

# An Algebra for Fine-Grained Integration of XACML Policies

Prathima Rao<sup>†</sup> Dan Lin<sup>†</sup> Elisa Bertino<sup>†</sup> Ninghui Li<sup>†</sup> Jorge Lobo<sup>‡</sup>

<sup>†</sup>Department of Computer Science  
Purdue University, USA  
{*prao,lindan,bertino,ninghui*}@cs.purdue.edu

<sup>‡</sup>IBM T.J. Watson Research Center  
USA  
*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 is able to support the specification of a large variety of integration constraints. To assess the expressive power of our algebra, we prove its completeness and minimality. 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.

## 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 combine 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 three-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. 3-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

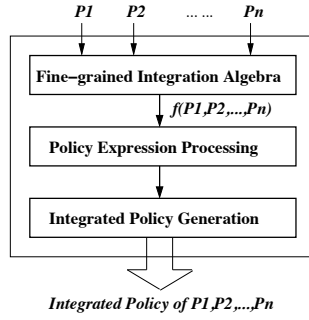


Figure 1: Policy integration

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. [4]. 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 al. [15] 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 they related their languages 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  $P_1, P_2, \dots, P_n$ , one is able to specify the integration requirements for these input policies through a FIA expression, denoted as  $f(P_1, P_2, \dots, P_n)$  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) [9] for representing policies and generating the integrated policies in XACML syntax. The novel contributions of this paper can be summarized as follows:

- We propose a fine-grained integration algebra for language independent 3-valued policies. This is the first time that such an algebra has been proposed.
- 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.

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

## 2 Preliminaries

### 2.1 An Overview of XACML

XACML [1] is the OASIS standard language for the specification of access control policies. XACML policies include three main components: a *Target*, a *Rule* set and a *Rule combining algorithm*. 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 “NotApplicable” and the effect will be ignored. The *Rule combining algorithm* is used to resolve conflicts among applicable rules with different effects.

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, regulating access to the company’s customer information.  $P_1$  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 the customer information in the time interval [8am, 6pm].  $P_1.Rul_{12}$  states that any other staff is not allowed to read the customer information.  $P_2$  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 the customer information in the time interval [8am, 8pm], and  $P_2.Rul_{22}$  states that the staff cannot update the customer information. The two policies in XACML syntax can be found in Appendix. 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.

### 2.2 Policy Semantics

Before we introduce our algebra we need to find a suitable definition of policy semantics. Even though different approaches to the definition of such semantics are possible [4, 15], we propose a simple yet powerful notion of semantics according to which the semantics of 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 our work, we assume the existence of a finite set  $A$  of names. Each attribute, characterizing a subject or an object or the environment, has a name  $a$  in  $A$ , and a domain, denoted by  $dom(a)$ , of possible

values. The following two definitions introduce the notion of access request (request, for short) and policy semantics.

**Definition 1** Let  $a_1, a_2, \dots, a_k$  be attribute names, and let  $v_i \in \text{dom}(a_i)$  ( $1 \leq i \leq k$ ).  $r \equiv \{(a_1, v_1), (a_2, v_2), \dots, (a_k, v_k)\}$  is a request.

**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 customer information at 10am. According to Definition 1, such request can be expressed as  $r \equiv \{(\text{role}, \text{manager}), (\text{act}, \text{read}), (\text{time}, \text{10am})\}$ .

**Definition 2** Let  $P$  be a 3-valued access control policy. We define the semantics of  $P$  as a 2-tuple  $\langle R_Y^P, R_N^P \rangle$ , where  $R_Y^P$  and  $R_N^P$  is the set of requests that are permitted and denied by  $P$  respectively, and  $R_Y^P \cap R_N^P = \emptyset$ .

Note that a policy  $P$  is not applicable to requests not in  $R_Y^P \cup R_N^P$ .  $P$  can be viewed as a function mapping each request to a value in  $\{Y, N, NA\}$ . Also, our approach to formulating the policy semantics is independent of the language in which access control policies are expressed. Therefore, our approach can be applied to languages other than XACML.

### 3 A Fine-grained Integration Algebra

The Fine-grained Integration Algebra (FIA) is given by  $\langle \Sigma, P_Y, P_N, +, \&, -, \Pi_{dc} \rangle$ , where  $\Sigma$  is a vocabulary of attribute names and their domains,  $P_Y$  and  $P_N$  are two policy constants,  $+$  and  $\&$  are two binary operators, and  $-$  and  $\Pi_{dc}$  are two unary operators.

#### 3.1 Policy Constants and Operators in FIA

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

**Permit policy ( $P_Y$ )**.  $P_Y$  is a policy constant that permits everything.

**Deny policy ( $P_N$ )**.  $P_N$  is a policy constant that denies everything.

**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 is not permitted by the other are denied. More precisely:

$$P_I = P_1 + P_2 \iff R_Y^{P_I} = R_Y^{P_1} \cup R_Y^{P_2} \quad \wedge \quad R_N^{P_I} = (R_N^{P_1} \setminus R_Y^{P_2}) \cup (R_N^{P_2} \setminus R_Y^{P_1})$$

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  $r$ .

$P_1 \backslash P_2$	Y	N	NA
Y	Y	Y	Y
N	Y	N	N
NA	Y	N	NA

$P_1 \backslash P_2$	Y	N	NA
Y	Y	NA	NA
N	NA	N	NA
NA	NA	NA	NA

$P_1 \backslash P_2$	Y	N	NA
Y	NA	NA	Y
N	NA	NA	N
NA	NA	NA	NA

$P_1 \backslash P_2$	Y	N	NA
Y	Y	Y	Y
N	N	N	N
NA	Y	N	NA

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

One partial order on the set  $\{Y, N, NA\}$  is the *information order*:  $Y > NA$ ,  $N > NA$ , as both  $Y$  and  $N$  provide more information about a request than  $NA$ . The  $+$  operator can be viewed as taking maximum on

the strict order  $Y > N > NA$ , which can be obtained by using the information order and preferring  $Y$  to  $N$ .

**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,

$$P_I = P_1 \& P_2 \iff R_Y^{P_I} = R_Y^{P_1} \cap R_Y^{P_2} \quad \wedge \quad R_N^{P_I} = R_N^{P_1} \cap R_N^{P_2}$$

The intersection operator can be viewed as taking minimum on the information order. The integrated policy makes a decision only when the two policies agree.

