The Soundness and Completeness of Margrave with Respect to a Subset of XACML

Michael Matthew Greenberg, Casey Marks,
Leo Alexander Meyerovich, and
Michael Carl Tschantz

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-05-05
April 2005

The Soundness and Completeness of Margrave with Respect to a Subset of XACML

Michael Matthew Greenberg Casey Marks Leo Alexander Meyerovich
Michael Carl Tschantz

February 13, 2005

Abstract

Provides a natural semantics for a subset of XACML, a language for the specification of access-control policies. Presents the key algorithm of Margrave, a tool for analyzing XACML policies. Proves the soundness and completeness of the algorithm with respect to the subset of XACML.

1 Introduction

Multi-user and web applications necessitate controlling the actions a user may perform on a resource. Generally, some policy governs what each user is allowed to access. Applying *ad hoc* checks spread through out the code to enforce such policies makes the policy hard to understand or modify. Thus, such informal policies should be formalized into a central access-control policy, which is enforced through out the program. To allow for these policies to be shared across applications and for APIs to aid in enforcement, standardized languages are arising for the specification of access-control policies.

XACML is a language for the specification of such access-control policies [2]. XACML is an OASIS standard with backing from Sun Microsystems and others. In this paper we are concerned with the subset of XACML that is sufficient for role-based access control. In Section 3, we provide a natural semantics for this subset of XACML.

After specifying a policy in a language like XACML, the policy administrator should verify that the specification matches the intent of the policy. The policy verification tool Margrave aids in this task [1]. Margrave analyzes XACML policies belonging to the subset of XACML modeled in Section 3. Margrave uses an internal representation of an XACML policy called a multi-terminal binary decision diagram (MTBDD). In Section 4, we formalize the algorithm that produces a MTBDD from an XACML file. In Section 5, we prove the soundness and completeness of the results of this algorithm with respect to the subset of XACML presented in Section 3.

2 Syntax

For readability we will not use an XML style syntax for our subset of XACML, but rather a scheme-like syntax.

XACML has two syntaxes: one for specifications of policies and one for requests. Since the former has a key word "Policy", we will call it the spec syntax. The latter is called the request syntax.

2.1 XACML Spec Syntax

The start nonterminal is S.

S	::=	C
C	::=	Ps | Pol
Ps	::=	`(PolicySet` Ca T C*`)`
Ca	::=	`First-Applicable` | `Deny-Overrides` | `Permit-Overrides`
T	::=	`(` (Sub) (Res) (Act)`)`
Sub	::=	`Any` | Allow$^+$
Res	::=	`Any` | Allow$^+$
Act	::=	`Any` | Allow$^+$
Allow	::=	`(`AVC$^+$`)`
AVC	::=	`(`id val`)`
Pol	::=	`(Policy` Ca T R*`)`
R	::=	`(Rule` T Effect`)`
Effect	::=	`Permit` | `Deny`

The elements of the syntax catagory R are called "Rules"; Ps, "PolicySets"; Pol, "Policies"; and T, "targets". The elements of syntax catagories of Sub, Res, and Act are called the "subtargets". The elements of syntax catagories of Ps, Pol, and R are called the "laws".

2.2 XACML request syntax

Q	::=	`((`S`) (`R`) (`A`))`
S	::=	AVP*
R	::=	AVP*
A	::=	AVP*
AVP	::=	`(` id val `)`

2.3 Parsing

To make the semantics more understandable, we assume the existence of a parser. Informally, we assume that is parser can take a string generated by the grammar and produce denotational object suggested by the strings form, mostly lists. Since the above syntax matches the list notation in Scheme, this parser is equivalent to the Scheme command quote.

3 Semantics

Let $\mathcal{P}$ be the set of all policies (members of the syntactic category C of the language Spec). $\mathcal{P}$ includes both XACML Policies and PolicySets.[1] Let $\mathcal{Q}$ be the set of all requests (members of the syntactic category Q). Let $\mathcal{D}$ be the set of all decisions ($\mathcal{D} = \{\mathsf{Permit}, \mathsf{Deny}, \mathsf{NotApplicable}\}$). We will use the symbol used for the nonterminal in syntax in the natural semantics to refer to any element of the syntax category.

At the core of the semantics of XACML is the notation of a request matching the target of a rule, policy, or policy set. We will denote this relation by $q \Subset t$ where $q \in \mathcal{Q}$ and t is a target. We will define $\Subset$ using a natural semantics given below in Table 1.

The semantics of the Margrave subset of XACML is given in a natural semantics that makes use of the relation $\Subset$. The results of evaluating a Rule is given in Table 2 in terms of the $\models_r$ relationship. The $\models_r$ relation is then used to define the $\models$ relation, which gives the semantics of our subset of XACML. The default behavior of $\models$ is given in Table 3. The behavior of $\models$ under each of the combining algorithms are given in Table 4 for permit-overrides, in Table 5 for deny-overrides, and in Table 6 for first-applicable.

4 Margrave

Margrave is a tool for the analysis of XACML policies. The key function in Margrave takes an XACML policy and produces a function from requests to decisions. We model this function as Mar:

$$\textsc{Mar} : \mathcal{P} \to (\mathcal{Q} \to \mathcal{D}).$$

Below we define Mar as pseudo-Scheme program that consumes a parsed XACML file and produces a data structure called a MTBDD that defines the returned function from requests to decisions. First we describe MTBDDs and then we give a definition of Mar.

