qquad \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models G &&\\
\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F \vee G \ &\text{ iff } \ \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F \text{ or} && (4)\\
&\qquad \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models G &&\\
\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models \neg F \ &\text{ iff } \ \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \not\models F && (5)\\
\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models \exists x.H \ &\text{ iff } \ \text{for some } t \in \mathsf{S} \times \mathsf{O} \times \mathsf{A}, && (6)\\
&\qquad \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models H[t/x] &&
\end{aligned}$$

where $H[t/x]$ denotes the result of replacing all free occurrences of x with t in H. Universal quantification ($\forall$) and implication ($\rightarrow$) have been omitted as they can be expressed in terms of the above connectives in the usual way.

Now that the logical framework has been defined, we are ready to formulate the notion of equivalence between algebra expressions and logical formulae.

Definition Equivalence. *Let E be a policy expression, and F be a formula in $\mathcal{L}$ with one free variable x. We say that E and F are equivalent iff for all environments e defined for all free identifiers of E, and for all relations* $\mathsf{satisfy}$,

$$[\![E]\!]_e = \{t \in \mathsf{S} \times \mathsf{O} \times \mathsf{A} \mid \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F[t/x]\}\,.$$

In other words, E is equivalent to F if all the triples in E satisfy F and vice versa.

Example 8.1. *It is not hard to see that the expression $E = P_1 + P_2$ is equivalent to the formula $F(x) = P_1(x) \vee P_2(x)$.*

[9] A first-order language is monadic if all the predicate symbols are unary.
[10] A formula is closed if it has no free variables.

We shall prove the equivalence between policy expressions and the fragment of $\mathcal{L}$ identified by the following definition.

DEFINITION 0-1 FORMULAE. *A* 0-1 formula F *is a formula of* $\mathcal{L}$ *such that each subformula of* F *(including* F *itself) has at most one free variable.*

Intuitively, the restriction on subformulae is due to the fact that each subexpression returns a policy and, moreover, policies have a fixed relation schema (they may only contain members of $\mathsf{S} \times \mathsf{O} \times \mathsf{A}$). When we move to the logical framework we must ensure a similar property. If a subformula contained two or more free variables (each corresponding to a triple), then the "output schema" of the subformula would consist of several triples and would not correspond to any single policy. We shall see later that subexpressions without free variables cause no difficulties.

We shall almost exclusively investigate the expressive power of closure-free expressions. The closure operator introduces two parameters, namely, the rule language and its semantics. In particular, negation as failure has been given numerous alternative semantics, with significantly different properties. An extensive expressiveness analysis taking into account such different semantics lies beyond the scope of the current paper—it is more like an issue about existing rule languages, while our main interest here concerns the expressive power of the basic core of the algebra. Accordingly, our first result is about closure-free expressions. They correspond to the quantifier-free, 0-1 fragment of monadic first-order logic.

THEOREM 8.1. *For each closure-free policy expression* E *there exists an equivalent quantifier-free 0-1 formula* F*. Conversely, for each quantifier-free 0-1 formula* F *there exists an equivalent closure-free policy expression* E*.*

PROOF. We first prove—by structural induction—that for each closure-free and template-free policy expression E there exists an equivalent quantifier-free 0-1 formula F.

Base case:

Let E be a policy identifier P_i. Then $[\![E]\!]_e = e(P_i)$. Moreover, from (1), $\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy}\rangle \models P_i(t)$ iff $t \in e(P_i)$. It follows immediately that E is equivalent to the 0-1 formula $P_i(x)$.

Induction step:

We assume that policy expressions E_1, E_2, and E_3 are equivalent to quantifier-free 0-1 formulas F_1, F_2, and F_3, respectively. Without loss of generality we assume that the free variable of F_1, F_2 and F_3 is x. Let e be an arbitrary environment. There are six possible cases.

—**Case 1:** $E = E_1 + E_2$. We have $t \in [\![E_1 + E_2]\!]_e$ iff either $t \in [\![E_1]\!]_e$ or $t \in [\![E_2]\!]_e$. In turn, by induction hypothesis, this holds iff either $\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy}\rangle \models F_1[t/x]$ or $\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy}\rangle \models F_2[t/x]$. By (4), this is equivalent to $\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy}\rangle \models F_1[t/x] \vee F_2[t/x]$. We conclude that $E_1 + E_2$ is equivalent to $F_1 \vee F_2$.

—**Case 2, 3:** By analogy with the previous case, it can be shown that $E_1 \& E_2$ is equivalent to $F_1 \wedge F_2$, and that $E_1 - E_2$ is equivalent to $F_1 \wedge \neg F_2$.

—**Case 4:** $E = E_1\hat{}C$. By definition, $t \in [\![E_1\hat{}C]\!]_e$ holds iff $t \in [\![E_1]\!]_e$ and t $\mathsf{satisfy}$ C. In turn, by induction hypothesis and (2), this is equivalent to

$\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F_1[t/x]$ and $\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models C(t)$, that is, $\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F_1[t/x] \wedge C(t)$. We can conclude that $E_1\hat{}C$ is equivalent to $F_1 \wedge C(x)$.