**Negation ( $\neg$ )**. 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:

$$P_I = \neg P \iff R_Y^{P_I} = R_N^P \quad \wedge \quad R_N^{P_I} = R_Y^P$$

**Domain projection ( $\Pi_{dc}$ )** The domain projection operator takes a parameter, the domain constraint  $dc$ , and restricts the policy only to the set of requests identified by  $dc$ .

**Definition 3** A domain constraint  $dc$  takes the form  $\{(a_1, range_1), (a_2, range_2), \dots, (a_k, range_k)\}^1$ , where  $a_1, a_2, \dots, a_k$  are attribute names, and  $range_i (1 \leq i \leq k)$  are sets of values from the vocabulary  $\Sigma$ . Given a request  $r = \{(a_{r_1}, v_{r_1}), \dots, (a_{r_m}, v_{r_m})\}$ . We say  $r$  satisfies  $dc$  if the following condition holds: for each  $(a_{r_j}, v_{r_j}) \in r (1 \leq j \leq m)$ , if there exists  $(a_{r_j}, range_i) \in dc$ , then  $v_{r_j} \in range_i$ .

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

$$P_I = \Pi_{dc}(P) \iff R_Y^{P_I} = \{r \mid r \in R_Y^P \text{ and } r \text{ satisfies } dc\}, R_N^{P_I} = \{r \mid r \in R_N^P \text{ and } r \text{ satisfies } dc\}$$

## 3.2 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.

In FIA expressions, the binary operators are viewed as left associative and unary operators are right associative. The precedence are  $\neg$  and  $\Pi_{dc}$  together have the highest precedence, following by  $\&$ , and then by  $+$ . For example,  $P_1 + \Pi_{dc}P_2 + \neg P_3 \& P_4$  is interpreted as  $((P_1 + (\Pi_{dc}P_2)) + ((\neg P_3) \& P_4))$ .

**Theorem 1** FIA has the following algebraic properties.

- **Commutativity:**  $P_1 + P_2 = P_2 + P_1$ ;  $P_1 \& P_2 = P_2 \& P_1$ ;
  - **Associativity:**  $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$ ;  $(P_1 \& P_2) \& P_3 = P_1 \& (P_2 \& P_3)$ ;
  - **Adsorption:**  $P_1 + (P_1 \& P_2) = P_1$ ;  $P_1 \& (P_1 + P_2) = P_1$ ;
  - **Distributivity:**  $P_1 + (P_2 \& P_3) = (P_1 + P_2) \& (P_1 + P_3)$ ;  $P_1 \& (P_2 + P_3) = (P_1 \& P_2) + (P_1 \& P_3)$ ;
- $$\Pi_{dc}(P_1 + P_2) = (\Pi_{dc}P_1) + (\Pi_{dc}P_2); \quad \Pi_{dc}(P_1 \& P_2) = (\Pi_{dc}P_1) \& (\Pi_{dc}P_2)$$

<sup>1</sup>In case of an ordered domain, these sets can be represented by ranges.

- **Complements:**  $P_Y = \neg P_N$ ;  $P_N = \neg P_Y$ ;
- **Idempotence:**  $P_1 + P_1 = P_1$ ;  $P_1 \& P_1 = P_1$ ;
- **Boundedness:**  $P_1 + P_Y = P_Y$ ;
- **Involution:**  $\neg(\neg P_1) = P_1$ .

### 3.3 Derived Operators

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

**Not-applicable policy ( $P_{NA}$ )** .  $P_{NA}$  is a policy constant that is not applicable for every request. It is defined as  $P_{NA} = P_Y \& P_N$ .

**Effect projection ( $\Pi_Y$  and  $\Pi_N$ )** .  $\Pi_Y(P)$  restricts the policy  $P$  to the requests allowed by it. It is defined as:  $\Pi_Y(P) = P \& P_Y$ . Similarly,  $\Pi_N(P)$  restricts the policy  $P$  to the requests denied by it; it is defined as  $\Pi_N(P) = P \& P_N$ . We are overloading  $\Pi$  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:

$$P_1 - P_2 = (P_Y \& (\neg(\neg P_1 + P_2 + \neg P_2))) + (P_N \& (P_1 + P_2 + \neg P_2)).$$

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  $P_Y \& (\neg(\neg P_1 + P_2 + \neg P_2))$  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,  $P_N \& (P_1 + P_2 + \neg P_2)$  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.

## 4 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 complete in that any possible policy integration requirements can be expressed using a FIA expression.

### 4.1 Expressing XACML Policy-Combining Algorithms in FIA

In XACML there are four 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 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  $P_1, P_2, \dots, P_n$  under this policy-combining algorithm can be expressed as  $P_1 + P_2 + \dots + P_n$  .

**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  $\neg((\neg P_1) + (\neg P_2) + \dots + (\neg P_n))$ .

**First-one-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  $P_1, P_2, \dots, P_n$ , the expression is  $P_1 \triangleright P_2 \triangleright \dots \triangleright P_n$ .

**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  $P_1, \dots, P_n$  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 :  $(P_1 - P_2 - P_3 - \dots - P_n) + (P_2 - P_1 - P_3 - \dots - P_n) + \dots + (P_n - P_1 - P_2 - \dots - P_{n-1})$ .

## 4.2 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-one-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 compared to 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  $P \triangleright P_Y$ . A closed policy of  $P$  denies everything that is not explicitly permitted, which can be represented as  $P \triangleright P_N$ .

Our algebra can also express the policy jump, 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 and fourth subexpressions deal with the case that  $P_1$  answers  $N$ .

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.

### 4.3 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 be combined to 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  $3^9 = 19683$ . We show that each such matrix can be expressed using  $\langle P_N, P_Y, +, \&, \neg \rangle$ . Second, when  $n$  ( $n > 2$ ) policies are combined, policy combination can be expressed using a  $n$ -dimensional matrix. We also show that each such  $n$ -dimensional matrix can be expressed using  $\langle P_N, P_Y, +, \&, \neg \rangle$ . Finally, a fine-grained integration may use different policy combination matrices for different requests. We show that this can be handled by using the operator  $\Pi_{dc}$  in addition to  $\langle P_N, P_Y, +, \&, \neg \rangle$ .

