An Algebra for Fine-Grained Integration of XACML Policies

Prathima Rao
Computer Science
Purdue University
prao@cs.purdue.edu

Dan Lin
Computer Science
Missouri S & T
lindan@mst.edu

Elisa Bertino
Computer Science
Purdue University
bertino@cs.purdue.edu

Ninghui Li
Computer Science
Purdue University
ninghui@cs.purdue.edu

Jorge Lobo
IBM T. J. Watson Research Center
jlobo@us.ibm.com

ABSTRACT

Collaborative and distributed applications, such as dynamic coalitions and virtualized grid computing, often require integrating access control policies of collaborating parties. Such an integration must be able to support complex authorization specifications and the fine-grained integration requirements that the various parties may have. In this paper, we introduce an algebra for fine-grained integration of sophisticated policies. The algebra, which consists of three binary and two unary operations, is able to support the specification of a large variety of integration constraints. To assess the expressive power of our algebra, we introduce a notion of completeness and prove that our algebra is complete with respect to this notion. We then propose a framework that uses the algebra for the fine-grained integration of policies expressed in XACML. We also present a methodology for generating the actual integrated XACML policy, based on the notion of Multi-Terminal Binary Decision Diagrams.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls

General Terms

Security

Keywords

access control, policy integration, XACML

1. INTRODUCTION

Many distributed applications such as dynamic coalitions and virtual organizations need to integrate and share resources, and these integration and sharing will require the integration of access control policies. In order to define a common policy for resources jointly owned by multiple parties applications may be required to integrate policies from different sources into a single policy. Even in a single organization, there could be multiple policy authoring units. If two different branches of an organization have different or even conflicting access control policies, what policy should the organization as a whole adopt? If one policy allows the access to certain resources, but another policy denies such access, how can they be composed into a coherent whole? Approaches to policy integration are also crucial when dealing with large information systems. In such cases, the development of integrated policies may be the product of a bottom-up process under which policy requirements are elicited from different sectors of the organization, formalized in some access control language, and then integrated into a global access control policy.

When dealing with policy integration, it is well known that no single integration strategy works for every possible situation, and the exact strategy to adopt depends on the requirements by the applications and the involved parties. An effective policy integration mechanism should thus be able to support a flexible fine-grained policy integration strategy capable of handling complex integration specifications. Some relevant characteristics of such an integration strategy are as follows. First, it should be able to support 3-valued policies. A 3-valued policy may allow a request, deny a request, or not make a decision about the request. In this case we say the policy is not applicable to the request. Three-valued policies are necessary for combining partially specified policies, which are very likely to occur in scenarios that need policy integration. When two organizations are merging and need policy integration, it is very likely that the organizations are unaware or might not have jurisdiction over each other resources, and thus a policy in one organization may be "NotApplicable" to requests about resources in the other organization. Second, it should allow one to specify the behavior of the integrated policy at the granularity of requests and effects. In other words, one should be able to explicitly characterize a set of requests that need to be permitted or denied by the integrated policy. For example, users may require the integrated policy to satisfy the condition that for accesses to an object O_i policy P_1 has the precedence, whereas for accesses to an object O_j, policy P_2 has precedence. Third, it should be able to handle *domain constraints* requiring the integrated policy to be applied to a restricted domain instead of the original domain. And fourth, it should be able to support policies expressed in rich policy languages, such as XACML with features like policy combining algorithms.

The problem of policy integration has been investigated in previous works. The concept of policy composition under constraints was first introduced by Bonatti et al. [7]. They proposed an algebra for composing access control policies and use logic programming and partial evaluation techniques for evaluating algebra expressions. Another relevant approach is by Wijesekera et

SACMAT'09, June 3–5, 2009, Stresa, Italy.

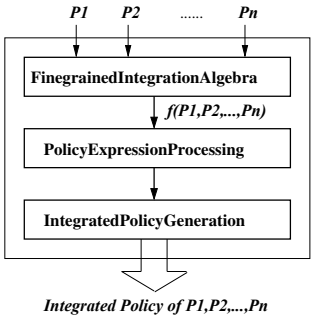

Figure 1: Policy integration

al. [21] who proposed a propositional framework for composing access control policies. Those approaches have however a number of shortcomings. They support only limited forms of compositions. For example, they are unable to support compositions that take into account policy effects or policy jumps (i.e., if P_1 permits, let P_2 makes decision, otherwise P_3 makes decision). They only model policies with two decision values, either "Permit" or "Deny". It is not clear the scope or expressive power of their languages since they do not have any notion of completeness. They do not provide an actual methodology or an implementation for generating the integrated policies. Neither work relates their formalisms to any language used in practice.

In this paper we propose a framework for the integration of access control policies that addresses the above shortcomings. The overall organization of our integration framework is outlined in Figure 1. The core of our framework is the *Fine-grained Integration Algebra* (FIA). Given a set of input policies P_1, P_2, $\cdots$, P_n, one is able to specify the integration requirements for these input policies through a FIA expression, denoted as $f(P_1, P_2, \cdots, P_n)$ in Figure 1. The FIA expression is then processed by the other components of the framework in order to generate the integrated policy. We demonstrate the effectiveness of our framework through an implementation that supports the integration of XACML policies. We choose XACML because of its widespread adoption and its many features, such as attribute-based access control and 3-valued policy evaluation. We use Multi-Terminal Binary Decision Diagrams (MTBDD) [10] for representing policies and generating the integrated policies in XACML syntax. With the aid of the implementation of algebra operators, users can now easily specify their integration requirements by an expression and do not need to write a new complex policy combining algorithm by themselves. The novel contributions of this paper can be summarized as follows:

- We propose a fine-grained integration algebra for language independent 3-valued policies. We introduce a notion of completeness and prove that our algebra is minimal and complete with respect to this notion.
- We propose a framework that uses the algebra for the fine-grained integration of policies expressed in XACML. The method automatically generates XACML policies as the policy integration result. To the best of our knowledge, none of the existing approaches has generated real policies as policy integration output.
- We have carried out experimental studies which demonstrate the efficiency and practical value of our policy integration approach.

The rest of the paper is organized as follows. Section 2 reviews related works on policy integration. Section 3 introduces background information on XACML and some preliminary definitions. Section 4 presents our fine-grained integration algebra. Section 5 discusses the expressiveness of the algebra. Section 6 presents detailed algorithms for generating well-formed integrated XACML policies. Section 7 reports our experimental study. Section 8 concludes the paper.

2. RELATED WORK

Many efforts have been devoted to policy composition [6, 7, 16, 19, 21, 12, 13, 8, 21]. However, for very few of these approaches, the expressive power has been analyzed. Moreover, none of them generates real policies as result of policy composition.

One early work on policy composition is the policy algebra proposed by Bonatti et al. [7], which aims at combining authorization specifications originating from heterogenous independent parties. They model an access control policy as a set of ground (variable-free) authorization terms, where an authorization term is a triple of the form (subject, object, action). However, their algebra only supports 2-valued policies and they do not clearly point out what authorization specifications can be expressed and what cannot by using their algebra. Regarding the algebra implementation, they suggest to use logic programming, but do not show any experimental result. Compared to their work, we have proved that our algebra can express any possible policy integration requirement and our implementation is based on representations used in model checking techniques which have been proven to be very efficient. Later, Jagadeesan et al. [13] proposed a 3-valued policy algebra using the timed concurrent constraint programming paradigm and define boolean operators whose expressive power is equivalent to the algebra in [7] in addition to added temporal features. However, this is only a theoretical work which does not have any implementation for generating the integrated policy.

Another related work is by Backes et al. [6] who proposed an algebra for combining enterprise privacy policies. They define conjunction, disjunction and scoping operations on 3-valued EPAL [5] policies. However, they did not prove the completeness. In other words, they cannot support some integration requirements that can be expressed by our algebra. Mazzoleni et al. [15] have proposed an extension to the XACML language, called *policy integration preferences*, using which a party can specify the approach that must be adopted when its policies have to be integrated with policies by other parties. They do not discuss mechanisms to perform such integrations. Also, the integration preferences discussed in such work are very limited and do not support fine-grained integration requirements as those presented in Section 5.2.