—**Case 5:** $E = o(E_1, E_2, E_3)$. $[\![o(E_1, E_2, E_3)]\!]_e$ is defined as $[\![(E_1 - E_2) + (E_2 \& E_3)]\!]_e$. From cases 2 and 3, it follows that $E_2 \& E_3$ and $E_1 - E_2$ are equivalent to $F_2 \wedge F_3$ and $F_1 \wedge \neg F_2$, respectively. From case 1, we can conclude that $o(E_1, E_2, E_3)$ is equivalent to $(F_2 \wedge F_3) \vee (F_1 \wedge \neg F_2)$.

—**Case 6:** $E = (\tau X.E')(E_1)$. This expression is equivalent to $E'[E_1/X]$. By exhaustively applying such transformations we can eliminate all templates, preserving the expression semantics. Then the proof for template-free expressions applies.

It follows that for all closure-free expressions there exists an equivalent quantifier-free 0-1 formula.

We are left to prove the opposite implication, that is, for each quantifier-free 0-1 formula F there exists an equivalent closure-free policy expression E. Also this proof is by structural induction.

Base case:
$P_i(x)$ is equivalent to P_i, while a constraint $C(x)$ is equivalent to $P_{all}\hat{}C$. The details are left to the reader.

Induction step:
We assume that formulas F_1 and F_2 are equivalent to closure-free policy expressions E_1 and E_2, respectively. There are three cases.

—**Case 1:** $F = \neg F_1$. It is easy to see that F is equivalent to $P_{all} - E_1$.

—**Case 2, 3:** $F_1 \vee F_2$ is equivalent to $E_1 + E_2$, and $F_1 \wedge F_2$ is equivalent to $E_1 \& E_2$, as shown in the first part of the proof.

$\square$

Capturing also quantifiers is not difficult, if we use the closure operator. Fortunately, there is no need to dig into sophisticated rule languages; indeed, one simple Horn clause interpreted with standard least Herbrand model semantics is enough (most logic programming semantics collapse to standard semantics on Horn clauses, so the choice of semantics is not an issue in this case).

THEOREM 8.2. *Let $R = \{(v, y, z) \leftarrow (v', y', z')\}$, where v, y, z, v', y' and z' are distinct variables. Suppose that for all subexpressions of E of the form $E' * R'$, $R' = R$. Then there exists an equivalent 0-1 formula F. Conversely, for each 0-1 formula F with one free variable there exists an equivalent policy expression E whose subformulae of the form $E' * R'$ satisfy $R' = R$.*

PROOF. (*First part*) Let E be a policy expression satisfying the hypothesis. We prove that an equivalent 0-1 formula F exists by structural induction on E. The base case can be proved as in Theorem 8.1. Similarly, the induction step can be proved as in Theorem 8.1 whenever E is not of the form $E' * R$. Finally, assume $E = E' * R$. By definition of R, for all environments e defined over the free identifiers of E,

$$[\![E]\!]_e = \begin{cases} \mathsf{S} \times \mathsf{O} \times \mathsf{A} & \text{if } [\![E']\!]_e \neq \emptyset, \\ \emptyset & \text{otherwise.} \end{cases} \tag{7}$$

By induction hypothesis there exists a 0-1 formula F' equivalent to E', with a unique free variable x. Let $F = (P_1(x) \vee \neg P_1(x)) \wedge \exists x.F'$. Note that

$$\begin{aligned} \{t \in \mathsf{S} \times \mathsf{O} \times \mathsf{A} \mid \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F[t/x]\} = \\ = \begin{cases} \mathsf{S} \times \mathsf{O} \times \mathsf{A} & \text{if } \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models \exists x.F' \\ \emptyset & \text{otherwise.} \end{cases} \end{aligned} \tag{8}$$

Moreover, since E' and F' are equivalent,

$$\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models \exists x.F' \text{ iff } [\![E']\!]_e \neq \emptyset\,.$$

From this equivalence, (7) and (8), it follows that E and F are equivalent.

(*Second part*) The proof is by structural induction on logical formulae. We need a stronger induction hypothesis: we shall prove simultaneously that

a) for each 0-1 formula F with one free variable there exists an equivalent policy expression E;

b) for each closed 0-1 formula F there exists a policy expression E such that for all environments e defined over the free identifiers of E, and for all relations $\mathsf{satisfy}$,

$$\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \not\models F \quad \text{iff} \quad [\![E]\!]_e = \emptyset\,, \tag{9}$$

$$\langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F \quad \text{iff} \quad [\![E]\!]_e = \mathsf{S} \times \mathsf{O} \times \mathsf{A}\,. \tag{10}$$

Let F be a 0-1 formula. Assume without loss of generality that F contains only existential quantifiers (as $\forall$ can be expressed in terms of $\exists$ and $\neg$). We prove a) first; therefore, assume F has one free variable x. The base case can be proved as in Theorem 8.1. The induction step can be proved as in Theorem 8.1 whenever the main connective of F is not a quantifier, and each immediate subformula of F has one free variable. We are left to prove the following cases:

—**Case 1:** F is $\exists y.F'[x]$. Note that y must be distinct from x (otherwise x would not be free, violating the assumption). Second, note that y cannot be free in $F'[x]$, otherwise $F'[x]$ would have two free variables, x and y, and F would not be 0-1, a contradiction. Since y is not free in $F'[x]$, F is equivalent to $F'[x]$. By induction hypothesis, there exists a policy expression E equivalent to $F'[x]$, hence to F.