**Theorem 2** (*Binary completeness*). *Given any policy combination matrix  $M$ , let  $M(P_1, P_2)$  denote the result of combining two policies  $P_1$  and  $P_2$  using  $M$ . There exists a FIA expression  $f_I(P_1, P_2)$  that is equivalent to  $M(P_1, P_2)$ . That is,  $f_I(P_1, P_2) = M(P_1, P_2)$  for any two policies  $P_1$  and  $P_2$ .*

**Proof.** When all entries in  $M$  are  $NA$ ,  $f_I(P_1, P_2) = P_Y \& P_N$ . For the case that  $M$  has at least one entry that is not  $NA$ , we use the divide-and-conquer methodology. We transform the problem of finding an expression for  $M$  into finding sub-expressions for each entry in  $M$ . We number the cells in  $M$  from 1 to 9, by starting from the top-left cell and going right first. We then consider nine *simple matrices*  $SM_1, SM_2, \dots, SM_9$  (see Figure 2). In each simple matrix, at most one entry is not  $NA$ .

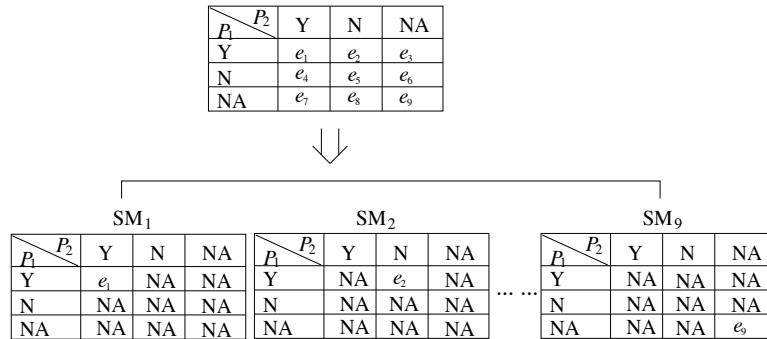


Figure 2: Policy combination matrix transformation

If no cell in  $M$  is  $NA$ , then let  $f_i(P_1, P_2)$  denote the FIA expression corresponding to the nine simple matrices.  $f_I(P_1, P_2)$  is thus the addition of the expression corresponding to each simple matrix, that is:  $f_I(P_1, P_2) = f_1(P_1, P_2) + f_2(P_1, P_2) + \dots + f_9(P_1, P_2)$ . If  $M$  contains  $NA$  somewhere in the nine cells, then we only need to add up the expressions corresponding to the cells that are not  $NA$ . For example, if cells 1 and 4 are not  $NA$ , then  $f_I(P_1, P_2) = f_1(P_1, P_2) + f_4(P_1, P_2)$ .

Now our task is to find proper expressions for each  $f_i(P_1, P_2)$ . Let  $f'_i(P_1, P_2)$  denote the expression that is equivalent to a policy matrix where the  $i$ 'th cell is  $Y$ , and all other cells are either  $N$  or  $NA$ . Then when  $e_i = Y$ , we have  $f_i(P_1, P_2) = f'_i(P_1, P_2) \& P_Y$  and when  $e_i = N$ , we have  $f_i(P_1, P_2) = \neg f'_i(P_1, P_2) \& P_N$ . Expressions  $f'_i(P_1, P_2)$ ,  $1 \leq i \leq 9$  are shown in Table 2.



To summarize, FIA can express any policy combination matrix. ■

Theorem 2 has proved the completeness when considering two policies. We now extend our completeness result to any number of policies. The proof of Theorem 3 is shown in appendix.

**Theorem 3 (General completeness)** Given  $n$  ( $n > 2$ ) policies  $P_1, P_2, \dots, P_n$ , let  $M^*(P_1, P_2, \dots, P_n)$  be a  $n$ -dimensional policy combination matrix which denotes the combination result of the  $n$  policies. There exists a FIA expression  $f_I(P_1, P_2, \dots, P_n)$  that is equivalent to  $M^*(P_1, P_2, \dots, P_n)$ .

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** Let  $\mathcal{R}_\Sigma$  denote the set of all requests in the vocabulary  $\Sigma$ . A fine-grained integration specification is given by  $[(R_1, M_1^*), (R_2, M_2^*), \dots, (R_k, M_k^*)]$ , where  $R_1, R_2, \dots, R_k$  form a partition of  $\mathcal{R}_\Sigma$ , i.e.,  $\mathcal{R}_\Sigma = R_1 \cup R_2 \cup \dots \cup R_k$  ( $k \geq 1$ ) and  $R_i \cap R_j = \emptyset$  when  $i \neq j$ , and each  $M_i^*(P_1, \dots, P_n)$  ( $1 \leq i \leq k$ ) 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 4** Given a fine-grained integration specification  $[(R_1, M_1^*), (R_2, M_2^*), \dots, (R_k, M_k^*)]$ , if for each  $R_i$ , there exists  $dc_{i,1}, \dots, dc_{i,m_i}$  such that  $R_i = R(dc_{i,1}) \cup \dots \cup R(dc_{i,m_i})$  (where  $R(dc_{i,j})$  denotes the set of requests satisfying  $dc_{i,j}$ ), then there exists a FIA expression  $f_I(P_1, P_2, \dots, P_n)$  that achieves the integration requirement.

**Proof.** We first use the domain projection operator  $\Pi_{dc}$  to project each policy according to  $dc_{1,1}, \dots, dc_{k,m_k}$ . For requests in each  $R(dc_{i,j})$ , there is one fixed  $M_i^*$ . By Theorem 3, there is a FIA expression (denoted as  $f_{i,j}$ ) for integrating policies  $\Pi_{dc_{i,j}}(P_1), \dots, \Pi_{dc_{i,j}}(P_n)$  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 finite number of domain constraints. This condition is always satisfied when the set of all requests is finite. However, when the set is infinite, it may not hold. For example, if one requires all requests that satisfy a predicate uses one combination matrix and all other requests use another matrix, but the predicate is uncomputable, then we cannot generate a FIA expression to achieve this. However, we believe that requirements that are likely to occur in practice will satisfy the condition that each  $R_i$  is expressible in finite number of domain constraints.