Most recently, Bruns et al. [8] proposed an algebra for four-valued policies based on Belnap bilattice. In particular, they map four possible policy decisions, i.e. grant, deny, conflict and unspecified, to Belnap bilattice and claim that their algebra is complete and minimal. However, such completeness is limited to Belnap space where policy decisions need to follow certain order according to the Belnap bilattice. Moreover, they did not propose any implementation of their algebra.

Our work is also related to the area of *many*-valued logics. Most work in such area focuses on establishing criteria for *Sheffer functions* in *m-valued* logic. A Sheffer function is a single logical function that is complete. Martin [14] isolates all binary sheffer functions in 3-valued logic and proves properties of such functions. Wheeler [20] proves a generalization of [14] and establishes the necessary and sufficient conditions for *n-nary* Sheffer functions in the context of 3-valued propositional calculus. Rousseau [18] provides further generalization and proves the necessary and sufficient conditions for any finite algebra with a single operation to be complete. Arieli et al. [4] propose a propositional language with four-

valued semantics and study the expressive power of their language. They also compare 3-valued and 4-valued formalisms. In contrast to these approaches, we do not find or establish criteria for all possible complete operators or functions for a 3-valued algebra. Instead, we focus on the definition of a set of operators that have intuitive semantic meaning in the context of combining 3-valued policies and study whether this set of operators is complete. We also study properties such as expressive power and minimality for this set of operators.

3. PRELIMINARIES

3.1 An Overview of XACML

XACML [2] is the OASIS standard language for the specification of access control policies. XACML policies include the following components: a *Target*, a *Rule* set, a *Rule combining algorithm* and a set of *Obligations*. The *Target* identifies the set of requests that the policy applies to. Each *Rule* consists of *Target*, *Condition* and *Effect* elements. The rule *Target* has the same structure as the policy *Target*. It specifies the set of requests that the rule applies to. The *Condition* element may further refine the applicability established by the target. The *Effect* element specifies whether the requested actions should be allowed ("Permit") or denied ("Deny"). The restrictions specified by the target and condition elements support the notion of attribute-based access control under which access control policies are expressed as conditions against the properties of subjects and protected objects. If a request satisfies both the rule target and rule condition predicates, the rule is applicable to the request and will yield a decision as specified by the *Effect* element. Otherwise, the rule is not applicable to the request and will yield a "NotApplicable" decision. If an error occurs to the evaluation of a rule, an "Indeterminate" decision is returned. The 3-valued algebra discussed in this paper is applicable to error free XACML policies where in the authorization decision is one of "Permit", "Deny" or "NotApplicable". However, the 3-valued algebra can be easily extended to a 4-valued algebra with similar properties that supports the "Indeterminate" decision [17].

The *Rule combining algorithm* is used to resolve conflicts among applicable rules with different effects. *Obligations*[1] represent a set of operations that must be executed in conjunction with an authorization decision. An obligation is associated with either "Permit" or "Deny" decision.

We now introduce an example of XACML policies that will be used throughout the paper.

EXAMPLE 1. *Consider a company with two departments D_1 and D_2. Each department has its own access control policies for the data under its control. Assume that P_1 and P_2 are the access control policies of D_1 and D_2 respectively. P_1 contains two rules, $P_1.Rul_{11}$ and $P_1.Rul_{12}$. $P_1.Rul_{11}$ states that the manager is allowed to read and update any data in the time interval [8am, 6pm]. $P_1.Rul_{12}$ states that any other staff is not allowed to read. P_2 also contains two rules, $P_2.Rul_{21}$ and $P_2.Rul_{22}$. $P_2.Rul_{21}$ states that the manager and staff can read any data in the time interval [8am, 8pm], and $P_2.Rul_{22}$ states that the staff cannot perform any update action. For simplicity, we adopt the following succinct representation in most discussion, where "role", "act" and "time" are attributes representing information on role, action and time, respectively.*

$P_1.Rul_{11}$: role=manager, act=read or update, time= [8am, 6pm], effect= Permit.
$P_1.Rul_{12}$: role=staff, act=read, effect = Deny.
$P_2.Rul_{21}$: role=manager or staff, act=read, time = [8am, 8pm], effect = Permit.
$P_2.Rul_{22}$: role=staff, act=update, effect = Deny.

[1]Due to limited space we have omitted discussion of methods for generating the correct set of obligations for an integrated policy in this paper. More details regarding this can be found in [17].

3.2 Definitions

Before we introduce our algebra we need to find a suitable definition for a *policy*. We propose a simple yet powerful definition for a policy according to which a policy is defined by the set of requests that are permitted by the policy and the set of requests that are denied by the policy. This simple notion will provide us with a precise characterization of the meaning of policy integration in terms of the sets of permitted and denied requests. In the rest of this paper, we use Y, N and NA to denote the "Permit", "Deny" and "NotApplicable" decisions respectively.

In our work, we assume the existence of a vocabulary Σ of attribute names and domains. Each attribute, characterizing a subject or an object or the environment, has a name a and a domain, denoted by $dom(a)$, in Σ. The following two definitions introduce the notion of access request (request, for short) and policy semantics.

DEFINITION 1. *Let a_1, a_2, ..., a_k be a set of attribute names and let $v_i \in dom(a_i)$ $(1 \leq i \leq k)$ in the vocabulary Σ. $r \equiv \{(a_1, v_1), (a_2, v_2), \cdots, (a_k, v_k)\}$ is a request over Σ. The set of all requests over Σ is denoted as R_Σ.*

EXAMPLE 2. *Consider policy P_1 from Example 1. An example of request to which this policy applies is that of a manager wishing to read any resource at 10am. According to Definition 1, such request can be expressed as $r \equiv$* {(role, manager), (act, read), (time, 10am)}.

DEFINITION 2. *A 3-valued access control policy P is a function mapping each request to a value in $\{Y, N, NA\}$. R^P_Y, R^P_N and R^P_{NA} denote the set of requests permitted, denied and not applicable by the policy P respectively, and $R_\Sigma = R^P_Y \cup R^P_N \cup R^P_{NA}$, $R^P_Y \bigcap R^P_N = \emptyset$, $R^P_Y \bigcap R^P_{NA} = \emptyset$, $R^P_N \bigcap R^P_{NA} = \emptyset$. We define a policy P as a triple $\langle R^P_Y, R^P_N, R^P_{NA}\rangle$.*

Our approach to formulating the definition of a policy is independent of the language in which access control policies are expressed. Therefore, our approach can be applied to languages other than XACML.

4. A FINE-GRAINED INTEGRATION ALGEBRA

The Fine-grained Integration Algebra (FIA) is given by $\langle \Sigma, \mathcal{P}_\mathsf{Y}, \mathcal{P}_\mathsf{N}, +, \&, \neg, \Pi_{dc}\rangle$, where Σ is the vocabulary of attribute names and their domains, $\mathcal{P}_\mathsf{Y}$ and $\mathcal{P}_\mathsf{N}$ are two policy constants, $+$ and $\&$ are two binary operators, and $\neg$ and Π_{dc} are two unary operators.

4.1 Policy Constants and Operators in FIA

We now describe the policy constants and operators in FIA. In what follows, $P_1 \equiv \langle R^{P_1}_Y, R^{P_1}_N, R^{P_1}_{NA}\rangle$ and $P_2 \equiv \langle R^{P_2}_Y, R^{P_2}_N, R^{P_2}_{NA}\rangle$ denote two policies to be combined, and $P_I \equiv \langle R^{P_I}_Y, R^{P_I}_N, R^{P_I}_{NA}\rangle$ denotes the policy obtained from the combination. Operators on policies are described as set operations.