4.1 MTBDDs

Let f be a function from $\{\mathsf{T}, \mathsf{F}\}^n$ to a finite set of terminals M with the formal arguments called $\vec{x}$. The function f can be represented as a binary tree called a binary decision tree. Each leaf of the tree is labeled with an element of M. Each internal node represents a question of the form "is the ith element of ordered n-tuple $\vec{x}$ true?". Given $\vec{x}$, one determines $f(\vec{x})$ by starting at the root node and if the answer to the question at the current node is "yes", recur on the right child (the high branch); otherwise, recur on left child (the low branch). The base case is reached at a leaf. The returned value is the label at the reached leaf.

The same procedure succeeds when the tree requirement is lifted and directed acyclic graphs (DAGs) are allowed. Such a DAG is called a multi-terminal binary decision diagram (MTBDD). Given an MTBDD, one can easily convert it to be an MTBDD in which at most one leaf node is labeled by

[1] The word "policy" when uncapitalized will refer to access-control policies in general. When "policy" is capitalized, it will refer to the XACML tag `Policy` and the associated structure.

$$\frac{S \Subset_S \textsc{Sub} \qquad R \Subset_R \textsc{Res} \qquad A \Subset_A \textsc{Act}}{(S, R, A) \Subset (\textsc{Sub}, \textsc{Res}, \textsc{Act})} \tag{1}$$

$$X \Subset \texttt{Any} \quad \forall X \in \{\mathrm{S}, \mathrm{R}, \mathrm{A}\} \tag{2}$$

$$\frac{\exists i \text{ s.t. } \mathrm{S} \Subset \textsc{Allow}_i}{\mathrm{S} \Subset_S (\textsc{Allow}_1, \textsc{Allow}_2, \ldots, \textsc{Allow}_n)} \tag{3}$$

$$\frac{\exists i \text{ s.t. } \mathrm{R} \Subset \textsc{Allow}_i}{\mathrm{R} \Subset_R (\textsc{Allow}_1, \textsc{Allow}_2, \ldots, \textsc{Allow}_n)} \tag{4}$$

$$\frac{\exists i \text{ s.t. } \mathrm{A} \Subset \textsc{Allow}_i}{\mathrm{A} \Subset_A (\textsc{Allow}_1, \textsc{Allow}_2, \ldots, \textsc{Allow}_n)} \tag{5}$$

$$\frac{\forall i \quad X \Subset_X \mathrm{AVC}_i}{X \Subset_X (\mathrm{AVC}_1, \mathrm{AVC}_2, \ldots, \mathrm{AVC}_n)} \text{ where } X \in \{S, R, A\} \tag{6}$$

$$\frac{\exists j \text{ s.t. } \mathrm{AVP}_j = \mathrm{AVC}}{(\mathrm{AVP}_1, \mathrm{AVP}_2, \ldots, \mathrm{AVP}_n) \Subset_X \mathrm{AVC}} \text{ where } X \in \{S, R, A\} \tag{7}$$

Table 1: The Match Relationship

$$\frac{Q \notin T}{((\texttt{Rule} \quad T \quad \textsc{Effect}), Q) \models_r \mathsf{NotApplicable}} \tag{8}$$

$$\frac{Q \Subset T}{((\texttt{Rule} \quad T \quad \texttt{Permit}), Q) \models_r \mathsf{Permit}} \tag{9}$$

$$\frac{Q \Subset T}{((\texttt{Rule} \quad T \quad \texttt{Deny}), Q) \models_r \mathsf{Deny}} \tag{10}$$

Table 2: The Rule Relationship $\models_r$

$$\langle(\texttt{Policy} \quad \text{CA} \quad T), Q\rangle \models \mathsf{NotApplicable} \tag{11}$$

$$\langle(\texttt{PolicySet} \quad \text{CA} \quad T), Q\rangle \models \mathsf{NotApplicable} \tag{12}$$

$$\frac{Q \notin T}{\langle(\texttt{Policy} \quad \text{CA} \quad \text{T} \quad \text{R}^*), \text{Q}\rangle \models \mathsf{NotApplicable}} \tag{13}$$

$$\frac{Q \notin T}{\langle(\texttt{PolicySet} \quad \text{CA} \quad \text{T} \quad \text{C}^*), \text{Q}\rangle \models \mathsf{NotApplicable}} \tag{14}$$

Table 3: Default NotApplicable Judgements

$$\frac{Q \Subset T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models_r \mathsf{Permit}}{\langle(\texttt{Policy} \quad \texttt{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Permit}} \tag{15}$$

$$\frac{Q \Subset T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models \mathsf{Permit}}{\langle(\texttt{PolicySet} \quad \texttt{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Permit}} \tag{16}$$

$$\frac{Q \Subset T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models_r \mathsf{Deny} \qquad \forall j \neq i, \langle C_j, Q\rangle \not\models_r \mathsf{Permit}}{\langle(\texttt{Policy} \quad \texttt{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Deny}} \tag{17}$$

$$\frac{Q \Subset T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models \mathsf{Deny} \qquad \forall j \neq i, \langle C_j, Q\rangle \not\models \mathsf{Permit}}{\langle(\texttt{PolicySet} \quad \texttt{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Deny}} \tag{18}$$

$$\frac{Q \Subset T \qquad \forall i, \langle C_i, Q\rangle \models_r \mathsf{NotApplicable}}{\langle(\texttt{Policy} \quad \texttt{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{NotApplicable}} \tag{19}$$

$$\frac{Q \Subset T \qquad \forall i, \langle C_i, Q\rangle \models \mathsf{NotApplicable}}{\langle(\texttt{PolicySet} \quad \texttt{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{NotApplicable}} \tag{20}$$