#### 4.4 Minimal Set of Operators

Recall that FIA has  $\{P_Y, P_N, +, \&, \neg, \Pi_{dc}\}$ . The operator  $\Pi_{dc}$  is needed to deal with fine-grained integration. Operators  $\{P_Y, P_N, +, \&, \neg\}$  are complete in the sense that any policy combination matrix can be expressed using them. A natural question is among the set  $\Theta = \{P_N, P_Y, P_{NA}, +, \&, \neg, \Pi_Y, \Pi_N, -, \triangleright\}$ ,

$P_1 \backslash P_2$	Y	N	NA
Y	$f'_1 = P_1 \& P_2$	$f'_2 = P_1 \& (\neg P_2)$	$f'_3 = \neg(\neg P_1 + P_2 + (\neg P_2))$
N	$f'_4 = (\neg P_1) \& P_2$	$f'_5 = (\neg P_1) \& (\neg P_2)$	$f'_6 = \neg(P_1 + P_2 + (\neg P_2))$
NA	$f'_7 = \neg(\neg P_2 + P_1 + (\neg P_1))$	$f'_8 = \neg(P_2 + P_1 + (\neg P_1))$	$f'_9 = P_Y - (P_1 + P_2)$

Table 2: Expressions for  $f'_i(P_1, P_2)$

what subsets are *minimally complete*. We say a subset of  $\Theta$  is minimally complete, if operators in the subset are sufficient for defining all other operators in  $\Theta$ , and any smaller subset cannot define all operators in  $\Theta$ . The following theorem answers this question. The only redundancy in  $\{P_Y, P_N, +, \&, \neg\}$  is that only one of  $P_Y$  and  $P_N$  is needed.

**Theorem 5** *Among the 10 operators in  $\Theta$ , there are 12 minimally complete subsets. They are the 12 elements in the cartesian product  $\{\neg\} \times \{P_Y, P_N\} \times \{\Pi_Y, \Pi_N, \&\} \times \{+, \triangleright\}$ .*

Details of the proof are in the appendix. Here, we summarize the key lemmas that lead to the result:

- The policy constant  $P_N$  cannot be expressed using  $\Theta \setminus \{P_Y, P_N\}$ . When given  $\neg$ ,  $P_Y$  and  $P_N$  can be derived from each other.
- The unary operator  $\neg$  cannot be expressed using  $\Theta \setminus \{\neg\}$ .
- The binary operator  $\&$  cannot be expressed using  $\Theta \setminus \{\&, \Pi_Y, \Pi_N\}$ . Given  $\neg$ ,  $\Pi_Y$  and  $\Pi_N$  can be expressed from each other.  $\Pi_Y$  can be expressed using  $\{\&, P_Y\}$ . And  $\&$  can be expressed using  $\{P_N, +, \neg, \Pi_N\}$ .
- The binary operator  $+$  cannot be expressed using  $\Theta \setminus \{+, \triangleright\}$ . However,  $+$  can be expressed using  $\{P_N, \neg, \&, \triangleright\}$  or  $\{P_N, \neg, \Pi_N, \triangleright\}$ , and  $\triangleright$  can be expressed using  $\{P_N, +, \neg, \&\}$ .

In summary, among the 10 operators in  $\Theta$ , for completeness, we must have  $\neg$ , one in  $\{P_Y, P_N\}$ , one in  $\{\Pi_Y, \Pi_N, \&\}$ , and one in  $\{+, \triangleright\}$ . There are 12 combinations. It is not difficult to verify that every such combination is in fact complete. For example, once we have  $\{\neg, P_N, \Pi_N\}$ , adding  $+$  allows us to derive  $\&$ , and then derive  $\triangleright$ , adding  $\triangleright$  allows us to derive  $+$  and then  $\&$ . There are thus 12 minimally complete subsets in  $\Theta$ .

## 5 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) [9], and then perform operations on the underlying MTBDD structures to generate the integrated policy. We have chosen a MTBDD based implementation of the proposed algebra because (i) MTBDDs have proven to be a simple and efficient representation for XACML policies [8] 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  $f(P_1, P_2, \dots, P_n)$ , 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.

### 5.1 Policy Representation

Recall from section 2 that we characterize a policy  $P$  as a 2-tuple  $\langle R_Y^P, R_N^P \rangle$ , where  $R_Y^P$  is the set of requests permitted by the policy, and  $R_N^P$  is the set of requests denied by the policy. Alternatively, we can define  $P$  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 [2]. 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 inequality constraints,  $c_1 \triangleleft a \triangleright c_2$ , where  $a$  is an attribute name,  $c_1$  and  $c_2$  are constants, and  $\triangleleft, \triangleright \in \{<, \leq, >, \geq\}$ .

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

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

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.

**Definition 6** Two atomic Boolean expressions  $a_i \triangleright_i c_i$  and  $a_j \triangleright_j c_j$  are equal iff  $a_i = a_j \wedge \triangleright_i = \triangleright_j \wedge c_i = c_j$ . Two atomic Boolean expressions  $c_{i1} \triangleleft_i a_i \triangleright_i c_{i2}$  and  $c_{j1} \triangleleft_j a_j \triangleright_j c_{j2}$  are equal iff  $a_i = a_j \wedge \triangleleft_i = \triangleleft_j \wedge \triangleright_i = \triangleright_j \wedge c_{i1} = c_{j1} \wedge c_{i2} = c_{j2}$ .

**Example 5** The Boolean encoding for policy  $P_1$  is given in Table 3.

$x_0$	$x_1$	$x_2$	$x_3$	$x_4$
$role = manager$	$role = staff$	$act = read$	$act = update$	$8am \leq time \leq 6pm$

Table 3: 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,  $\vec{x} = x_0, x_1, \dots, x_n$ , 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.

**Example 6** After Boolean encoding, the policy  $P_1$  is transformed into the function :

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

The transformed policy function can now be represented as a MTBDD. A MTBDD provides a compact representation of functions of the form  $f : B^n \mapsto \mathbb{R}$ , which maps bit vectors over a set of variables ( $B^n$ ) 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

[10]. For examples discussed in this paper, we use the variable ordering  $x_0 \prec x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$ . The MTBDD of the policy  $P_1$  is shown in Figure 3, 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 sequences 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)  $d_i'^-, d_i'^+ \in D$ ,  $D = \{d_1^-, d_1^+, \dots, d_n^-, d_n^+\}$ ; (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 7** By introducing another atomic Boolean expression  $6pm \leq time \leq 8pm$ , i.e.  $x_5$ , the transformed function for policy  $P_2$  is :

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

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 3.