Permit policy $(\mathcal{P}_\mathsf{Y})$. $\mathcal{P}_\mathsf{Y}$ is a policy constant that permits everything. Thus $\mathcal{P}_\mathsf{Y} \equiv \langle R_\Sigma, \emptyset, \emptyset\rangle$

Deny policy $(\mathcal{P}_\mathsf{N})$. $\mathcal{P}_\mathsf{N}$ is a policy constant that denies everything.

Thus $\mathcal{P}_{\mathsf{N}} \equiv \langle \emptyset, R_\Sigma, \emptyset \rangle$

Addition (+) . Addition of policies P_1 and P_2 results in a combined policy P_I in which requests that are permitted by either P_1 or P_2 are permitted, requests that are denied by one policy and are not permitted by the other are denied. More precisely:

$$P_I = P_1 + P_2 \iff \begin{cases} R_Y^{P_I} = R_Y^{P_1} \cup R_Y^{P_2} \\ R_N^{P_I} = (R_N^{P_1} \backslash R_Y^{P_2}) \cup (R_N^{P_2} \backslash R_Y^{P_1}) \end{cases}$$

A binary operator can be viewed as a function that maps a pair of values $\{Y, N, NA\}$ to one value. We give this view of addition, intersection, and two other derived binary operators to be introduced later in Table 1. A binary operator is represented using a matrix that illustrates the effect of integration for a given request r. The first column of each matrix denotes the effect of P_1 with respect to r and the first row denotes the effect of P_2 with respect to r.

Intersection (&) . Given two policies P_1 and P_2, the intersection operator returns a policy P_I which is applicable to all requests having the same decisions from P_1 and P_2. More precisely,

$$P_I = P_1 \,\&\, P_2 \iff \begin{cases} R_Y^{P_I} = R_Y^{P_1} \cap R_Y^{P_2} \\ R_N^{P_I} = R_N^{P_1} \cap R_N^{P_2} \end{cases}$$

The integrated policy makes a decision only when the two policies agree.

Negation (¬) . Given a policy P, $\neg P$ returns a policy P_I, which permits (denies) all requests denied (permitted) by P. The negation operator does not affect those requests that are not applicable to the policy. More precisely:

$$P_I = \neg P \iff \begin{cases} R_Y^{P_I} = R_N^{P} \\ R_N^{P_I} = R_Y^{P} \end{cases}$$

Domain projection (Π_{dc}) The domain projection operator takes as parameter the domain constraint dc and restricts a policy to the set of requests identified by dc.

DEFINITION 3. *A domain constraint dc takes the form $\{(a_1, range_1), (a_2, range_2), \cdots, (a_k, range_k)\}$[2], where a_1, a_2, ..., a_k are attribute names, and $range_i (1 \le i \le k)$ is a set of values in* dom*(a_i). Given a request $r = \{(a_{r_1}, v_{r_1}), \cdots, (a_{r_m}, v_{r_m})\}$, we say that r satisfies dc if the following condition holds: for each $(a_{r_j}, v_{r_j}) \in r$ $(1 \le j \le m)$ there exists $(a_i, range_i) \in dc$, such that $a_{r_j} = a_i$ and $v_{r_j} \in range_i$.*

The semantics of $\Pi_{dc}(P)$ is given by

$$P_I = \Pi_{dc}(P) \iff \begin{cases} R_Y^{P_I} = \{r | r \in R_Y^P \ and \ r \ satisfies \ dc\} \\ R_N^{P_I} = \{r | r \in R_N^P \ and \ r \ satisfies \ dc\} \end{cases}$$

4.2 FIA expressions

The integration of policies may involve multiple operators, and hence we introduce the concept of FIA expressions.

DEFINITION 4. *A FIA expression is recursively defined as follows:*
- If P is policy, then P is a FIA expression.
- If f_1 and f_2 are FIA expressions so are $(f_1) + (f_2)$, $(f_1) \,\&\, (f_2)$, and $\neg(f_1)$.
- If f is a FIA expression and dc is a domain constraint then $\Pi_{dc}(f)$ is a FIA expression.

[2]In case of an ordered domain, these sets can be represented by ranges.

In what follows we will use the terms "policy" and "expression" synonymously. In FIA expressions, the binary operators are viewed as left associative and unary operators are right associative. The precedence are $\neg$ and Π_{dc} together have the highest precedence, followed by $\&$, and then by $+$. For example, $P_1 + \Pi_{dc} P_2 + \neg P_3 \,\&\, P_4$ is interpreted as $((P_1 + (\Pi_{dc} P_2)) + ((\neg P_3) \,\&\, P_4)$. FIA has algebraic properties including commutativity, associativity, absorption, distributivity, complement, idempotence, boundedness and involution (proofs can be found in [17]).

4.3 Derived Operators

In this section, we introduce some commonly used operators. They are defined using the core operators.

Not-applicable policy ($\mathcal{P}_{\mathsf{NA}}$) . $\mathcal{P}_{\mathsf{NA}}$ is a policy constant that is not applicable for every request. Since the $\&$ operator applies only to requests that have common effects and $\mathcal{P}_{\mathsf{Y}}$ and $\mathcal{P}_{\mathsf{N}}$ have no requests with common effects, $\mathcal{P}_{\mathsf{Y}} \,\&\, \mathcal{P}_{\mathsf{N}}$ yields a policy that is not applicable for every request. Thus $\mathcal{P}_{\mathsf{NA}}$ can be defined as $\mathcal{P}_{\mathsf{Y}} \,\&\, \mathcal{P}_{\mathsf{N}}$.

Effect projection (Π_Y and Π_N) . $\Pi_Y(P)$ restricts the policy P to the requests allowed by it. It is defined as: $\Pi_Y(P) = P \,\&\, \mathcal{P}_{\mathsf{Y}}$. Similarly, $\Pi_N(P)$ restricts the policy P to the requests denied by it; it is defined as $\Pi_N(P) = P \,\&\, \mathcal{P}_{\mathsf{N}}$. We are overloading Π to denote both effect projection and domain projection; the meaning should be clear from the subscript.

Subtraction (−) . Given two policies P_1 and P_2, the subtraction operator returns a policy P_I which is obtained by starting from P_1 and limiting the requests that the integrated policy applies only to those that P_2 does not apply to. The subtraction operator is defined as:

$$P_1 - P_2 = (\mathcal{P}_{\mathsf{Y}} \,\&\, (\neg(\neg P_1 + P_2 + \neg P_2))) + (\mathcal{P}_{\mathsf{N}} \,\&\, (P_1 + P_2 + \neg P_2)) \,.$$

To see why this is correct, observe that $\neg P_1 + P_2 + \neg P_2$ will deny a request if and only if P_1 allows it and P_2 gives NA for it. Thus $\mathcal{P}_{\mathsf{Y}} \,\&\, (\neg(\neg P_1 + P_2 + \neg P_2))$ allows a request if and only if P_1 allows it and P_2 gives NA it, and is not applicable for all other requests. Similarly, $\mathcal{P}_{\mathsf{N}} \,\&\, (P_1 + P_2 + \neg P_2)$ denies a request if and only if P_1 denies it and P_2 gives NA for it.

Precedence ($\triangleright$) . Given two policies P_1 and P_2, the precedence operator returns a policy P_I which yields the same decision as P_1 for any request applicable to P_1, and yields the same decisions as P_2 for the remaining requests. The precedence operator can be expressed as $P_1 + (P_2 - P_1)$. By limiting P_2 to requests that P_1 does not decide, this operator can be used as a building block for resolving possible conflicts between two policies.

5. EXPRESSIVENESS OF FIA

In this section, we first show that our operators can express the standard policy-combining algorithms defined for XACML policies as well as other more complex policy integration scenario. We then show that the operators in FIA are minimal and complete in that any possible policy integration requirements can be expressed using a FIA expression. Finally, we discuss some interesting reasonability properties of FIA.