Table 4: Permit-Overrides Judgements

$$\frac{Q \notin T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models_r \mathsf{Deny}}{\langle(\texttt{Policy}\ \ \texttt{Deny-Overrides}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Deny}} \tag{21}$$

$$\frac{Q \notin T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models \mathsf{Deny}}{\langle(\texttt{PolicySet}\ \ \texttt{Deny-Overrides}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Deny}} \tag{22}$$

$$\frac{Q \notin T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models_r \mathsf{Permit} \qquad \forall j \neq i, \langle C_j, Q\rangle \not\models_r \mathsf{Deny}}{\langle(\texttt{Policy}\ \ \texttt{Deny-Overrides}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Permit}} \tag{23}$$

$$\frac{Q \notin T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models \mathsf{Permit} \qquad \forall j \neq i, \langle C_j, Q\rangle \not\models \mathsf{Deny}}{\langle(\texttt{PolicySet}\ \ \texttt{Deny-Overrides}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Permit}} \tag{24}$$

$$\frac{Q \notin T \qquad \forall i, \langle C_i, Q\rangle \models_r \mathsf{NotApplicable}}{\langle(\texttt{Policy}\ \ \texttt{Deny-Overrides}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{NotApplicable}} \tag{25}$$

$$\frac{Q \notin T \qquad \forall i, \langle C_i, Q\rangle \models \mathsf{NotApplicable}}{\langle(\texttt{PolicySet}\ \ \texttt{Deny-Overrides}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{NotApplicable}} \tag{26}$$

Table 5: Deny-Overrides Judgements

$$\frac{Q \notin T \qquad \langle R_1, Q\rangle \models \mathsf{Permit}}{\langle(\texttt{Policy}\ \ \texttt{First-Applicable}\ \ T\ \ R_1, R_2, \ldots, R_n), Q\rangle \models \mathsf{Permit}} \tag{27}$$

$$\frac{Q \notin T \qquad \langle C_1, Q\rangle \models \mathsf{Permit}}{\langle(\texttt{PolicySet}\ \ \texttt{First-Applicable}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Permit}} \tag{28}$$

$$\frac{Q \notin T \qquad \langle R_1, Q\rangle \models \mathsf{Deny}}{\langle(\texttt{Policy}\ \ \texttt{First-Applicable}\ \ T\ \ R_1, R_2, \ldots, R_n), Q\rangle \models \mathsf{Deny}} \tag{29}$$

$$\frac{Q \notin T \qquad \langle C_1, Q\rangle \models \mathsf{Deny}}{\langle(\texttt{PolicySet}\ \ \texttt{First-Applicable}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{Deny}} \tag{30}$$

$$\frac{\begin{array}{c}Q \notin T \\ \langle R_1, Q\rangle \models \mathsf{NotApplicable} \qquad \langle(\texttt{Policy}\ \ \texttt{First-Applicable}\ \ T\ \ R_2, \ldots, R_n), Q\rangle \models D\end{array}}{\langle(\texttt{Policy}\ \ \texttt{First-Applicable}\ \ T\ \ R_1, R_2, \ldots, R_n), Q\rangle \models D} \quad \forall D \in \mathcal{D} \tag{31}$$

$$\frac{\begin{array}{c}Q \notin T \\ \langle C_1, Q\rangle \models \mathsf{NotApplicable} \qquad \langle(\texttt{PolicySet}\ \ \texttt{First-Applicable}\ \ T\ \ C_2, \ldots, C_n), Q\rangle \models D\end{array}}{\langle(\texttt{PolicySet}\ \ \texttt{First-Applicable}\ \ T\ \ C_1, C_2, \ldots, C_n), Q\rangle \models D} \quad \forall D \in \mathcal{D} \tag{32}$$

Table 6: First-Applicable Judgements

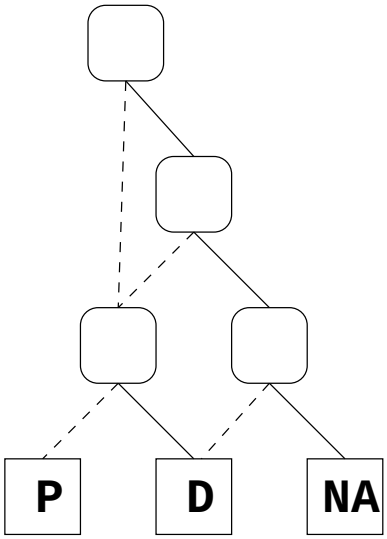

Figure 1: An example MTBDD. The low branches are shown as dashed lines and the high branches as solid lines.

each terminal: for each $m \in M$, make a new node n labeled with m; for each edge going from an internal node n' to a leaf labeled with m, make an edge going from n' to n; remove all the leaves labeled with m except n and all the edges going to them. Henceforth, we assume that all MTBDDs discussed have this property. If $\hat{f}$ is an MTBDD representation of f, we will let $\hat{f}(\vec{x}) = f(\vec{x})$.