—**Case 2:** F is $F_1[x] \wedge F_2$ or $F_2 \wedge F_1[x]$, where F_2 is closed and x is the unique free variable of F_1. By induction hypotheses a) and b), there exist two policy expressions E_1 and E_2 such that E_1 is equivalent to F_1, and E_2, F_2 satisfy (9) and (10). Let $E = E_1 \& E_2$. We have:

$$\begin{aligned} [\![E]\!]_e &= [\![E_1]\!]_e \cap [\![E_2]\!]_e \\ &= \{t \in \mathsf{S} \times \mathsf{O} \times \mathsf{A} \mid \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F_1[t/x]\} \\ &\quad \cap \begin{cases} \mathsf{S} \times \mathsf{O} \times \mathsf{A} & \text{if } \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models F_2 \\ \emptyset & \text{otherwise.} \end{cases} \\ &= \{t \in \mathsf{S} \times \mathsf{O} \times \mathsf{A} \mid \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy} \rangle \models (F_1[x] \wedge F_2)[t/x]\} \end{aligned}$$

that is, E is equivalent to F.

—**Case 3:** F is $F_1[x] \vee F_2$ or $F_2 \vee F_1[x]$, where F_2 is closed and x is the unique free variable of F_1. By analogy with the previous case, it can be shown that F is equivalent to $E = E_1 + E_2$, for some E_1 and E_2 such that E_1 is equivalent to F_1, and E_2, F_2 satisfy (9) and (10).

We are only left to prove induction hypothesis b). Therefore, assume F is closed. The following cases are possible:

—**Case 1:** F is $\exists y.F'$. First assume that y is not free in F'. Then F is equivalent to F' and hence—by induction hypothesis, there exists a policy expression E such that (9) and (10) are satisfied. Now assume y is free in F' (since F is 0-1, y is the only free variable of F'). By induction hyp. there exists E' equivalent to F'. By definition of equivalence and (6),

$$[\![E']\!]_e = \emptyset \text{ iff } \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy}\rangle \not\models \exists y.F' \,. \tag{11}$$

Moreover, $[\![E']\!]_e = \emptyset$ iff $[\![E]\!]_e = [\![E' * R]\!]_e = \emptyset$. From this and (11) we obtain (9). Furthermore, $[\![E]\!]_e = [\![E' * R]\!]_e \neq \emptyset$ iff $[\![E' * R]\!]_e = \mathsf{S} \times \mathsf{O} \times \mathsf{A}$. From this and (11) we obtain (10). Summarizing, (9) and (10) are satisfied by $E = E' * R$ and F.

—**Case 2:** F is $F_1 \wedge F_2$. By induction hypotheses a) and b), there exist two policy expressions E_1 and E_2 such that both E_1, F_1 and E_2, F_2 satisfy (9) and (10). Let $E = E_1 \& E_2$. We have:

$$\begin{aligned} [\![E]\!]_e &= [\![E_1]\!]_e \cap [\![E_2]\!]_e \\ &= \left\{ \begin{array}{ll} \mathsf{S} \times \mathsf{O} \times \mathsf{A} & \text{if } \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy}\rangle \models F_1 \\ \emptyset & \text{otherwise.} \end{array} \right\} \cap \\ &\quad\ \left\{ \begin{array}{ll} \mathsf{S} \times \mathsf{O} \times \mathsf{A} & \text{if } \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy}\rangle \models F_2 \\ \emptyset & \text{otherwise.} \end{array} \right. \\ &= \left\{ \begin{array}{ll} \mathsf{S} \times \mathsf{O} \times \mathsf{A} & \text{if } \langle \mathsf{S} \times \mathsf{O} \times \mathsf{A}, e, \mathsf{satisfy}\rangle \models F_1 \wedge F_2 \\ \emptyset & \text{otherwise.} \end{array} \right. \end{aligned}$$

and hence, E, F satisfy (9) and (10).

—**Case 3:** F is $F_1 \vee F_2$. By analogy with the previous case, it can be shown that F is equivalent to $E = E_1 + E_2$, for some E_1 and E_2 such that E_1, F_1 and E_2, F_2 satisfy (9) and (10).

—**Case 4:** F is $\neg F'$. By analogy with the previous case, it can be shown that F is equivalent to $E = P_{all} - E'$, for some E' such that E', F' satisfy (9) and (10).

□

Summarizing, closure-free policy expressions capture exactly the quantifier-free 0-1 fragment of monadic first-order logic. Quantifiers can be captured with the closure operator and one simple rule. Our results have the following interesting consequences:

—From the point of view of formal expressiveness, logical connectives are not needed within scoping restrictions (i.e., in the constraint language). Complex constraints can be simulated via basic constraints plus other algebra operators. For example, the logical formula $C_1(x) \wedge C_2(x)$ can be expressed in the algebra as $(P_{all}\hat{}C_1)\&(P_{all}\hat{}C_2)$, without resorting to constructs such as $P_{all}\hat{}[C_1 \wedge C_2]$ (nonetheless they might be appealing as syntactic sugar).

—Deciding whether a monadic logical property is entailed by (the first-order version of) a policy expression is a decidable problem, as monadic first-order logic is decidable.

The latter point has interesting consequences on important decision problems, such as *expression containment* and *expression equivalence*. The containment problem consists in checking whether given two policies E_1 and E_2, it holds that $[\![E_1]\!]_e \subseteq [\![E_2]\!]_e$, for all total environments e and all relations satisfy. Checking containment may be useful to verify that policy refinements and updates do not open security breaches. These problems are decidable for a large class of expressions, as proved by the next theorem.