5.1 Expressing XACML Policy Combining Algorithms in FIA

In XACML there are six standard policy-combining algorithms as follows:

Permit-overrides : The combined result is "Permit" if any policy evaluates to "Permit", regardless of the evaluation result of the

$P_1 + P_2$

P_1 \ P_2	Y	N	NA
Y	Y	Y	Y
N	Y	N	N
NA	Y	N	NA

$P_1 \,\&\, P_2$

P_1 \ P_2	Y	N	NA
Y	Y	NA	NA
N	NA	N	NA
NA	NA	NA	NA

$P_1 - P_2$

P_1 \ P_2	Y	N	NA
Y	NA	NA	Y
N	NA	NA	N
NA	NA	NA	NA

$P_1 \triangleright P_2$

P_1 \ P_2	Y	N	NA
Y	Y	Y	Y
N	N	N	N
NA	Y	N	NA

Table 1: Policy combination matrix of operator $+, \&, -, \triangleright$

other policies. If no policy evaluates to "Permit" and at least one policy evaluates to "Deny", the combined result is "Deny". The combination of policies P_1, P_2,..., P_n under this policy-combining algorithm can be expressed as $P_1 + P_2 + \cdots + P_n$.

Deny-overrides : The combined result is "Deny" if any policy is encountered that evaluates to "Deny". The combined result is "Permit" if no policy evaluates to "Deny" and at least one policy evaluates to "Permit". Deny-overrides is the opposite of permit-overrides. By using the combination of the negation and addition operator, we can express deny-overrides as $\neg((\neg P_1) + (\neg P_2) + \cdots + (\neg P_n))$.

First-applicable : The combined result is the same as the result of the first applicable policy. This combining algorithm can be expressed by using the precedence operator. Given policies P_1, P_2, ..., P_n, the expression is $P_1 \triangleright P_2 \triangleright \cdots \triangleright P_n$.

Only-one-applicable : The combined result corresponds to the result of the unique policy in the policy set which applies to the request. Specifically, if no policy or more than one policies are applicable to the request, the result of policy combination should be "NotApplicable"; if only one policy is considered applicable, the result should be the result of evaluating the policy.

When combining policies $P_1, \cdots, P_n$ under this policy-combining algorithm, we need to remove from each policy the requests applicable to all the other policies and then combine the results using the addition operator. The final expression is : $(P_1 - P_2 - P_3 - \cdots - P_n) + (P_2 - P_1 - P_3 - \cdots - P_n) + \cdots + (P_n - P_1 - P_2 - \cdots - P_{n-1})$.

Note that the behaviour of *Ordered-Permit-overrides* and *Ordered-Deny-overrides* policy combining algorithms is exactly the same as *Permit-overrides* and *Deny-overrides* respectively except that the policies are evaluated in the originally specified order. The behaviour of the combined policy is the same in the unordered and ordered versions of *Permit(Deny)-overrides* except the set of obligations enforced might be different in the two versions. Thus *Ordered-Permit-overrides* and *Ordered-Deny-overrides* policy combining algorithms can be expressed using the same FIA expressions used for *Permit-overrides* and *Deny-overrides*.

5.2 Expressing Complex Policy Integration Requirements in FIA

Our algebra supports not only the aforementioned policy combining algorithms, but also other types of policy combining requirements, like rule constraints. A rule constraint specifies decisions for a set of requests. It may require that the integrated policy has to permit a critical request. Such an integration requirement can be represented as a new policy. Let P be a policy, and c be the policy specifying an integration constraint. We can combine c and P by using the first-applicable combining-algorithm. The corresponding expression is $c \triangleright P$. Another frequently used operator is to find the portion of a policy P_1 that differs from a policy P_2, which can be expressed as: $P_1 \,\&\, (\neg P_2)$.

By using the two policy constants, we can easily modify a policy P as an *open policy* or a *closed policy*. An open policy of P allows everything that is not explicitly denied, which can be represented as $P \triangleright \mathcal{P}_{\mathsf{Y}}$. A closed policy of P denies everything that is not explicitly permitted, which can be represented as $P \triangleright \mathcal{P}_{\mathsf{N}}$.

Our algebra can also express the policy jump(similar to if-then-else), a feature in the iptables firewall languages. The specific requirement is that if a request is permitted by policy P_1, then the final decision on this request is given by policy P_2; otherwise, the final decision is given by policy P_3. This can be expressed using

$$\Pi_Y(P_1 \,\&\, P_2) + \Pi_N(\neg P_1 \,\&\, P_2)) + \Pi_Y(\neg P_1 \,\&\, P_3) + \Pi_N(\neg P_1 \,\&\, \neg P_3))$$

Among the four sub-expressions, the first one gives Y when both P_1 and P_2 do so, and gives NA in all other cases. Similarly, the second sub-expression gives N when P_1 gives Y and P_2 gives N, and gives NA otherwise. The third sub-expression gives Y when P_1 gives N and P_3 gives Y and finally the fourth sub-expression gives N when both P_1 and P_3 give N.

Next, we elaborate the example mentioned in the introduction where the combination requirements are given for parts of a policy.

EXAMPLE 3. *Consider the policies introduced in Example 1. Assume that the policies must be integrated according to the following combination requirement: for users whose role is manager, the access has to be granted according to policy P_1; for users whose role is a staff, the access has to be granted according to policy P_2.*

The resultant policy will consist of two parts. One part is obtained from P_1 by restricting the policy to only deal with managers. Such extraction can be expressed in our algebra as $\Pi_{dc_1}(P_1)$ where dc_1 ={(role, {manager}), (act, {read,update}), (time, [8am,8pm])}. *The other part is obtained from P_2 by restricting the policy to only deal with staff. Correspondingly, we can use the expression: $\Pi_{dc_2}(P_2)$ with dc_2 =* {(role, {staff}), (act, {read,update}), (time, [8am,8pm])}. *Finally, we have the following expression representing the integrated policy : $\Pi_{dc_1}(P_1) + \Pi_{dc_2}(P_2)$. The integrated policy P_I is thus:*

$P_I.Rul_{I1}$: role=manager, act=read or update, time=[8am, 6pm], effect=Permit.
$P_I.Rul_{I2}$: role=staff, act=read, time=[8am, 8pm], effect=Permit.
$P_I.Rul_{I3}$: role=staff, act=update, effect=Deny.

5.3 Completeness

While we have shown that many policy integration scenarios can be handled by the operators in the algebra, our list of examples is certainly not exhaustive. A question of both theoretical and practical importance is whether FIA can express all possible ways of integrating policies, that is, whether FIA is *complete*. Addressing this question requires choosing a suitable notion of completeness. There are different degrees of completeness, and we show that FIA is complete in the strongest sense. First, while Table 1 gave the *policy combination matrices* for the four binary operators, many other matrices are possible, and each such matrix can be viewed as a binary operator for combining two policies. As there are three possibilities for each cell in a matrix, namely, Y, N, and NA, and there are nine cells, the total number of matrices is $3^9 = 19683$. Second, when n ($n \geq 2$) policies are combined, policy combination can be expressed using a n-dimensional matrix. We show that each such n-dimensional matrix can be expressed using $\langle \mathcal{P}_{\mathsf{N}}, \mathcal{P}_{\mathsf{Y}}, +, \&, \neg \rangle$. Finally, a fine-grained integration may use different policy combi-

$P_1, P_2, ..., P_{k-1}$	P_k	M^*	$f^{k-1}(P_1, P_2, ..., P_{k-1})$
$Y, Y, ..., Y$	Y	$e_{1,1}$	$f^{k-1}_{1,1}(P_1, P_2, ..., P_{k-1})$
...	...	...	...
$NA, NA, ..., NA$	Y	$e_{1,3^{k-1}}$	$f^{k-1}_{1,3^{k-1}}(P_1, P_2, ..., P_{k-1})$
$Y, Y, ..., Y$	N	$e_{2,1}$	$f^{k-1}_{2,1}(P_1, P_2, ..., P_{k-1})$
...	...	...	...
$NA, NA, ..., NA$	N	$e_{2,3^{k-1}}$	$f^{k-1}_{2,3^{k-1}}(P_1, P_2, ..., P_{k-1})$
$Y, Y, ..., Y$	NA	$e_{3,1}$	$f^{k-1}_{3,1}(P_1, P_2, ..., P_{n-1})$
...	...	...	...
$NA, NA, ..., NA$	NA	$e_{3,3^{k-1}}$	$f^{k-1}_{3,3^{k-1}}(P_1, P_2, ..., P_{k-1})$