A Binary Decision Diagram (BDD) is an MTBDD in which the set of terminals is $\{\mathsf{T}, \mathsf{F}\}$. A BDD may be viewed as a set where all elements that are mapped to T are in the set and all mapped to F are not in the set. Each element of the set formed from a BDD is itself a set of answers to the questions in the BDD. We let $\in_{\text{bdd}}$ represent: an element is "in" the BDD. That is, if $\hat{b}$ is a BDD representing a set B with a characteristic function of b, then

$$x \in_{\text{bdd}} \hat{b} \iff \hat{b}(x) \iff b(x) \iff x \in B \tag{33}$$

Given a policy, Margrave constructs a MTBDD that represents that policy. The terminals of this MTBDD are elements of the set $\{\mathsf{Permit}, \mathsf{Deny}, \mathsf{NotApplicable}\}$. During this process Margrave uses BDDs. For both the three-terminal MTBDDs and the BDDs. $\vec{x}$ is a representation of a request.

The algorithm makes use of the following constants: *deny-term*, *permit-term*, *na-term*, *true-term*, and *false-term* are the leaf node labeled the Deny, Permit, NotApplicable, T, and F, respectively.

The algorithm makes use of a few basic operations on MTBDDs given below[2]:

(**define** (*make-bdd subtarget-name attribute-id attribute-value*)

[2]Although many of the following algorithms are given a Scheme like notation, we intend for these algorithms to given in the meta-language and not an additional object language. Thus, no distinction is made between variables in the Scheme notation and meta-variables.

creates a BDD with one internal node which corresponds to the question "Does the *attribute-id* in the *subtarget-name* have the value *attribute-value*?" and has the terminal T on its high bench and the terminal F on its low branch. Formally, (*make-bdd n i v*) returns a BDD that represents a function f from $\mathcal{Q}$ to $\{\mathsf{T}, \mathsf{F}\}$ such given a request $q = (S, R, A)$

$$f = \lambda(S, R, A) \in \mathcal{Q}[(i, v) \in X] \tag{34}$$

where $X = S$ if $n =$ "Subject", $X = R$ if $n =$ "Resource", $X = A$ if $n =$ "Action".

(*bdd-and l-bdd r-bdd*)

returns the BDD that represents the function $f = \lambda\vec{x}[f_l(\vec{x}) \wedge f_r(\vec{x})]$ where f_l is the function represented by *l-bdd*, and f_r by *r-bdd*. This function is extended to a list of BDDs as

```
(define (bdd-and* bdd-list)
  (if (empty? bdd-list)
      true-term
      (bdd-and (first bdd-list) (bdd-and* (rest bdd-list)))))
```

(*bdd-or l-bdd r-bdd*)

returns the BDD that represents the function $f = \lambda\vec{x}[f_l(\vec{x}) \vee f_r(\vec{x})]$ where f_l is the function represented by *l-bdd*, and f_r by *r-bdd*. This function is extended to a list of BDDs as

```
(define (bdd-or* bdd-list)
  (if (empty? bdd-list)
      false-term
      (bdd-or (first bdd-list) (bdd-or* (rest bdd-list)))))
```

(*replace-terminal terminal replacing-mtbdd in-mtbdd*)

returns a BDD that is identical to *in-mtbdd* except that the terminal *terminal* is replaced by the whole MTBDD *replacing-mtbdd*. (After the replacement all the leaf nodes labeled with the same value are combined as above.)

4.2 The Algorithm of Margrave

Rather than give the transition rules and axioms as in the normal semantics, we give pseudo-Scheme code so as to follow the actual implementation discussed in section 4.3 more closely.

The algorithm consists of two major parts: creating a BDD representations of targets and combining these to represent Rules, Policies, and PolicySets.

4.2.1 Representing Targets

The target of a policy may be viewed as a representation of the set of requests to which this policy applies. The Margrave algorithm creates these sets using BDDs. The algorithm *process-target* takes a (parsed) target and converts it to such a BDD.

The algorithm for creating and representing these sets as BDDs is given below. Note that the functions starting with *ast:* simply access the named child in the abstract syntax tree. For example, given a element of the syntax catagory of the nonterminal T called *target*, (*ast:target-subjects target*) returns the subject section of the target. If the abstract syntax tree is viewed as scheme list as explaned in Section 2.3, this would be (*first target*).

```
(define (process-target target)
  (bdd-and (process-subtarget 'Subject (ast:target-subjects target))
           (process-subtarget 'Resource (ast:target-resources target))
           (process-subtarget 'Action (ast:target-actions target))))

(define (process-subtarget subtarget-name subtarget-body)
  (if (= (first subtarget-body) 'Any)
      true-term
      (bdd-or* (map process-allowance subtarget-body))))

(define (process-allowance subtarget-name match-list)
   (bdd-and* process-avc match-list)))

(define (process-avc subtarget-name match)
  (make-bdd subtarget-name
            (ast:attribute-id match)
            (ast:attribute-value match)))
```