THEOREM 8.3. *The containment problem is decidable for policy expressions satisfying the restriction on closures specified in Theorem 8.2.*

PROOF. Let E_1 and E_2 be two arbitrary policy expressions and let $F_1(x)$ and $F_2(x)$ be two logical formulae equivalent to E_1 and E_2, respectively. Checking containment is equivalent to check that the formula $\forall x(F_1(x) \rightarrow F_2(x))$ is valid (the connectives $\forall$ and $\rightarrow$ can be expressed in terms of those dealt with in the above results). This is a decidable problem, because the above formula is monadic. □

From the notion of containment we obtain a strong form of expression equivalence, according to which two expressions E_1 and E_2 are *strongly equivalent* iff E_1 is contained in E_2 and vice versa.[11] Checking the strong equivalence of two policy expressions may be useful for optimization purposes. Clearly, the following corollary holds:

COROLLARY 8.1. *Checking strong equivalence is decidable for policy expressions satisfying the restriction on closures specified in Theorem 8.2.*

It is worth noting that the analogous equivalence and containment problems for the relational algebra are not decidable. The above decidability results open the way to automated optimization and verification techniques.

9. EVALUATION W.R.T. THE DESIDERATA

Before concluding, we summarize how our approach addresses the different requirements discussed in Section 2.

(1) *Heterogeneous policies* can be supported either by exploiting the algebra constructs to represent the different policies (see Section 7) or by referring to heterogeneous policies through policy identifiers then interpreted by means of wrappers (see Section 6.2).

(2) *Unknown policies* are supported by means of policy identifiers that can remain unbound in the environment. The translation of expressions into logic programs and the application of partial evaluation techniques on them allow us to delay

[11] In Proposition 6.1 we introduced a weaker form of equivalence which holds w.r.t. a fixed relation satisfy.

the evaluation of unknown policies until run time, without need of redoing the whole computation, and guarantee the correctness of the resulting controls (see Section 6.2). Furthermore, templates allow the expression of partially specified policies in the formal semantics and the proofs of correctness properties on incomplete specifications (see Section 5).

(3) *Interference* of program rules and authorizations coming from different policies is controlled by restricting rule application to specific policies by means of the closure construct.

(4) *Expressiveness* is achieved by the different operators that easily allow the formulation of protection restrictions as illustrated in the examples and discussions contained herein.

(5) *Different abstraction levels* are naturally supported by the component-based approach. Each component may internally be structured in sub-components and security administrators can zoom in the different policies or look at a higher level view as desired (see Figure 2). Templates provide a formal means to operate on the different levels as one may simply look at the template (higher abstraction level) or at its actual parameters corresponding to the contents (zoom in) of the formal parameters.

(6) *Formal semantics* has been provided in Section 3. We have also illustrated how the algebra semantics can be exploited to reason about properties of the specifications and not only to implement them. Our algebra is implementation independent and can be used to design, analyze, and combine requirements in different systems.

10. CONCLUDING REMARKS

The main contributions of this paper can be summarized as follows. First, we have analyzed the problem of composing security policies in a modular and incremental fashion and identified six desiderata for a policy composition framework. Second, we then proposed an algebra of security policies as a composition language. Third, we proposed an implementation approach based on logic programming and partial evaluation techniques which we formally proved correct. Fourth, we provided an extensive preliminary analysis of the algebra. The analysis was based on four different evaluation measures:

—The algebra has been applied to example scenarios as a preliminary usability test. We have also shown how the algebra can be used to model and reason about incomplete specifications.

—In Section 7, we have reproduced with the algebra a number of specification styles (open and closed policies, inheritance, etc.); the algebra should principally function as glue to combine existing policies, rather than as a tool for building component policies, however the absence of directly competing formalisms encouraged us to compare the algebra with such traditional tasks as well.

—It has been checked that the algebra is compatible with the desiderata.

—We moved the first step toward a formal expressiveness analysis of the algebra, in the style of query language analysis. The analysis shows that the composition algebra with restricted closure operators is less powerful than the relational algebra, since it cannot manipulate multiple relation schemata (the composition algebra is specialized on authorization triples). An advantage of such specialization is that certain important decision problems (such as expression containment and equivalence) that are undecidable for query algebras, are decidable for the composition algebra. This opens the way to automated policy verification techniques.

Such a composite analysis sets up a basis for a rich assessment methodology tailored to policy composition frameworks. The methodology may be enriched with direct comparisons with other approaches, as soon as more will appear. Of course, the expressiveness analysis carried out in this paper is not meant to replace realistic case studies. It should rather be considered as a tool for preliminary feasibility analysis and fine-grained comparison of different approaches.

We recall that the influence of different rule languages on the expressiveness of the algebra has not been investigated in this paper. This is an interesting subject for future work. Further work to be carried out includes the consideration of administrative policies regulating the authority and interoperation of different parties involved in the specification of the components within a global policy; the analysis of incremental approaches to updating component policies; automated verification techniques to prove properties of the specifications; and the performance assessment of different partial evaluation techniques.

ACKNOWLEDGMENTS

The work reported in this paper was partially supported by the European Community within the Fifth (EC) Framework Programme under contract IST-1999-11791 – FASTER project.