(a)

P_k	$e_{i,j}$	$f^k_{i,j}$
Y	Y	$f^{k-1}_{i,j}\&(P_k\&\mathcal{P}_\mathsf{Y})$
Y	N	$f^{k-1}_{i,j}\&[\neg(P_k\&\mathcal{P}_\mathsf{Y})]$
N	Y	$f^{k-1}_{i,j}\&[\neg(P_k\&\mathcal{P}_\mathsf{N})]$
N	N	$f^{k-1}_{i,j}\&(P_k\&\mathcal{P}_\mathsf{N})$
NA	Y	$f^{k-1}_{i,j}$
NA	N	$f^{k-1}_{i,j}$

(b)

Table 2: n policies

nation matrices for different requests. We show that this can be handled by using the operator Π_{dc} in addition to $\langle \mathcal{P}_\mathsf{N}, \mathcal{P}_\mathsf{Y}, +, \&, \neg \rangle$.

THEOREM 1. *(Completeness) Given n ($n \geq 1$) policies P_1, P_2, ..., P_n, let $M^*(P_1, P_2, ..., P_n)$ be a n-dimensional policy combination matrix which denotes the combination result of the n policies. There exists a FIA expression $f_I(P_1, P_2, ..., P_n)$ that is equivalent to $M^*(P_1, P_2, ..., P_n)$.*

PROOF. We prove this theorem by induction. The base case is when $n = 1$. Given a policy P_1, its 1-dimensional matrix (Table 3) contains three entries corresponding to the Permit, Deny and NotApplicable request sets . For each entry, we aim to find an expression f_i ($1 \leq i \leq 3$). When e_1 is Y, $f_1 = \mathcal{P}_\mathsf{Y}\&P_1$; when e_1 is N, $f_1 = \neg(\mathcal{P}_\mathsf{Y}\&P_1)$. Similarly, we can obtain f_2, which is $\mathcal{P}_\mathsf{N}\&P_1$ for e_2 equal to N and $\neg(\mathcal{P}_\mathsf{N}\&P_1)$ for e_2 equal to Y. f_3 is $\mathcal{P}_\mathsf{Y} - P_1$ when e_3 is Y, and $\mathcal{P}_\mathsf{N} - P_1$ when e_3 is N. Finally, f_I is the sum of three expressions, i.e., f_I=f_1+f_2+f_3. Note that when all three entries are NA, the integrated policy will be $\mathcal{P}_\mathsf{NA}$.

P_1	Y	N	NA
P_I	e_1	e_2	e_3

Table 3: 1-dimensional Policy Combining Matrix

Assuming that when $n = k - 1$ the theorem holds, we now consider the case when $n = k$. As shown in Table 2(a), $M^*(P_1, ..., P_k)$ has 3^k entries in total, each of which is denoted as $e_{i,j}$ ($1 \leq i \leq 3, 1 \leq j \leq 3^{k-1}$). Take entries $e_{i,1}$ to $e_{i,3^{k-1}}$ as a *(k-1)*-dimensional policy combination matrix, and we have three such *(k-1)*-dimensional policy combination matrices corresponding to the policy P_k's effect. Based on the assumption, we obtain the FIA expression for each cell for the $k-1$ policies as shown in the column of $f^{k-1}(P_1, ..., P_{k-1})$.

Next, we extend $f^{k-1}(P_1, ..., P_{k-1})$ to $f^k(P_1, ..., P_k)$ for each cell in M^* (in what follows we use f^{k-1} and f^k for short). According to the effect of P_k and $e_{i,j}$, we summarize the expressions of f^k in Table 2(b). Note that we do not need to consider the cell where $e_{i,j}$ is NA.

Finally, we add up f^k for all the cells and obtain the expression $f(P_1, P_2, ..., P_k)$.

We have shown that the theorem holds for $n = 1$, and we have also shown that if the theorem holds for $n = k - 1$ then it holds for $n = k$. We can therefore state that it holds for all n. □

So far, we have proved the completeness in the scenario when there is one n-dimensional combination matrix for all requests. In the following theorem, we further consider the fine-grained integration when there are multiple combination matrices each of which is corresponding to a subset of the requests.

DEFINITION 5. *A* fine-grained integration specification *is given by $[(R_1, M^*_1), (R_2, M^*_2), \cdots, (R_k, M^*_k)]$, where $R_1, R_2, \cdots, R_k$ form a partition of $\mathcal{R}_\Sigma$ (the set of all requests over the vocabulary Σ), i.e., $\mathcal{R}_\Sigma = R_1 \cup R_2 \cup ... \cup R_k$ ($k \geq 1$) and $R_i \cap R_j = \emptyset$ when $i \neq j$, and each $M^*_i(P_1, .., P_n)$ ($1 \leq i \leq k$) is a n-dimensional policy combination matrix. This specification asks requests in each set R_i to be integrated according to the matrix M^*_i.*

THEOREM 2. *Given a fine-grained integration specification $[(R_1, M^*_1), (R_2, M^*_2), \cdots, (R_k, M^*_k)]$, if for each R_i, there exists $dc_{i,1}$, $\cdots$, dc_{i,m_i} such that $R_i = R(dc_{i,1}) \cup \cdots \cup R(dc_{i,m_i})$ (where $R(dc_{i,j})$ denotes the set of requests satisfying $dc_{i,j}$), then there exists a FIA expression $f_I(P_1, P_2, ..., P_n)$ that achieves the integration requirement.*

PROOF. We first use the domain projection operator Π_{dc} to project each policy according to $dc_{1,1}, \cdots, dc_{k,m_k}$. For requests in each $R(dc_{i,j})$, there is one fixed M^*_i. By Theorem 1, there is a FIA expression (denoted as $f_{i,j}$) for integrating policies $\Pi_{dc_{i,j}}(P_1)$, ..., $\Pi_{dc_{i,j}}(P_n)$ according to M^*_i. Finally, f_I is the addition of all $f_{i,j}$'s. □

We note that the above theorem requires that each R_i in the partition to be expressible in a finite number of domain constraints.

5.4 Minimal Set of Operators

Recall that FIA has $\{\mathcal{P}_\mathsf{Y}, \mathcal{P}_\mathsf{N}, +, \&, \neg, \Pi_{dc}\}$. The operator Π_{dc} is needed to deal with fine-grained integration. Operators $\{\mathcal{P}_\mathsf{Y}, \mathcal{P}_\mathsf{N}, +, \&, \neg\}$ are complete in the sense that any policy combination matrix can be expressed using them. A natural question is among the set $\Theta = \{\mathcal{P}_\mathsf{N}, \mathcal{P}_\mathsf{Y}, \mathcal{P}_\mathsf{NA}, +, \&, \neg, \Pi_Y, \Pi_N, -, \triangleright\}$, what subsets are *minimally complete*. We say a subset of Θ is minimally complete, if operators in the subset are sufficient for defining all other operators in Θ, and any smaller subset cannot define all operators in Θ. The following theorem answers this question. The only redundancy in $\{\mathcal{P}_\mathsf{Y}, \mathcal{P}_\mathsf{N}, +, \&, \neg\}$ is that only one of $\mathcal{P}_\mathsf{Y}$ and $\mathcal{P}_\mathsf{N}$ is needed.

THEOREM 3. *Among the 10 operators in Θ, there are 12 minimally complete subsets. They are the 12 elements in the cartesian product $\{\neg\} \times \{\mathcal{P}_\mathsf{Y}, \mathcal{P}_\mathsf{N}\} \times \{\Pi_Y, \Pi_N, \&\} \times \{+, \triangleright\}$.*

Proof of Theorem 3 can be found in [17].