4.2.2 Forming Rules, Policies, and PolicySets

Each law can be viewed a function from requests to a decision (Permit, Deny, or NotApplicable). The function *ast->mtbdd* takes a (parsed) law and returns just such a function.

```
(define (ast->mtbdd law)
    (cond [(ast:rule? law)
           (augment-rule (process-target (ast:law-target law))
                         (cond [(eq? (ast:rule-effect law) 'Deny) deny-term]
                               [(eq? (ast:rule-effect law) 'Permit) permit-term]))]
          [(or (ast:policy? law) (ast:policySet? law))
           (let [ [ca (ast:get-ca law)]
                  [children (map ast->mtbdd (ast:get-children law))] ]
             (augment-rule (process-target (ast:law-target law))
                           (cond [(equal? ca "first-applicable")
                                  (build-first-applicable children)]
                                 [(equal? ca "deny-overrides")
                                  (build-deny-override children)]
                                 [(equal? ca "permit-overrides")
                                  (build-permit-override children)])))]))
```

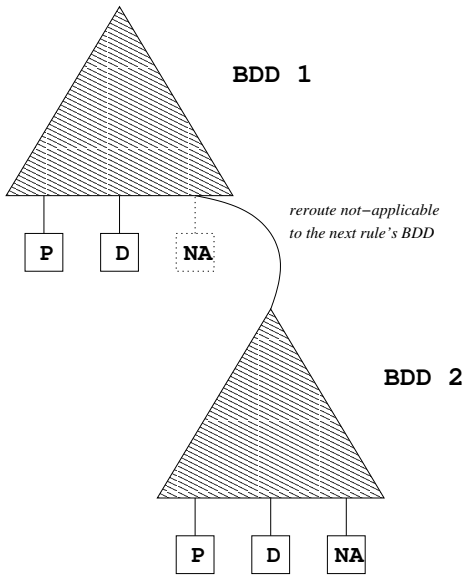

Figure 2: First-applicable.

```
(define (augment-rule target-bdd effect-term)
  (replace-terminal true-term effect-term
                                    (replace-terminal false-term na-term target-bdd))

(define (build-first-applicable children)
  (if (empty? children)
      na-term
      (replace-terminal na-term (build-first-applicable (rest children))
                                (first children))
```

Deny-overrides can be seen in Figure 2.

```
(define (build-permit-overrides children)
  (if (empty? children)
      na-term
      (replace-terminal deny-term
                        (replace-terminal na-term deny-term (build-permit-overrides (rest children)))
                        (replace-terminal na-term (build-permit-overrides (rest children))
                                                  (first children)))))
```

Permit-overrides can be seen in Figure 3.

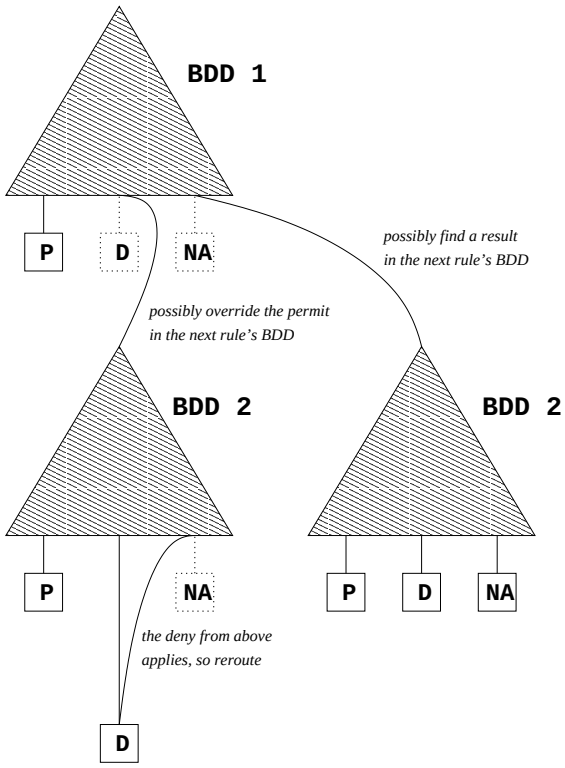

Figure 3: Permit-override.