APPENDIX

A. PROOF OF THEOREM 6.1

Some preliminary definitions and notation are needed. See [Lloyd 1984] and [Gelfond and Lifschitz 1988] for more details. Let $\mathsf{Ground}(P)$ denote the ground instantiation of program P. When P is a positive (i.e. negation free) program, let $\mathsf{lm}(P)$ denote the unique least Herbrand model of P. Recall that

$$\mathsf{lm}(P) = \bigcup_{i \geq 0} T_P^i(\emptyset)\,.$$

The Gelfond-Lifschitz transformation of P w.r.t. an Herbrand interpretation I, denoted by P^I, is obtained from $\mathsf{Ground}(P)$ by:

—deleting all rules with some negative literals $\neg A$ such that $A \in I$;

—deleting all negative literals from the remaining rules.

Then M is a *stable model* of P iff $M = \mathsf{lm}(P^M)$.

The stable model of a stratified program (i.e. programs with no recursion through negation) is unique and coincides with all the canonical models prescribed by the major competing semantics, such as the perfect model semantics and the well-founded semantics. We shall denote by $\mathsf{sm}(P)$ the unique stable model of a stratified program P.

Let $P \cup Q$ be a stratified program, such that the predicates defined in P do not occur in Q. Let $HU(Q)$ denote the Herbrand universe of Q (i.e. the set of ground atoms in the language of Q). Define $P^{[Q]}$ as the program obtained from $\mathsf{Ground}(P)$ by:

—deleting all rules containing a subgoal $A \in HU(Q) \setminus \mathsf{sm}(Q)$;

—deleting all rules containing a subgoal $\neg A$ s.t. $A \in \mathsf{sm}(Q)$;

—deleting all subgoals with an atom from $HU(Q)$ from the remaining rules.

Since the predicates defined in P do not occur in Q, $HU(Q)$ is a *splitting set* as defined in [Lifschitz and Turner 1994]. Moreover, $P^{[Q]}$ is nothing but a specialization of the program $e_{HU(Q)}(P, M_Q)$. Then the Splitting Theorem of [Lifschitz and Turner 1994] can be restated and specialized as follows:

LEMMA A.1. ([Lifschitz and Turner 1994]) *If $P \cup Q$ is a stratified program such that the predicates defined in P do not occur in Q, then*

$$\mathsf{sm}(P \cup Q) = \mathsf{sm}(P^{[Q]}) \cup \mathsf{sm}(Q)\,.$$

As a consequence of this result we have the following lemma.

LEMMA A.2. *Let $P \cup Q \cup R$ be a stratified program. Suppose that the predicates defined in P do not occur in $Q \cup R$, and that the predicates defined in Q do not occur in $P \cup R$. Then*

$$\mathsf{sm}(P \cup Q \cup R) = \mathsf{sm}(P \cup R) \cup \mathsf{sm}(Q \cup R)\,.$$

PROOF. Note that since the predicates defined in Q do not occur in P we have that no atom A in $\mathsf{Ground}(P)$ belongs to $HU(Q \cup R)$ and hence:

$$P^{[Q \cup R]} = P^{[R]}\,.$$

From this equality and Lemma A.1 derive:

$$\begin{aligned}\mathsf{sm}(P \cup Q \cup R) &= \mathsf{sm}(P^{[Q \cup R]}) \cup \mathsf{sm}(Q \cup R)\\ &= \mathsf{sm}(P^{[R]}) \cup \mathsf{sm}(Q \cup R)\\ &= \mathsf{sm}(P^{[R]}) \cup \mathsf{sm}(Q^{[R]}) \cup \mathsf{sm}(R)\\ &= \mathsf{sm}(P \cup R) \cup \mathsf{sm}(Q \cup R)\,.\end{aligned}$$

□

THEOREM 6.1 *Let B be a set of ground atoms defining the basic predicates and operators of $\mathcal{L}_{acon}$ and $\mathcal{L}_{rule}$. For all clash-free expressions E and all environments e defining all free identifiers in E,*

$$mainp_E(t,u,v) \in \mathsf{sm}(pe2lp(E^\ell, e) \cup B) \text{ iff } (t,u,v) \in [\![E]\!]_e\,.$$

PROOF. By structural induction on E.

Base case. $E = P$ where P is a policy identifier. By definition, $pe2lp(E^\ell, e) = \{\mathtt{auth}_i(t', u', v') \mid (t', u', v') \in e(P)\}$. This program and B are sets of ground atoms, so

$$\mathsf{sm}(pe2lp(E^\ell, e) \cup B) = pe2lp(E^\ell, e) \cup B\,.$$

Moreover, $mainp_E$ is $\mathtt{auth}_P$. It follows that $mainp_E(t,u,v) \in \mathsf{sm}(pe2lp(E^\ell,e) \cup B)$ iff $(t,u,v) \in e(P)$ iff $(t,u,v) \in [\![P]\!]_e = [\![E]\!]_e$.