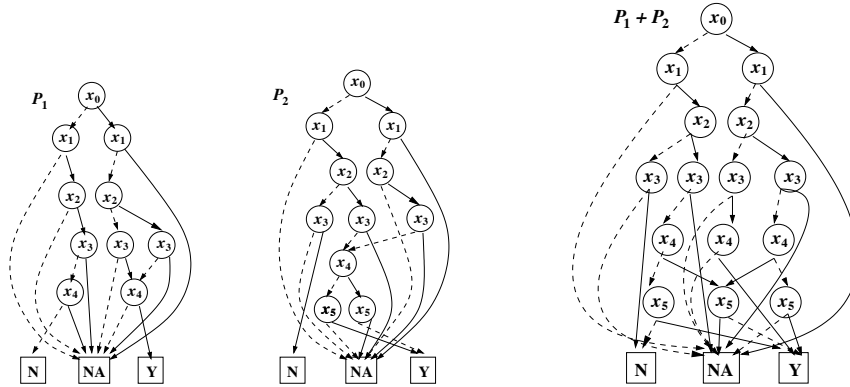


Figure 3: MTBDDs of  $P_1$ ,  $P_2$  and  $P_1 + P_2$

## 5.2 Construction of Integrated Policy MTBDD

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

---

**Procedure Apply**( $Node_1, Node_2, OP$ )

**Input** :  $Node_1, Node_2$  are MTBDD nodes,  $OP$  is a policy operation

1. initiate  $Node_I$  //  $Node_I$  is the combination result
  2. **if**  $Node_1$  and  $Node_2$  are terminals **then**
  3.      $Node_I \leftarrow (Node_1 OP Node_2, null, null)$
  4. **else**
  5.     **if**  $Node_1.var = Node_2.var$  **then**
  6.          $Node_I.var \leftarrow Node_1.var$
  7.          $Node_I.left \leftarrow Apply(Node_1.left, Node_2.left, OP)$
  8.          $Node_I.right \leftarrow Apply(Node_1.right, Node_2.right, OP)$
  9.     **if**  $Node_1.var$  precedes  $Node_2.var$  **then**
  10.          $Node_I.var \leftarrow Node_1.var$
  11.          $Node_I.left \leftarrow Apply(Node_1.left, Node_2, OP)$
  12.          $Node_I.right \leftarrow Apply(Node_1.right, Node_2, OP)$
  13.     **if**  $Node_2.var$  precedes  $Node_1.var$  **then**
  14.          $Node_I.var \leftarrow Node_2.var$
  15.          $Node_I.left \leftarrow Apply(Node_2.left, Node_1, OP)$
  16.          $Node_I.right \leftarrow Apply(Node_2.right, Node_1, OP)$
  17. **return**  $Node_I$
- 

Figure 4: Description of the Apply procedure

Operations on policies can be expressed as operations on the corresponding policy MTBDDs. The **negation**  $\neg$  operation can be performed by interchanging the terminal  $Y$  and  $N$ . Many efficient operations have been defined and implemented for MTBDDs [9]. In particular, we use the **Apply** operation defined on MTBDDs to perform the FIA binary operations  $\{+, -, \&, \triangleright\}$ . We introduce a new MTBDD operation called **Projection** to perform the domain projection operation  $\Pi_{dc}$  defined in FIA.

The **Apply** operation combines two MTBDDs by a specified binary arithmetic operation. A high level description of the **Apply** operation is shown in Figure 4, where *var*, *left*, *right* refer to the variable, left child and right child of a MTBDD node, respectively. 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.

**Example 8** 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 the effect projection operation is the following. For  $\Pi_Y$ , those paths in  $T^P$  that lead to  $N$  are redirected to the terminal  $NA$ . Similarly, for  $\Pi_N$ , 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. If the projection operation contains both types of constraints, we apply one constraint first and then apply the other by using previous two algorithms.

Thus, given any arbitrary FIA expression  $f(P_1, P_2, \dots, P_n)$ , 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.

**Example 9** 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 :

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

### 5.3 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 into 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*.

Next, we replace Boolean variables in the path-expressions with the corresponding atomic Boolean expressions by using the mapping built 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.

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.

Note that the number of rules generated for the integrated policy depends on the number of paths leading to  $Y(N)$  which may be exponential in the number of nodes in the MTBDD. To address this issue, we propose to leverage existing BDD path-minimization techniques [6, 7] that minimize the number of paths leading to one-terminal in a BDD along with logic minimization techniques [5]. Our preliminary results by using Espresso [5] show a significant reduction (from 75% to 99%) in the number of rules generated.

**Example 10** Consider policies  $P_1$  and  $P_2$  in Example 1. Figure 5 shows an example of  $P_1 + P_2$ . The left part of the figure shows the paths leading to the  $N$  terminal and the corresponding Boolean expressions which

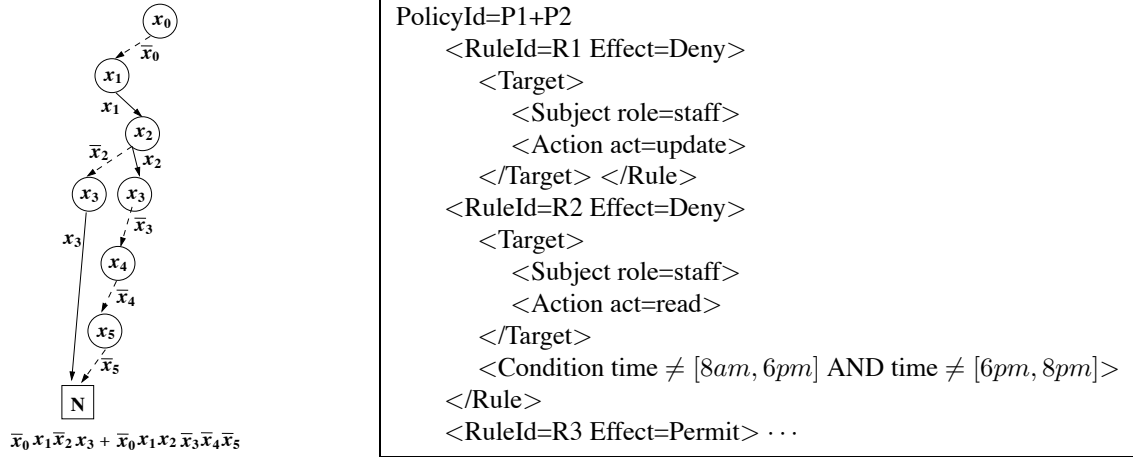


Figure 5: The integrated XACML policy representing  $P_1 + P_2$

can be 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 right part of the figure shows the corresponding deny rules for the integrated policy.