6. INTEGRATED POLICY GENERATION

In this section, we present an approach to automatically generate the integrated policy given the FIA policy expression. Internally, we represent each policy as a Multi-Terminal Binary Decision Diagram (MTBDD) [10], and then perform operations on the underlying MTBDD structures to generate the integrated policy. We have

chosen an MTBDD based implementation of the proposed algebra because (i) MTBDDs have proven to be a simple and efficient representation for XACML policies [9] and (ii) operators in FIA can be mapped to efficient operations on the underlying policy MTBDDs. Our approach consists of three main phases:

1. **Policy representation**: For each policy P_i in the FIA expression $f(P_1, P_2, ..., P_n)$, we construct a policy MTBDD, T^{P_i}.
2. **Construction of the integrated policy MTBDD**: We combine the individual policy MTBDD structures according to the operations in the FIA expression to construct the *integrated policy MTBDD*.
3. **Policy generation**: The *integrated policy MTBDD* is then used to generate the actual integrated XACML policy.

6.1 Policy Representation

We can define a policy P as a function $P : R \rightarrow E$ from the domain of requests R onto the domain of effects E, where $E = \{Y, N, NA\}$.

An XACML policy can be transformed into a compound Boolean expression over request attributes [3]. A compound Boolean expression is composed of atomic Boolean expressions (AE) combined using the logical operations $\vee$ and $\wedge$. Atomic Boolean expressions that appear in most policies belong to one of the following two categories: (i) one-variable equality constraints, $a \triangleright c$, where a is an attribute name, c is a constant, and $\triangleright \in \{=, \neq\}$; (ii) one-variable range constraints, $c_1 \triangleleft a \triangleright c_2$, where a is an attribute name, c_1 and c_2 are constants, and $\triangleleft, \triangleright \in \{<, \leq\}$.

EXAMPLE 4. *Policy P_1 from Example 1 can be defined as a function :*

$$P_1(r) = \begin{cases} Y & \text{if } role = manager \wedge (act = read \vee \\ & \quad act = update) \wedge 8am \leq time \leq 6pm \\ N & \text{if role=staff } \wedge \text{ act=read} \\ NA & \text{Otherwise} \end{cases}$$

where r is a request of the form $\{(role, v_1), (act, v_2), (time, v_3)\}$.

We now encode each *unique* atomic Boolean expression AE_i in a policy into a Boolean variable x_i such that: $x_i = 0$ if AE_i is `false`; $x_i = 1$ if AE_i is `true`. To determine *unique* atomic Boolean expressions we use the following definition. The Boolean encoding for policy P_1 is given in Table 4.

i	AE_i	x_i
0	$role = manager$	x_0
1	$role = staff$	x_1
2	$act = read$	x_2
3	$act = update$	x_3
4	$8am \leq time \leq 6pm$	x_4

Table 4: Boolean encoding for P_1

Using the above Boolean encoding, a policy P can be transformed into a function $P : B^n \mapsto E$, over a vector of Boolean variables, $\vec{x} = x_0, x_1, \cdots, x_n$, onto the finite set of effects $E = \{Y, N, NA\}$, where n is the number of unique atomic Boolean expressions in policy P. A request r corresponds to an assignment of the Boolean vector $\vec{x}$, which is derived by evaluating the atomic Boolean expressions with attribute values specified in the request.

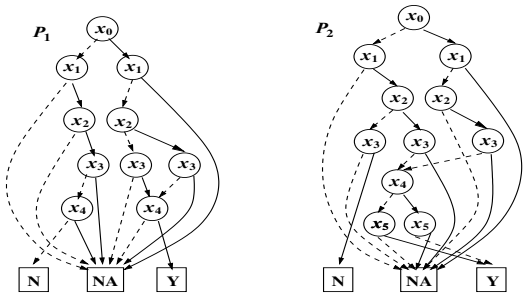

Figure 2: MTBDDs of P_1 and P_2

EXAMPLE 5. *After Boolean encoding, the policy P_1 is transformed into the function :*

$$P_1(\vec{x}) = \begin{cases} Y & \text{if } x_0 \wedge (x_2 \vee x_3) \wedge x_4 \\ N & \text{if } x_1 \wedge x_4 \\ NA & \text{Otherwise} \end{cases}$$

The transformed policy function can now be represented as a MTBDD. A MTBDD provides a compact representation of functions of the form $f : \mathbb{B}^n \mapsto \mathbb{R}$, which maps bit vectors over a set of variables ($\mathbb{B}^n$) to a finite set of results ($\mathbb{R}$). The structure of a MTBDD is a rooted acyclic directed graph. The internal (or non-terminal) nodes represent Boolean variables and the terminals represent values in a finite set. Each non-terminal node has two edges labeled 0 and 1 respectively. Thus when a policy is represented using a MTBDD, the non-terminal nodes correspond to the unique atomic Boolean expressions and the terminal nodes correspond to the effects. Each path in the MTBDD represents an assignment for the Boolean variables along the path, thus representing a request r. The terminal on a path represents the effect of the policy for the request represented by that path. Note that different orderings on the variables may result in different MTBDD representations and hence different sizes of the corresponding MTBDD representation. Several approaches for determining the variable ordering that results in an optimally sized MTBDD can be found in [11]. For examples discussed in this paper, we use the variable ordering $x_0 \prec x_1 \prec x_2 \prec x_3 \prec x_4$. The MTBDD of the policy P_1 is shown in Figure 2, where the dashed lines are 0-edges and solid lines are 1-edges.

Compound Boolean expression representing the policies to be integrated may have atomic Boolean expressions with matching attribute names but overlapping value ranges. In such cases, we need to transform the atomic Boolean expressions with overlapping value ranges into a sequence of new atomic Boolean expressions with disjoint value ranges, before performing the Boolean encoding. A generic procedure for computing the new atomic Boolean expression is described below.

Assume that the original value ranges of an attribute a are $[d_1^-, d_1^+]$, $[d_2^-, d_2^+]$, ..., $[d_n^-, d_n^+]$ (the superscript '-' and '+' denote lower and upper bound respectively). We sort the range bounds in an ascending order, and then employ a plane sweeping technique to obtain the disjoint ranges: $[d_1'^-, d_1'^+]$, $[d_2'^-, d_2'^+]$, ..., $[d_m'^-, d_m'^+]$, which satisfy the following three conditions: (i) $d_i'^-, d_i'^+ \in D$, $D = \{d_1^-, d_1^+, ..., d_n^-, d_n^+\}$; (ii) $\cup_{i=1}^{m}[d_i'^-, d_i'^+] = \cup_{j=1}^{n}[d_j^-, d_j^+]$; and (iii) $\cap_{i=1}^{m}[d_i'^-, d_i'^+] = \emptyset$.

Consider policy P_2 from Example 1. We can observe that the atomic Boolean expression $8am \leq time \leq 6pm$ in P_1 refers to the same attribute as in the atomic Boolean expression $8am \leq time \leq 8pm$ in P_2 and their value ranges overlap. In order to distinguish these two atomic Boolean expressions during the later

policy integration, we split the value ranges and introduce the new atomic Boolean expression $6pm \leq time \leq 8pm$. The expression $8am \leq time \leq 8pm$ in P_2 is replaced with $(8am \leq time \leq 6pm \vee 6pm \leq time \leq 8pm)$. Boolean encoding is then performed for the two policies by considering unique atomic Boolean expressions across both policies.

EXAMPLE 6. *By introducing another atomic Boolean expression $6pm \leq time \leq 8pm$, i.e. x_5, the transformed function for policy P_2 is :*

$$P_2(\vec{x}) = \begin{cases} Y, & \text{if } (x_0 \vee x_1) \wedge x_2 \wedge (x_4 \vee x_5) \\ N, & \text{if } x_1 \wedge x_3 \\ NA & \text{Otherwise} \end{cases}$$

Using the same variable ordering $x_0 \prec x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$ we construct the MTBDD for P_2 shown in Figure 2.