Induction step. There are several possibilities:

a) $E^\ell = F \&_i G$, for some integer i (and $mainp_E = \mathtt{auth}_i$). Let

$$P = \{\mathtt{auth}_i(x,y,z) \leftarrow mainp_F(x,y,z) \wedge mainp_G(x,y,z)\}\,.$$

By definition of $pe2lp$, Lemma A.1 and Lemma A.2,

$$\begin{aligned}
&\mathsf{sm}(pe2lp(E^\ell,e) \cup B) = \mathsf{sm}(P \cup pe2lp(F,e) \cup pe2lp(G,e) \cup B) \\
&\quad = \mathsf{sm}(P^{[pe2lp(F,e) \cup pe2lp(G,e) \cup B]}) \cup \mathsf{sm}(pe2lp(F,e) \cup pe2lp(G,e) \cup B) \\
&\quad = \mathsf{sm}(P^{[pe2lp(F,e) \cup pe2lp(G,e) \cup B]}) \cup \mathsf{sm}(pe2lp(F,e) \cup B) \qquad (12) \\
&\qquad \cup\, \mathsf{sm}(pe2lp(G,e) \cup B)\,.
\end{aligned}$$

Note that $mainp_F$ occurs only in $pe2lp(F,e)$ and $mainp_G$ occurs only in $pe2lp(G,e)$, therefore (with the help of lemma A.2),

$$\begin{aligned}
mainp_F(t,u,v) &\in \mathsf{sm}(pe2lp(F,e) \cup pe2lp(G,e) \cup B) \\
&\text{iff } mainp_F(t,u,v) \in \mathsf{sm}(pe2lp(F,e) \cup B)\,, \qquad (13) \\
mainp_G(t,u,v) &\in \mathsf{sm}(pe2lp(F,e) \cup pe2lp(G,e) \cup B) \\
&\text{iff } mainp_G(t,u,v) \in \mathsf{sm}(pe2lp(G,e) \cup B)\,.
\end{aligned}$$

It follows by definition of $P^{[pe2lp(F) \cup pe2lp(G) \cup B]}$ that

$$\begin{aligned}
&P^{[pe2lp(F,e) \cup pe2lp(G,e) \cup B]} = \{\mathtt{auth}_i(t',u',v') \mid \\
&\qquad mainp_F(t',u',v') \in \mathsf{sm}(pe2lp(F,e) \cup pe2lp(G,e) \cup B) \\
&\qquad \text{and } mainp_G(t',u',v') \in \mathsf{sm}(pe2lp(F,e) \cup pe2lp(G,e) \cup B)\} \\
&\quad = \{\mathtt{auth}_i(t',u',v') \mid mainp_F(t',u',v') \in \mathsf{sm}(pe2lp(F,e) \cup B) \qquad (14) \\
&\qquad \text{and } mainp_G(t',u',v') \in \mathsf{sm}(pe2lp(G,e) \cup B)\}.
\end{aligned}$$

Now note that $\mathtt{auth}_i$ may not occur in $pe2lp(F,e) \cup pe2lp(G,e) \cup B$ (as labels are unique), therefore, by (12),

$$\begin{aligned}
&mainp_E(t,u,v) \in \mathsf{sm}(pe2lp(E^\ell,e) \cup B) \text{ iff} \\
&\qquad mainp_E(t,u,v) \in \mathsf{sm}(P^{[pe2lp(F,e) \cup pe2lp(G,e) \cup B]})\,.
\end{aligned}$$

Moreover, $P^{[pe2lp(F,e) \cup pe2lp(G,e) \cup B]}$ is a set of ground atoms and hence it coincides with its unique stable model. It follows from (14) that

$$\begin{aligned}
&mainp_E(t,u,v) \in \mathsf{sm}(pe2lp(E^\ell,e) \cup B) \\
&\qquad \text{iff} \quad mainp_F(t,u,v) \in \mathsf{sm}(pe2lp(F,e) \cup B) \\
&\qquad \text{and} \quad mainp_G(t,u,v) \in \mathsf{sm}(pe2lp(G,e) \cup B)\,.
\end{aligned}$$

By induction hypothesis, it follows that

$$mainp_E(t,u,v) \in \mathsf{sm}(pe2lp(E^\ell,e) \cup B) \text{ iff } (t,u,v) \in [\![F]\!]_e \cap [\![G]\!]_e = [\![F \& G]\!]_e\,.$$

b) When E^ℓ is $F +_i G$, $F -_i G$ or $o_i(F,G,G')$ the proof is analogous to the previous case.

c) $E^\ell = F\,\hat{}_i c$. Let $P = \{\mathtt{auth}_i(s,o,a) \leftarrow \mathit{mainp}_F(s,o,a) \wedge c\}$. By definition of *pe2lp* and Lemma A.1,

$$\begin{aligned} \mathsf{sm}(\mathit{pe2lp}(E^\ell, e) \cup B) &= \mathsf{sm}(P \cup \mathit{pe2lp}(F,e) \cup B) \\ &= \mathsf{sm}(P^{[\mathit{pe2lp}(F,e)\cup B]}) \cup \mathsf{sm}(\mathit{pe2lp}(F,e) \cup B)\,. \end{aligned} \tag{15}$$

Since $\mathtt{auth}_i$ does not occur in $\mathit{pe2lp}(F,e) \cup B$, (15) entails:

$$\begin{aligned} &\mathit{mainp}_E(t,u,v) \in \mathsf{sm}(\mathit{pe2lp}(E^\ell, e) \cup B) \text{ iff} \\ &\qquad \mathit{mainp}_E(t,u,v) \in \mathsf{sm}(P^{[\mathit{pe2lp}(F,e)\cup B]})\,. \end{aligned} \tag{16}$$