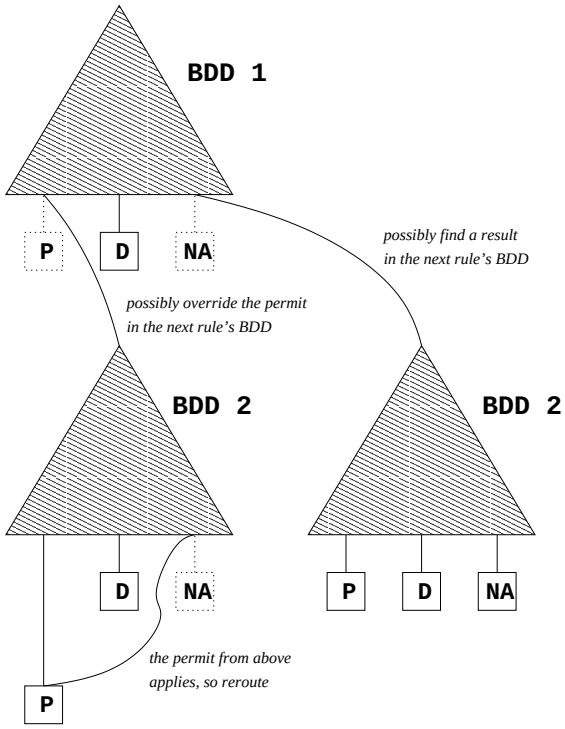

Figure 4: Deny-override.

```
(define (build-deny-overrides children)
  (if (empty? children)
      na-term
      (replace-terminal permit-term
                        (replace-terminal na-term permit-term (build-deny-overrides (rest children)))
                        (replace-terminal na-term (build-deny-overrides (rest children))
                                                  (first children)))))
```

Deny-overrides can be seen in Figure 4.

The function MAR is (**lambda** (*policy*) (*ast->mtbdd* (*parse policy*))).

4.3 An Implementation of Margrave

An implementation of Margrave exists and is available at `www.cs.brown.edu/research/plt/software/margrave`. The implementation differs from the above algorithm in that the implementation uses reduced ordered multi-terminal BDDs instead of just MTBDDs [1]. Since the relationship between ROMTBDDs and general MTBDDs are well understood and the algorithms to maintain reduced ordering complicates the proof considerably, the following proofs are not for ROMTBDDs.

5 Soundness and Completeness of Margrave Analysis

Margrave is equivalent to XACML evaluation in the following sense

$$\forall p \in \mathcal{P}, \forall q \in \mathcal{Q}, [\langle p, q\rangle \models d \Longleftrightarrow (\textsc{Mar}(p))(q) = d)]. \tag{35}$$

The proof of this equivalence follows from the soundness and completeness of Margrave with respect to XACML evaluation as shown in the next two sections. Below we will use the symbol that represents a nonterminal in the syntax to refer to the set of all strings in that syntax category.

5.1 The Match Relationship

Lemma 1.

$$\forall (i,v) \in AVC, \forall (s,r,a) \in \mathcal{Q}, [(s,r,a) \in_{bdd} \textit{(process-avc 'Subject (i, v))} \iff s \Subset_S (i,v)] \tag{36}$$
$$\forall (i,v) \in AVC, \forall (s,r,a) \in \mathcal{Q}, [(s,r,a) \in_{bdd} \textit{(process-avc 'Resource (i, v))} \iff r \Subset_R (i,v)] \tag{37}$$
$$\forall (i,v) \in AVC, \forall (s,r,a) \in \mathcal{Q}, [(s,r,a) \in_{bdd} \textit{(process-avc 'Action (i, v))} \iff a \Subset_A (i,v)] \tag{38}$$

Proof. For the subject case, *(process-avc 'Subject (i, v))* produces a BDD with one internal node. The internal node represents the question "Does the request include in the Subject subrequest the id-value pair (i,v)?". The high branch goes to T and the low branch to F. Thus, $(s,r,a) \in_{\text{bdd}}$ *(process-ac 'Subject (i, v))* if and only if (s,r,a) includes attribute-value pair (i,v) in the Subject subtarget. Formally, $(s,r,a) \in_{\text{bdd}}$ *(process-ac 'Subject (i, v))* if and only if $\exists (i',v') \in S$ s.t. $(i',v') = (i,v)$. Such a request (s,r,a) will make $s \Subset_S (i,v)$ true since this is the same requirement to for Judgement 7.

For the other direction for the subject case, if $q \Subset (i,v)$, then $\exists (i',v') \in s$ s.t. $(i',v') = (i,v)$. This will allow the high branch of the single node in the BDD created to be taken yielding T. Thus, $q \Subset (i,v)$ implies $(s,r,a) \in_{\text{bdd}}$ *(process-avc 'Subject (i, v))*.

The resource and action cases follow the same reasoning as the subject case. □

Lemma 2.

$$\forall \ell \in \textit{Allow}, \forall (s,r,a) \in \mathcal{Q}, [(s,r,a) \in_{bdd} \textit{(process-allowance 'Subject } \ell\textit{)} \iff q \Subset \ell] \tag{39}$$
$$\forall \ell \in \textit{Allow}, \forall (s,r,a) \in \mathcal{Q}, [(s,r,a) \in_{bdd} \textit{(process-allowance 'Resource } \ell\textit{)} \iff q \Subset \ell] \tag{40}$$
$$\forall \ell \in \textit{Allow}, \forall (s,r,a) \in \mathcal{Q}, [(s,r,a) \in_{bdd} \textit{(process-allowance 'Action } \ell\textit{)} \iff q \Subset \ell] \tag{41}$$