## 6 Related Work

In the literature, many efforts have been devoted into policy composition [3, 4, 12, 14, 15]. Few approaches have been proposed for dealing with the fine-grained integration of XACML policies. Approaches most closest to ours are by Mazzoleni et al. [11], Bonatti et al. [4], Wijesekera et al. [15] and Backes et al. [3].

Mazzoleni et al. [11] 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 this work are very limited and do not support fine-grained integration requirements. Bonatti et al. [4] have proposed an algebra for combining authorization specifications that may originate 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). They propose an implementation of their algebra based on logic programming and partial evaluation techniques. Unlike our work which can handle 3-valued policies, their model does not explicitly support negative authorizations. Also, our implementation is based on representations used in model checking techniques which have been proven to be very efficient. In addition, we also provide mechanisms to synthesize a concrete integrated policy resulting from the evaluation of a policy expression. Wijesekera et al. [15] have proposed a propositional algebra for access control. They model policies as nondeterministic transformers of permission set assignments to subjects and interpret operations on policies as set-theoretic operations on the transformers. Their work does not discuss an implementation for the algebra. Backes et al. [3] have proposed an algebra for combining enterprise privacy policies. They define conjunction, disjunction and scoping operations on 3-valued EPAL [13] policies. In contrast to our work, the *don't care* value is treated as a special value that can only be used by the default rulings of a policy and they do not have an implementation of their algebra.

## 7 Conclusion

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. Based on this algebra, we propose a framework for integration of XACML policies. We also discuss the generation of an actual XACML policy representing the integrated policy corresponding to a FIA policy expression.

## References

- [1] Extensible access control markup language (xacml) version 2.0. *OASIS Standard*, 2005.
- [2] A. Anderson. Evaluating xacml as a policy language. *Technical report, OASIS*, 2003.
- [3] Michael Backes, Markus Duermuth, and Rainer Steinwandt. An algebra for composing enterprise privacy policies. In *Proceedings of 9th European Symposium on Research in Computer Security (ES-ORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 33–52. Springer, September 2004.
- [4] P. Bonatti, S. D. C. D. Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISS)*, 5(1):1–35, 2002.
- [5] Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1984.
- [6] G. Fey and R. Drechsler. A hybrid approach combining symbolic and structural techniques for disjoint sop minimization. In *Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI), pages 54–60, 2003.*, 2003.
- [7] Görschwin Fey and Rolf Drechsler. Minimizing the number of paths in bdds. In *SBCCI '02: Proceedings of the 15th symposium on Integrated circuits and systems design*, page 359, Washington, DC, USA, 2002. IEEE Computer Society.
- [8] 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.
- [9] 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.
- [10] O. Grumberg, S. Livne, and S. Markovitch. Learning to order bdd variables in verification. *Journal of Artificial Intelligence Research*, 18:83–116, 2003.
- [11] 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.
- [12] 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.
- [13] G. Karjoth C. Powers M. Schunter P. Ashley, S. Hada. Enterprise privacy authorization language (epal). *Research report 3485, IBM Research*, 2003.



- [14] F. B. Schneider. Enforceable security policies. *ACM Transaction of Information System and Security (TISS)*, 3(1):30–50, 2000.
- [15] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and System Security (TISS)*, 6(2):286–325, 2003.

## Appendix

**Theorem 3** Given  $n$  ( $n \geq 2$ ) policies  $P_1, P_2, \dots, P_n$ , let  $M^*(P_1, P_2, \dots, P_n)$  be a  $n$ -dimensional policy combination matrix which denotes the combination result of the  $n$  policies. There exists a FIA expression  $f_I(P_1, P_2, \dots, P_n)$  that is equivalent to  $M^*(P_1, P_2, \dots, P_n)$ .

**Proof.** We prove this theorem by induction. The base case is when  $n = 2$ , which is true according to Theorem 2.

Assuming that when  $n = k - 1$  the corollary holds, we now consider the case when  $n = k$ . As shown in Table 4,  $M^*(P_1, \dots, P_k)$  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  $f^{k-1}(P_1, \dots, P_{k-1})$ .

$P_1, P_2, \dots, P_{k-1}$	$P_k$	$M^*$	$f^{k-1}(P_1, P_2, \dots, P_{k-1})$
$\ddots, Y, \dots, Y$	$\ddots$	$e_{1,1}$	$f_{1,1}^{k-1}(P_1, P_2, \dots, P_{k-1})$
$\ddots, NA, NA, \dots, NA$	$Y$	$e_{1,3^{k-1}}$	$f_{1,3^{k-1}}^{k-1}(P_1, P_2, \dots, P_{k-1})$
$Y, Y, \dots, Y$	$N$	$e_{2,1}$	$f_{2,1}^{k-1}(P_1, P_2, \dots, P_{k-1})$
$\ddots, NA, NA, \dots, NA$	$\ddots$	$e_{2,3^{k-1}}$	$f_{2,3^{k-1}}^{k-1}(P_1, P_2, \dots, P_{k-1})$
$Y, Y, \dots, Y$	$NA$	$e_{3,1}$	$f_{3,1}^{k-1}(P_1, P_2, \dots, P_{k-1})$
$\ddots, NA, NA, \dots, NA$	$\ddots$	$e_{3,3^{k-1}}$	$f_{3,3^{k-1}}^{k-1}(P_1, P_2, \dots, P_{k-1})$

Table 4:  $n$  Policies

Next, we extend  $f^{k-1}(P_1, \dots, P_{k-1})$  to  $f^k(P_1, \dots, P_k)$  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 5. Note that we do not need to consider the cell where  $e_{i,j}$  is  $NA$ .