Note that

$$\begin{aligned} P^{[\mathit{pe2lp}(F,e)\cup B]} &= \{\mathtt{auth}_i(s,o,a)\theta \mid \\ &\quad \mathit{mainp}_F(s,o,a)\theta \in \mathsf{sm}(\mathit{pe2lp}(F,e) \cup B) \\ &\quad \text{and } c\theta \in \mathsf{sm}(\mathit{pe2lp}(F,e) \cup B)\} \\ &= \{\mathtt{auth}_i(s,o,a)\theta \mid \\ &\quad \mathit{mainp}_F(s,o,a)\theta \in \mathsf{sm}(\mathit{pe2lp}(F,e) \cup B) \\ &\quad \text{and } c\theta \in B\}\,. \end{aligned} \tag{17}$$

By induction hypothesis and the definition of B it follows that

$$\begin{aligned} P^{[\mathit{pe2lp}(F,e)\cup B]} &= \{\mathtt{auth}_i(s,o,a)\theta \mid (s,o,a)\theta \in [\![F]\!]_e \\ &\quad \text{and } (s,o,a)\theta \text{ satisfy } c\theta\}\,. \end{aligned} \tag{18}$$

From (16) and (18) we get

$$\mathit{mainp}_E(t,u,v) \in \mathsf{sm}(\mathit{pe2lp}(E^\ell, e) \cup B) \text{ iff } (t,u,v) \in [\![F\,\hat{}\,c]\!]_e\,.$$

d) $E^\ell = F *_i R$. Let

$$\begin{aligned} P = &\{\mathtt{auth}_i(s,o,a) \leftarrow \mathtt{auth}_i(s_1,o_1,a_1) \wedge \ldots \wedge \mathtt{auth}_i(s_n,o_n,a_n) \wedge c_1 \wedge \ldots \wedge c_m \mid \\ &\quad ((s,o,a) \leftarrow (s_1,o_1,a_1) \wedge \ldots \wedge (s_n,o_n,a_n) \wedge c_1 \wedge \ldots \wedge c_m) \in R\} \\ &\cup \{\mathtt{auth}_i(s,o,a) \leftarrow \mathit{mainp}_F(s,o,a)\}\,. \end{aligned}$$

By analogy with the previous case,

$$\begin{aligned} &\mathit{mainp}_E(t,u,v) \in \mathsf{sm}(\mathit{pe2lp}(E^\ell, e) \cup B) \text{ iff} \\ &\quad \mathit{mainp}_E(t,u,v) \in \mathsf{sm}(P^{[\mathit{pe2lp}(F,e)\cup B]})\,. \end{aligned} \tag{19}$$

Note that by definition of $P^{[pe2lp(F,e)\cup B]}$ and by induction hypothesis,

$$\begin{aligned}
&P^{[pe2lp(F,e)\cup B]} = \\
&= \{(\mathtt{auth}_i(s,o,a) \leftarrow \mathtt{auth}_i(s_1,o_1,a_1) \wedge \ldots \wedge \mathtt{auth}_i(s_n,o_n,a_n))\theta \mid \\
&\quad ((s,o,a) \leftarrow (s_1,o_1,a_1) \wedge \ldots \wedge (s_n,o_n,a_n) \wedge c_1 \wedge \ldots \wedge c_m)\theta \in \mathsf{Ground}(R) \\
&\quad \text{and } \{c_1,\ldots,c_m\}\theta \subseteq B\} \\
&\cup\ \{\mathtt{auth}_i(s,o,a)\theta \mid \mathit{mainp}_F(s,o,a)\theta \in \mathsf{sm}(\mathit{pe2lp}(F,e)\cup B)\} \\
&= \{(\mathtt{auth}_i(s,o,a) \leftarrow \mathtt{auth}_i(s_1,o_1,a_1) \wedge \ldots \wedge \mathtt{auth}_i(s_n,o_n,a_n))\theta \mid \\
&\quad ((s,o,a) \leftarrow (s_1,o_1,a_1) \wedge \ldots \wedge (s_n,o_n,a_n) \wedge c_1 \wedge \ldots \wedge c_m)\theta \in \mathsf{Ground}(R) \\
&\quad \text{and } \{c_1,\ldots,c_m\}\theta \subseteq B\} \\
&\cup\ \{\mathtt{auth}_i(s,o,a)\theta \mid (s,o,a)\theta \in [\![F]\!]_e\}
\end{aligned}$$

Then

$$\begin{aligned}
\mathtt{auth}_i(t,u,v) \in \mathsf{sm}(P^{[pe2lp(F,e)\cup B]}) \ &\text{iff}\ (t,u,v) \in \mathsf{lm}(P \cup [\![F]\!]_e \cup B) \\
&= \mathsf{closure}(R, [\![F]\!]_e) = [\![F *_i R]\!]_e \,.
\end{aligned}$$

e) $E^\ell = (\tau_i X.F)(G)$. Let $P = \{\mathtt{auth}_X(x,y,z) \leftarrow \mathit{mainp}_G(x,y,z)\}$. Define

$$\begin{aligned}
Q &= P \cup \mathit{pe2lp}(G,e) \cup B \\
Q' &= \{\mathtt{auth}_X(t,u,v) \mid (t,u,v) \in [\![G]\!]_e\} \cup B \,.
\end{aligned}$$