6.2 Construction of Integrated Policy MTBDD

Given the FIA expression $f(P_1, P_2, ..., P_n)$ and the MTBDD representations $T^{P_1}, T^{P_2}, ..., T^{P_n}$ of the policies P_1,P_2,...,P_n respectively, we construct the integrated policy MTBDD T^{P_I}, by performing the operations (specified in f) on the individual policy MTBDDs.

Operations on policies can be expressed as operations on the corresponding policy MTBDDs. Many efficient operations have been defined and implemented for MTBDDs [10]. In particular, we use the `Apply` operation defined on MTBDDs to perform the FIA binary operations $\{+, -, \&, \triangleright\}$ and `not` operation defined on MTBDD to perform the FIA unary negation ($\neg$) operation. We introduce a new MTBDD operation called `Projection` to perform the effect(Π_Y and Π_N) projection and domain projection(Π_{dc} operations defined in FIA.

The `Apply` operation combines two MTBDDs by a specified binary arithmetic operation. The `Apply` operation traverses each of the MTBDDs simultaneously starting from the root node. When the terminals of both MTBDDs are reached, the specified operation is applied on the terminals to obtain the terminal for the resulting combined MTBDD. A variable ordering needs to be specified for the `Apply` procedure.

The integrated MTBDD T^{P_I} for the policy expression $f(P_1, P_2) = P_1 + P_2$ is obtained by using MTBDD operation `Apply` $(T^{P_1}.root,\ T^{P_2}.root,\ +)$, where "root" refers to the root node of the corresponding MTBDD. Figure 3 shows the integrated policy MTBDD. The same variable ordering $x_0 \prec x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$ has been used in the construction of the integrated policy MTBDD.

The procedure for performing effect projection operations is the following. For Π_Y, those paths in T^P that lead to N are redirected to the terminal NA. Similarly, for Π_N, those paths in T^P that lead to Y are redirected to the terminal NA.

For the domain projection operation with domain constraint dc, we traverse the policy MTBDD from the top to the bottom and check the atomic Boolean expression associated with each node (denoted as $Node$). There are two cases. If the atomic Boolean expression of $Node$ contains an attribute specified in dc, we simply replace the attribute domain with the new domain given by dc. Otherwise, it means $Node$ represents an attribute no longer applicable to the resulting policy, and hence we should remove it. After removing $Node$, we need to adjust the pointer from its parent node by redirecting it to $Node$'s left child which leads to the path when N is not considered. After all nodes have been examined, those nodes that have no incoming edges are also removed.

Thus, given any arbitrary FIA expression $f(P_1, P_2, ..., P_n)$, we can use a combination of the `Apply`, `not`, `Projection` MTBDD operations on the policy MTBDDs to generate the integrated policy MTBDD. An example is given below.

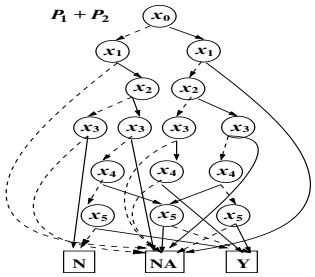

Figure 3: MTBDDs of $P_1 + P_2$

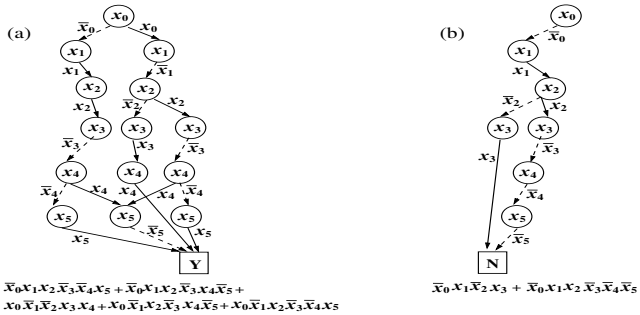

Figure 4: Policy generation using MTBDD

Consider the FIA policy expression for the only-one-applicable policy combining algorithm together with the domain constraint *dc* = {*(role, {manager}), (act, {read, update}), (time, [8am, 8pm])*}. Here, $f(P_1, P_2) = \Pi_{dc}((P_1 - P_2) + (P_2 - P_1))$. The integrated MTBDD can be obtained by using the `Apply` and `Projection` operations as follows :

`Projection`(`Apply`(`Apply`($T^{P_1}.root, T^{P_2}.root, -$),
`Apply`($T^{P_2}.root, T^{P_1}.root, -$), $+$), dc) .

6.3 XACML Policy Generation

In the previous section, we have presented how to construct the integrated MTBDD given any policy expression f. Though such integrated MTBDD can be used to evaluate requests with respect to the integrated policy, they cannot be directly deployed in applications using the access control system based on XACML. Therefore, we develop an approach that can automatically transform MTBDDs to actual XACML policies. The policy generation consists of three steps :

1. Find the paths in the combined MTBDD that lead to the Y and N terminals, and represent each path as a Boolean expression over the Boolean variable of each node.
2. Map the above Boolean expressions to the Boolean expressions on actual policy attributes.
3. Translate the compound Boolean expression obtained in step 2 into a XACML policy.

We first elaborate on step 1. In the MTBDD, each node has two edges, namely *0-edge* and *1-edge*. The 0-edge and 1-edge of a node labelled x_i correspond to *edge-expressions* $\bar{x}_i$ and x_i respectively. A path in the MTBDD corresponds to an expression which is the conjunction of *edge-expressions* of all edges along that path. We refer to this as a *path-expression*. Those paths leading to the same terminal correspond to the disjunction of *path-expressions*. Figure 4 shows an example of $P_1 + P_2$, where the Figure 4(a) and (b) show

PolicyId=P1+P2
<RuleId=R1 Effect=Deny>
<Target>
<Subject role=staff>
<Action act=update>
</Target>
</Rule>
<RuleId=R2 Effect=Deny>
<Target>
<Subject role=staff>
<Action act=read>
</Target>
<Condition time $\neq [8am, 6pm]$ AND time $\neq [6pm, 8pm]$>
</Rule>
<RuleId=R3 Effect=Permit> $\cdots$

Figure 5: The integrated XACML policy representing $P_1 + P_2$

the paths leading to the Y and N terminals and the corresponding Boolean expressions.

Next, we replace Boolean variables in the path-expressions with the corresponding atomic Boolean expressions by using the mapping constructed in the Boolean encoding phase. During the transformation in each path-expression, we need to remove some redundant information. For instance, the resulting expression may contain an attribute with both equality and inequality functions like $(role = manager) \wedge (role \neq staff)$. In that case, we only need to keep the equality function of the attribute.

EXAMPLE 7. *After the replacement, the Boolean expression of the N terminal in Figure 4 is transformed as follows:*
$(role = staff \wedge act = update) \vee (role = staff \wedge act = read \wedge time \neq [8am, 6pm] \wedge time \neq [6pm, 8pm])$

The last step is to generate the actual XACML policy from the compound Boolean expression obtained in previous step. Specifically, for each path-expression whose evaluation is Y, a permit rule is generated; and for each path-expression whose evaluation is N, a deny rule is generated. Attributes that appear in conditions of the rules in original policies still appear in conditions of the newly generated rules, and attributes that appear in targets in the original policies still appear in targets in the integrated policy. Here we do not distinguish the policy target with rule target. Instead, all targets appear as rule targets.

Consider policies P_1 and P_2 in Example 1, and the Boolean expression in Example 7. We generate the corresponding deny rules for the integrated policy of $P_1 + P_2$ as shown in Figure 5.

7. EXPERIMENTAL STUDY

We performed experiments to evaluate the time taken for performing FIA operations and the time for generating an integrated policy. We also examined the size of the generated integrated policy in terms of the number of rules and number of atomic Boolean expressions in each rule. All experiments were conducted on a Pentium III 3GHz 500 MB machine. MTBDD operations were implemented using the modified CUDD library developed in [9].