Proof. For subject case: Let $(i_j, v_j) \in AC$ be the jth element of $\ell \in \textit{Allow}$ where j ranges from 1 to $|\ell|$. Let b be *(process-allowance 'Subject ℓ)*. *process-allowance* produces b as the *bdd-and* of *(process-avc 'Subject* (i_j, v_j)*)* for all j. Thus, $(s,r,a) \in_{\text{bdd}} b$ if and only if

$$(s,r,a) \in_{\text{bdd}} \textit{(process-avc 'Subject } (i_j, v_j)\textit{)} \tag{42}$$

for all j. Thus,

$$(s,r,a) \in_{\text{bdd}} \textit{(process-allowance 'Subject } \ell\textit{)} \iff \forall j, (s,r,a) \in_{\text{bdd}} \textit{(process-avc 'Subject } (i_j, v_j)\textit{)} \tag{43}$$

$$\iff \forall j, (s,r,a) \Subset_S (i_j, v_j) \tag{44}$$

$$\iff (s,r,a) \Subset_S \ell. \tag{45}$$

where middle step follows from Lemma 1 and the last from Judgement 6.
The resource and action cases follows the same steps. □

Lemma 3.

$$\forall s' \in Sub, \forall (s,r,a) \in \mathcal{Q}, \quad [(s,r,a) \in_{bdd} \textit{(process-subtarget 'Subject } s'\textit{)} \iff s \Subset s'] \tag{46}$$

$$\forall r' \in Res, \forall (s,r,a) \in \mathcal{Q}, \quad [(s,r,a) \in_{bdd} \textit{(process-subtarget 'Resource } r'\textit{)} \iff r \Subset r'] \tag{47}$$

$$\forall a' \in Act, \forall (s,r,a) \in \mathcal{Q}, \quad [(s,r,a) \in_{bdd} \textit{(process-subtarget 'Action } a'\textit{)} \iff a \Subset a'] \tag{48}$$

Proof. For the subject case: There are two cases: (1) s' is either a list of elements from *Allow* or (2) s' is the singleton list holding the keyword "`Any`".

1. $s' \in Allow^+$: Let $\ell_j \in Allow$ be the jth element of $s' \in Sub$ where j ranges from 1 to $|s|$. *(process-subtarget 'Subject s')* produces the *bdd-or* of *(process-allowance 'Subject ℓ_j)* over all j. Thus, $(s,r,a) \in_{\text{bdd}}$ *(process-subtarget 'Subject s')* if and only if $(s,r,a) \in_{\text{bdd}}$ *(process-allowance 'Subject ℓ_j)* for at least one j. Thus,

   $$(s,r,a) \in_{\text{bdd}} \textit{(process-subtarget 'Subject } s'\textit{)} \iff \exists j, (s,r,a) \in_{\text{bdd}} \textit{(process-allowance 'Subject } \ell_j\textit{)} \tag{49}$$

   $$\iff \exists j, (s,r,a) \Subset_S \ell_j \tag{50}$$

   $$\iff (s,r,a) \Subset_S s'. \tag{51}$$

   where middle step follows from Lemma 2 and the last from Judgement 3.

2. $s' =$ `(Any)`: In this case, *process-subtarget* will return *true-term*. Thus, for any request (s,r,a), $(s,r,a) \in_{\text{bdd}}$ *(process-subtarget 'Subject s')*. Equivalently, by Judgement 2, all requests $(s,r,a) \in \mathcal{Q}$, $s \Subset$ `(Any)`.

The resource and action cases follows the same steps. □

Lemma 4.

$$\forall (s',r',a') \in Target, \forall (s,r,a) \in \mathcal{Q}, [(s,r,a) \in_{bdd} \textit{(process-target } (s',r',a')\textit{)} \iff (s,r,a) \Subset (s',r',a')] \tag{52}$$

Proof. *(process-target (s',r',a'))* constructs the *bdd-and* of

- *(process-subtarget 'Subject s')*,

- *(process-subtarget 'Resource r')*, and
- *(process-subtarget 'Action a').*

Thus,

$$(s, r, a) \in_{\text{bdd}} \textit{(process-target } (s', r', a')\textit{)} \tag{53}$$
$$\iff \textit{(process-subtarget 'Subject } s'\textit{)} \wedge \textit{(process-subtarget 'Resource } r'\textit{)} \wedge \textit{(process-subtarget 'Action } a'\textit{)} \tag{54}$$
$$\iff (s, r, a) \Subset_S s' \wedge (s, r, a) \Subset_R r' \wedge (s, r, a) \Subset_A a' \tag{55}$$
$$\iff (s, r, a) \Subset (s', r', a'). \tag{56}$$

where middle step follows from Lemma 3 and the last from Judgement 1. □

5.2 Rules

Lemma 5.

$$\forall r \in \mathit{Rule}, \forall q \in \mathcal{Q}, [(\textsc{Mar}(r))(q) = d \iff \langle p, q\rangle \models_r d]. \tag{57}$$

Proof. The relative branch of *ast->add* is as follows:

```
(augment-rule (process-target (ast:law-target law))
              (cond [(eq? (ast:rule-effect law) 'Deny) deny-term]
                    [(eq? (ast:rule-effect law) 'Permit) permit-term]))]
```

The function *augment-rule* first replaces the false terminal with *na-term*, which means that any request that yields false in BDD (was not $\in_{\text{bdd}}$ the BDD) will now yield NotApplicable. Second, *augment-rule* replaces the true terminal of the BDD produced by (*process-target* (*ast:law-target law*)) with the effect of the rule (either a *deny-term* or *permit-term* as selected the effected list in *law*).