It follows easily from Lemma A.1 that $\mathsf{sm}(Q)$ and $\mathsf{sm}(Q')$ agree on $\mathtt{auth}_X$ and on all the basic predicates and operators defined by B. As a consequence, $\mathsf{sm}(Q)$ and $\mathsf{sm}(Q')$ agree on all the predicates in $\mathit{pe2lp}(F,e)$, and hence,

$$\mathit{pe2lp}(F,e)^{[Q]} = \mathit{pe2lp}(F,e)^{[Q']} \,.$$

It follows that

$$\begin{aligned}
&\mathtt{auth}_i(t,u,v) \in \mathsf{sm}(\mathit{pe2lp}(E^\ell,e) \cup B) = \mathsf{sm}(\mathit{pe2lp}(F,e) \cup Q) \\
&\ \text{iff}\ \mathtt{auth}_i(t,u,v) \in \mathsf{sm}(\mathit{pe2lp}(F,e)^{[Q]}) = \mathsf{sm}(\mathit{pe2lp}(F,e)^{[Q']}) \\
&\ \text{iff}\ \mathtt{auth}_i(t,u,v) \in \mathsf{sm}(\mathit{pe2lp}(F,e) \cup Q') = \mathsf{sm}(\mathit{pe2lp}(F,e[X/[\![G]\!]_e]) \cup B) \,,
\end{aligned}$$

and hence, by induction hypothesis,

$$\mathtt{auth}_i(t,u,v) \in \mathsf{sm}(E^\ell,e) \ \text{iff}\ (t,u,v) \in [\![F]\!]_{e[X/[\![G]\!]_e]} = [\![(\tau_i X.F)(G)]\!]_e \,.$$

□

REFERENCES

ABADI, M. AND LAMPORT, L. 1992. Composing specifications. *ACM Transactions on Programming Languages 14*, 4 (October), 1–60.

BANISAR, D. AND DAVIES, S. 1999. *Privacy & Human Rights - An International Survey of Privacy Laws and Developments.* EPIC.

BERTINO, E., JAJODIA, S., AND SAMARATI, P. 1999. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems 17*, 2 (April), 101–140.

Bonatti, P., De Capitani di Vimercati, S., and Samarati, P. 2000. A modular approach to composing access control policies. In *7th ACM Conference on Computer and Communication Security (CCS 2000)* (Athens, Greece, November 2000), pp. 164–173.

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., and Samarati, P. 2000. Design and implementation of an access control processor for XML documents. In *Proc. of the WWW9 Conference* (Amsterdam, May 2000).

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of the International Conference on Logic Programming (ICLP'88)* (1988), pp. 1070–1080. MIT Press.

Hosmer, H. 1992. The multipolicy paradigm. In *Proc. of the 15th National Computer Security Conference* (Baltimore, MD, October 1992), pp. 409–422.

Jaeger, T. 1999. Access control in configurable systems. In J. Vitek and C. Jensen Eds., *Secure Internet Programming*, Volume 1603, pp. 289–310. Springer-Verlag.

Jajodia, S., Samarati, P., Sapino, M., and Subrahmanian, V. 2001. A unified framework for supporting multiple access control policies. *ACM Transactions on Database Systems*. To appear.

Kanellakis, P. 1990. Elements of relational database theory. In J. van Leeuwen Ed., *Handbook of Theoretical Computer Science*, Chapter 17. Elsevier Science Publishers.

Landwehr, C. 1981. Formal models for computer security. *Computing Surveys 13*, 3 (September), 247–278.

Li, N., Feigenbaum, J., and Grosof, B. 1999. A logic-based knowledge representation for authorization with delegation. In *Proc. of the 12th IEEE Computer Security Foundations Workshop* (Mordano, Italy, June 1999), pp. 162–174.

Lifschitz, V. and Turner, H. 1994. Splitting a logic program. In *Proc. of the International Conference on Logic Programming (ICLP'94)* (1994), pp. 23–37. MIT Press.

Lloyd, J. 1984. *Foundations of logic programming.* Springer-Verlag.

Lunt, T. 1989. Access control policies for database systems. In C. Landwehr Ed., *Database Security II: Status and Prospects*, pp. 41–52. North-Holland.

Rabitti, F., Bertino, E., Kim, W., and Woelk, D. 1991. A model of authorization for next-generation database systems. *ACM Transactions on Database Systems 16*, 1 (March), 89–131.

Sagonas, K., Swift, T., Warren, D., Freire, J., and Rao, P. 2000. The XSB programmer's manual, version 2.2. http://xsb.sourceforge.net.

Shen, H. and Dewan, P. 1992. Access control for collaborative environments. In *Proc. Int. Conf. on Computer Supported Cooperative Work* (Toronto, Canada, November 1992), pp. 51–58.

Sterling, L. and Shapiro, E. 1997. *The art of Prolog.* MIT Press, Cambridge, MA.

Subrahmanian, V., Adali, S., Brink, A., Emery, R., Lu, J., Rajput, A., Rogers, T., Ross, R., and Ward, C. Hermes: Heterogeneous reasoning and mediator system. http://www.cs.umd.edu/projects/hermes/publications/abstracts/hermes.html.

Woo, T. and Lam, S. 1993. Authorizations in distributed systems: A new approach. *Journal of Computer Security 2*, 2,3, 107–136.