We implemented a random attribute based access control policy generator to generate XACML policies in Boolean form. Each policy contained atomic Boolean expressions on a set of predefined attribute names and values. The Boolean expressions corresponding to the *Condition* element of an XACML policy was derived by randomly concatenating atomic Boolean expressions with the logical $\vee$, $\wedge$ and $\neg$ operators. Each rule was randomly assigned to either permit or deny effect. Each policy was also associated with either a deny-override or permit-override rule combining algorithm.

In the first set of experiments we measured the average time required for performing the FIA operations and the size of the obtained MTBDDs. Figure 6 shows along the left y-axis the average time (in ms) for performing $+$ and $\triangleright$ operations on policies in which the total number of atomic Boolean expressions in a policy was fixed to 50 and the number of rules was varied between 2 and 10. This graph shows along the right y-axis the average size, i.e., the number of nodes, in the corresponding integrated MTBDDs. From Figure 6, we can observe that the average time taken to perform these operations increases with the increase in the size of the integrated MTBDDs, and it differs for different operators. The reason is that the actual time for performing operations depend on the size of the resulting integrated MTBDD. The larger the MTBDD is, the longer time the integration will take. Performing the $\triangleright$ operation usually resulted in MTBDDs with a smaller size and hence it took lesser time. Considering that for typical policies we have encountered in real world applications the average number of atomic Boolean expressions lies between 10 and 50, the time trends observed in Figure 6 is very encouraging.

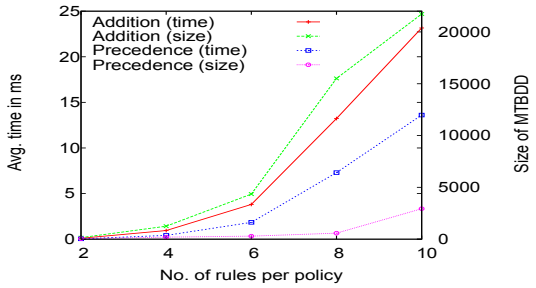

Figure 6: Average time and average integrated MTBDD size with respect to operations "+" and "$\triangleright$"

We have also evaluated other operations and policies with 10 rules [17], which demonstrate similar trends. Due to the limited space, we do not include them here.

In the second set of experiments, we studied the characteristics of the integrated policy. Because the number of rules generated in the integrated policy is equal to the number of paths which can be exponential in the size of the integrated MTBDD a large number of rules can be generated. We used ESPRESSO [1], a two level logic minimizer to reduce the number of rules. ESPRESSO uses state-of-the-art heuristic Boolean minimization algorithms to produce a minimal equivalent representation of two-valued or multiple-valued Boolean functions. The minterms obtained from the integrated MTBDD were transformed to ESPRESSO inputs and the minimized output was used for policy generation. Table 5 summarizes the results obtained for $+$ operation performed on data sets that contained policies with 4 and 8 rules with an average 20 atomic Boolean expressions per policy. We observe that using ESPRESSO a substantial decrease in the number of rules and atomic Boolean expressions(terms) was obtained. For the data sets used in our experiments we observed a 75% to 99% reduction in the number of rules and 35% to 71% reduction in the number of atomic Boolean expressions per rule.

Finally, we evaluated the time required for generating the integrated XACML policy. It is worth noting that this step is optional. Users can also directly use the integrated MTBDD for request evaluation. Figure 7 shows the policy generation times for integrated MTBDDs with different number of paths. The time for policy generation is observed to be proportional to the number of

		Without ESPRESSO		With ESPRESSO	
# of Rules in a Policy	Rule Type	Avg # of Rules	Avg # of terms per Rule	Avg # of Rules	Avg # of terms per Rule
4 Rules	Permit Deny	790 3625	19 21	69 233	9 6
8 Rules	Permit Deny	20192 131348	37 36	1221 152	19 11

Table 5: Characteristics of integrated policy

paths in the integrated MTBDD. The number of paths in the integrated MTBDD is in turn determined by the nature of the compound Boolean expressions in the policies and the chosen variable ordering.

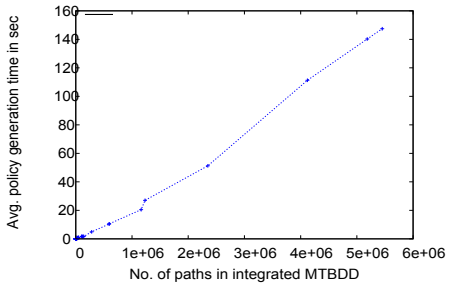

Figure 7: Average time for integrated policy generation

8. CONCLUSIONS

In this work we have proposed an algebra for the fine-grained integration of language independent policies. Our operations can not only express existing policy-combining algorithms but can also express any arbitrary combination of policies at a fine granularity of requests, effects and domains, as we have proved in the completeness theorem. We have implemented this algebra and our experimental results indicate that the FIA operations can be performed efficiently.

We plan to extend this work in several interesting directions. One direction involves the extension with respect to obligations. Another interesting question that arises when using such integration algebras is: how can one be sure that a given algebriac expression will behave as expected. We believe that we can leverage techniques from software engineering field to examine the integrated policy with a random number of requests and assure with some high probability that the expression indeed corresponds to the expected behaviour.

9. ACKNOWLEDGEMENTS

The work reported in this paper has been partially supported by the NSF grant 0712846 “IPS: Security Services for Healthcare Applications”, and MURI award FA9550-08-1-0265 from the Air Force Office of Scientic Research.

10. REFERENCES

[1] http://fke.utm.my/downloads/espresso/.

[2] Extensible access control markup language (XACML) version 2.0. *OASIS Standard*, 2005.

[3] A. Anderson. Evaluating xacml as a policy language. *Technical report, OASIS*, 2003.

[4] O. Arieli and A. Avron. The value of the four values. *Artificial Intelligence*, 102(1):97–141, 1998.

[5] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (EPAL). *Research report 3485, IBM Research*, 2003.

[6] M. Backes, M. Duermuth, and R. Steinwandt. An algebra for composing enterprise privacy policies. In *Proceedings of 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 33–52. Springer, September 2004.

[7] P. Bonatti, S. D. C. D. Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, 2002.

[8] G. Bruns, D. S. Dantas, and M. Huth. A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 5th ACM Workshop on Formal Methods in Security Engineering (FMSE)*, 2007.

[9] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 196–205, 2005.

[10] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient datastructure for matrix representation. *Formal Methods in System Design*, 10(2-3):149–169, 1997.

[11] O. Grumberg, S. Livne, and S. Markovitch. Learning to order bdd variables in verification. *Journal of Artificial Intelligence Research*, 18:83–116, 2003.

[12] J. Halpern and V. Weissman. Using first-order logic to reason about policies. *In Proceedings of the Computer Security Foundations Workshop (CSFW'03)*, 2003.

[13] R. Jagadeesan, W. Marrero, C. Pitcher, and V. Saraswat. Timed constraint programming: a declarative approach to usage control. In *Proc. of the 7th ACM SIGPLAN international conference on Principles and practice of declarative programming (PPDP)*, pages 164–175, 2005.

[14] N. Martin. The sheffer functions of 3-valued logic. *The Journal of Symbolic Logic*, 19(1):45–51, 1954.

[15] P. Mazzoleni, E. Bertino, and B. Crispo. XACML policy integration algorithms. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 223–232, 2006.

[16] P. McDaniel and A. Prakash. Methods and limitations of security policy reconciliation. *ACM Transactions on Information and System Security (TISSEC)*, 9(3):259 – 291, 2006.

[17] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo. An algebra for fine-grained integration of xacml policies. *Technical report, CERIAS*, 2008.

[18] G. Rousseau. Completeness in finite algebras with a single operation. *Proceedings of the American Mathematical Society*, 18(6):1009–1013, 1966.

[19] F. B. Schneider. Enforceable security policies. *ACM Transanction of Information System and Security (TISSEC)*, 3(1):30–50, 2000.

[20] R. Wheeler. Complete connectives for 3-valued propositional calculus. *Proceedings of London Mathematical Society*, 3(16):167–191, 1966.

[21] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):286–325, 2003.