$P_k$	$e_{i,j}$	$f_{i,j}^k$
$Y$	$Y$	$f_{i,j}^{k-1} \& (P_k \& P_Y)$
$Y$	$N$	$f_{i,j}^{k-1} \& [\neg(P_k \& P_Y)]$
$N$	$Y$	$f_{i,j}^{k-1} \& [\neg(P_k \& P_N)]$
$N$	$N$	$f_{i,j}^{k-1} \& (P_k \& P_N)$
$NA$	$Y$	$f_{i,j}^{k-1}$
$NA$	$N$	$f_{i,j}^{k-1}$

Table 5:  $n$  Policies

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

We have shown that the corollary holds for  $n = 2$ , and we have also shown that if the corollary holds for  $n = k - 1$  then it holds for  $n = k$ . We can therefore state that it holds for all  $n$ . ■

**Theorem 5** Among the 10 operators in  $\Theta$ , there are 12 minimally complete subsets. They are the 12 elements in the cartesian product  $\{\neg\} \times \{P_Y, P_N\} \times \{\Pi_Y, \Pi_N, \&\} \times \{+, \triangleright\}$ .

**Proof.**

- The policy constant  $P_N$  cannot be expressed using  $\Theta \setminus \{P_Y, P_N\}$ . When given  $\neg$ ,  $P_Y$  and  $P_N$  can be derived from each other:  $P_Y = \neg P_N$  and  $P_N = \neg P_Y$ .

We need to show that, there does not exist a policy expression using operators in  $\Theta \setminus \{P_Y, P_N\}$  that is equivalent to  $P_N$ . Consider the information ordering among the three values:  $Y > NA$  and  $N > NA$ . The key observation is that the operators in  $\Theta \setminus \{P_Y, P_N\}$  are all non-increasing in the information ordering.

Suppose, for the sake of contradiction, that a policy expression  $f(P_1, P_2, \dots, P_n)$  constructed from  $\Theta \setminus \{P_Y, P_N\}$  and policy variables  $P_1, \dots, P_n$  is equivalent to  $P_N$ . Then this must mean that no matter what actual policies are used to instantiate  $P_1, \dots, P_n$ , the result is  $P_N$ . Let  $e_0 = f(P_{NA}, P_{NA}, \dots, P_{NA}) = P_N$ . We now use a structural induction to show that  $e_0$  must give  $NA$  for every request; thus contradiction. For the base case, we have policy constant  $P_{NA}$ , this is true. For the unary operators, if  $e$  gives  $NA$  for a request, then  $\Pi_Y(e)$ ,  $\Pi_N(e)$ , and  $\neg(e)$  are also  $NA$ . For the binary operators  $+$ ,  $-$ ,  $\&$ , and  $\triangleright$ , if both operands are  $NA$  for a request, then the result is also  $NA$  for the request.

- The unary operator  $\neg$  cannot be expressed using  $\Theta \setminus \{\neg\}$ .

The key observation is that without  $\neg$ , one cannot switch  $Y$  and  $N$ .

Suppose, for the sake of contradiction, that  $e_0(P)$  is equivalent to  $\neg P$ . Let  $P$  be a policy that returns  $Y$  on  $q_1$  and  $N$  on  $q_2$ . Then  $e_0(P)$  must return  $N$  on  $q_1$  and  $Y$  on  $q_2$ , i.e., it must give  $(N, Y)$  on  $q_1$  and  $q_2$ . We use structural induction to show that the result  $e_0(P)$  gives for  $q_1$  and  $q_2$  must be among  $(Y, N)$ ,  $(Y, Y)$ ,  $(N, N)$ ,  $(NA, NA)$ ,  $(Y, NA)$ ,  $(NA, N)$ . That is, if the answer for  $q_1$  is  $N$ , then the answer for  $q_2$  must be  $N$ , and if the answer for  $q_2$  is  $Y$ , the answer for  $q_1$  must be  $Y$ . Hence contradiction. For the base case, this holds for  $P$  and the three constants  $P_Y, P_N, P_{NA}$ . One can verify that the six pairs are closed under  $\Pi_Y, \Pi_N, +, -, \&, \triangleright$ .

- The binary operator  $\&$  cannot be expressed using  $\Theta \setminus \{\&, \Pi_Y, \Pi_N\}$ . Given  $\neg$ ,  $\Pi_Y$  and  $\Pi_N$  can be expressed from each other:  $\Pi_Y(P) = P \& \neg P_N$ ,  $\Pi_N(P) = \neg \Pi_Y(\neg P)$ .  $\Pi_Y$  can be expressed using  $\{\&, P_Y\}$  by definition, and  $\&$  can be expressed using  $\{P_N, +, \neg, \Pi_N\}$ .

Assume, for the sake of contradiction, that  $e_0(P_1, P_2)$  is equivalent to  $P_1 \& P_2$ . Let  $P_1$  be a policy that returns  $(Y, Y)$  on  $q_1$  and  $q_2$ , and  $P_2$  returns  $(Y, N)$  on  $q_1$  and  $q_2$ . Then  $e_0(P_1, P_2)$  must return  $(Y, NA)$  on  $q_1$  and  $q_2$ . We show this is not possible. The key insight here is that without  $\&$ ,  $\Pi_Y, \Pi_N$ , one cannot get information asymmetry  $Y$  or  $N$  for one request and  $NA$  for another from symmetric policies.

We use a structural induction to show that the result  $e(P_1, P_2)$  gives for  $q_1$  and  $q_2$  must be among the following set:  $\{(Y, N), (Y, Y), (NA, NA), (N, Y), (N, N)\}$ . This holds for all  $P_1, P_2, P_Y, P_N, P_{NA}$ . One can verify that the set is closed under  $\neg, +, -, \triangleright$ .

Given  $\{P_N, +, \neg\}$ ,  $\&$  can be expressed using either  $\Pi_Y$  or  $\Pi_N$ :

$$P_1 \& P_2 = (P_1 \sqcap P_2) + \neg(\neg P_1 \sqcap \neg P_2),$$

where

$$P_1 \sqcap P_2 = \Pi_Y(P_1) - (\Pi_N(P_N + P_2)),$$

and  $-$  can be defined using  $\Pi_Y, \Pi_N$  as:

$$P_1 - P_2 = (\Pi_Y(\neg(\neg P_1 + P_2 + \neg P_2))) + (\Pi_N(P_1 + P_2 + \neg P_2)).$$

The effect of  $P_1 \sqcap P_2$  is to authorize any request that is authorized by both  $P_1$  and  $P_2$ , and to be  $NA$  for all other requests.

- The binary operator  $+$  cannot be expressed using  $\Theta \setminus \{+, \triangleright\}$ . However,  $+$  can be expressed using  $\{P_N, \neg, \&, \triangleright\}$  or  $\{P_N, \neg, \Pi_N, \triangleright\}$ , and  $\triangleright$  can be expressed using  $\{P_N, +, \neg, \&\}$ .

Suppose, for the sake of contradiction, that an expression  $e_0(P_1, P_2)$  constructed from  $P_1, P_2$  and  $\Theta \setminus \{+, \triangleright\}$  is equivalent to  $P_1 + P_2$ . Let  $P_1$  be a policy that returns  $(Y, NA)$  on  $q_1$  and  $q_2$ , and  $P_2$  be a policy that returns  $(NA, N)$  on  $q_1$  and  $q_2$ . Then  $e_0(P_1, P_2)$  must return  $(Y, N)$  on  $q_1$  and  $q_2$ .

We use a structural induction to show that the result  $e(P_1, P_2)$  gives for  $q_1$  and  $q_2$  must be among  $(Y, Y), (Y, NA), (N, N), (N, NA), (NA, Y), (NA, N), (NA, NA)$ . Hence contradiction. This is satisfied by  $P_1, P_2, P_Y, P_N, P_{NA}$ . One can verify that these seven values are closed under  $\neg, \Pi_Y, \Pi_N, -, \&$ .

The  $+$  operator can be expressed using  $\{P_N, \neg, \&, \triangleright\}$ :

$$P_1 + P_2 = (P_1 \& P_Y) \triangleright (P_2 \& P_Y) \triangleright (P_1 \& P_N) \triangleright (P_2 \& P_N).$$

Similarly, the  $+$  operator can be expressed using  $\{P_N, \neg, \Pi_N, \triangleright\}$ :

$$P_1 + P_2 = (\Pi_Y P_1) \triangleright (\Pi_Y P_2) \triangleright (\Pi_N P_1) \triangleright (\Pi_N P_2).$$

Recall that the operator  $\triangleright$  is defined using  $+$  and  $-$  as  $P_1 \triangleright P_2 = P_1 + (P_2 - P_1)$ , and  $P_1 - P_2 = (P_Y \& (\neg(\neg P_1 + P_2 + \neg P_2))) + (P_N \& (P_1 + P_2 + \neg P_2))$ . Thus,  $\triangleright$  can be expressed using  $\{P_N, -, \neg, \&\}$ .

In summary, among the 10 operators in  $\Theta$ , for completeness, we must have  $\neg$ , one in  $\{P_Y, P_N\}$ , one in  $\{\Pi_Y, \Pi_N, \&\}$ , and one in  $\{+, \triangleright\}$ . There are 12 combinations. It is not difficult to verify that every such combination is in fact complete. For example, once we have  $\{\neg, P_N, \Pi_N\}$ , adding  $+$  allows us to derive  $\&$ , and then derive  $\triangleright$ , adding  $\triangleright$  allows us to derive  $+$  and then  $\&$ . There are thus 12 minimally complete subsets in  $\Theta$ . ■

```

<xml version="1.0" encoding="UTF-8"?>
<Policy PolicyId= $P_1$ >
  <Rule RuleId="Rul $_{11}$ " Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="Name-Match">
            <AttributeValue DataType="#string">
              manager </AttributeValue>
            <SubjectAttributeDesignator AttributeId="role"
              DataType="#string" />
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="#string">
            read </AttributeValue>
          <ActionAttributeDesignator AttributeId="act"
            DataType="#string" />
        </ActionMatch>
      </Action>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="#string">
            update </AttributeValue>
          <ActionAttributeDesignator AttributeId="act"
            DataType="#string" />
        </ActionMatch>
      </Action>
    </Actions>
    <Condition>
      <Apply FunctionId="Time-In-Range">
        <Apply FunctionId="Time-one-and-only">
          <SubjectAttributeDesignator AttributeId="time"
            DataType="#time" />
        </Apply>
        <AttributeValue DataType="#time">
          08:00 </AttributeValue>
        <AttributeValue DataType="#time">
          18:00 </AttributeValue>
        </Apply>
      </Condition>
    </Target>
  </Rule>
  <Rule RuleId="Rul $_{12}$ " Effect="Deny">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="Name-Match">
            <AttributeValue DataType="#string">
              staff </AttributeValue>
            <SubjectAttributeDesignator AttributeId="role"
              DataType="#string" />
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="#string">
            read </AttributeValue>
          <ActionAttributeDesignator AttributeId="act"
            DataType="#string" />
        </ActionMatch>
      </Action>
    </Actions>
  </Rule>
</Policy>

```

Figure 6: Policy  $P_1$

```

<xml version="1.0" encoding="UTF-8">
<Policy PolicyId= $P_2$ >
  <Rule RuleId="Rul $_{21}$ " Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="Name-Match">
            <AttributeValue DataType="#string">
              manager </AttributeValue>
            <SubjectAttributeDesignator AttributeId="role"
              DataType="#string" />
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="#string">
            read </AttributeValue>
          <ActionAttributeDesignator AttributeId="act"
            DataType="#string" />
        </ActionMatch>
      </Action>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="#string">
            update </AttributeValue>
          <ActionAttributeDesignator AttributeId="act"
            DataType="#string" />
        </ActionMatch>
      </Action>
    </Actions>
    <Condition>
      <Apply FunctionId="Time-In-Range">
        <Apply FunctionId="Time-one-and-only">
          <SubjectAttributeDesignator AttributeId="time"
            DataType="#time" />
        </Apply>
        <AttributeValue DataType="#time">
          08:00 </AttributeValue>
        <AttributeValue DataType="#time">
          20:00 </AttributeValue>
        </Apply>
      </Condition>
    </Target>
  </Rule>
  <Rule RuleId="Rul $_{12}$ " Effect="Deny">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="Name-Match">
            <AttributeValue DataType="#string">
              staff </AttributeValue>
            <SubjectAttributeDesignator AttributeId="role"
              DataType="#string" />
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
    <Actions>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="#string">
            update </AttributeValue>
          <ActionAttributeDesignator AttributeId="act"
            DataType="#string" />
        </ActionMatch>
      </Action>
    </Actions>
  </Rule>
</Policy>

```

Figure 7: Policy  $P_2$