Consider two cases: (1) where the request does not match the target and (2) where the request matches the target:

1. From Lemma 4, we know that the only requests that will yield F from BDD produced by (*process-target* (*ast:law-target law*)) are those that are not in the match relationship with the target. Thus, since the terminal for F was replaced by the terminal for NotApplicable and none of the internal nodes were altered, the MTBDD that is produced by *augment-rule* will yield NotApplicable if and only if the request was not in the match relationship $\Subset$ with the target. This is the same behavior of the rule relationship $\models_r$ found in Table 2 (note esp. Judgement 8).

2. Now consider the case where the match relationship does hold (as in Judgements 9 and 10). Consider the sub-case where the effect of the rule is `Permit`. Since the terminal for T was replaced by the terminal for Permit and none of the internal nodes were altered, the MTBDD that is produced by *augment-rule* will yield Permit if and only if the request is in the match

relationship $\Subset$ with the target. Thus, the same behavior as found in Table 2 will be observed (note esp. Judgement 9). The proof of equivalence for the deny case follows in the same way.

Thus, in both cases MAR and $\models_r$ behaves the same making the above equivalence true. □

5.3 Policy

Lemma 6.

$$\forall p \in Pol, \forall q \in \mathcal{Q}, [(\text{MAR}(p))(q) = d \implies \langle p, q\rangle \models d]. \tag{58}$$

Proof. The case where the request does not match the target of the policy is almost identical to the corresponding case in the proof of Lemma 5. For the case where the request does match the target, we must proof that result returned by rule-combining function (either *build-first-applicable*, *build-permit-overrides*, or *build-deny-overrides*) is correct. We use proof by induction over the number of rules in p.

Base Case: Zero rules. The functions *build-first-applicable*, *build-permit-overrides*, and *build-deny-overrides* all return *na-term* for the rule-less policy. $\forall p \in \mathcal{P}, \forall q \in \mathcal{Q}, \langle p, q\rangle \models \mathsf{NotApplicable}$ holds when p is rule-less but not for either $\langle p, q\rangle \models \mathsf{Permit}$ or $\langle p, q\rangle \models \mathsf{Deny}$.

Inductive Case: Assume for $n \geq 0$ rules that $\forall p \in Pol, \forall q \in \mathcal{Q}, [q \in \text{MAR}(p, d) \implies \langle p, q\rangle \models d]$. Let p be in Pol and have $n + 1$ rules.

1. First-Applicable: Since the number of child rules is greater than zero, the applicable branch in *build-first-applicable* is

   (*replace-terminal na-term* (*build-first-applicable* (*rest children*))
   (*first children*)).

   Since (*first children*) is just a rule, it is correct by Lemma 5. Since the number rules in (*rest children*) is n, by the inductive hypothesis, (*build-first-applicable* (*rest children*)) is correct. These other rules only matter if the first rule yields NotApplicable for given request. Otherwise, the result of the first rule is the result for the given request. The results of *replace-terminal* ensures this behavior. If the first rule yields NotApplicable, then the other rules are consulted since the terminal for NotApplicable gets replaced by the correct MTBDD that results from the other rules. The other two terminals are unchanged and thus correctly returned.

2. Permit-Overrides: Since the number of child rules is greater than zero, the applicable branch in *build-permit-overrides* is

   (*replace-terminal deny-term*
   (*replace-terminal na-term deny-term* (*build-permit-overrides* (*rest children*)))
   (*replace-terminal na-term* (*build-permit-overrides* (*rest children*))
   (*first children*)))).

Lets decompose this as follows:

$$\hat{b} = \textit{(first children)} \tag{59}$$
$$\hat{c} = \textit{(build-permit-overrides (rest children))} \tag{60}$$
$$\hat{d} = \textit{(replace-terminal na-term } \hat{c}\ \hat{b}\textit{)} \tag{61}$$
$$\hat{e} = \textit{(replace-terminal na-term deny-term } \hat{c}\textit{)} \tag{62}$$
$$\hat{f} = \textit{(replace-terminal deny-term } \hat{e}\ \hat{d}\textit{)} \tag{63}$$

First we will prove soundness and then completeness:

Soundness:

(a) $\hat{f}(q) = \mathsf{Permit}$: For $\hat{f}(q)$ to equal Permit, either $\hat{d}(q) = \mathsf{Deny}$ and $\hat{e}(q) = \mathsf{Permit}$, or $\hat{d}(q) = \mathsf{Permit}$.

If $\hat{e}(q) = \mathsf{Permit}$, then $\hat{c}(q) = \mathsf{Permit}$. Since $\hat{c}$ is produced by n rules, by the inductive hypothesis, $\hat{c}(q) = \mathsf{Permit}$ is sound. That is, if p' is a policy created out the last n rules of the given policy p, then $\langle p', q\rangle \models \mathsf{Permit}$. Thus, since the only applicable judgement that implies that Permit is yielded is Judgement 15, the antecedents of Judgement 15 must hold. Thus, there exists a rule in (*rest children*) that yields Permit. This makes Judgement 15 applicable for policy p as well.

If $\hat{d}(q) = \mathsf{Permit}$, then either $\hat{b}(q) = \mathsf{Permit}$, or $\hat{b}(q) = \mathsf{NotApplicable}$ and $\hat{c}(q) = \mathsf{Permit}$. If $\hat{b}(q) = \mathsf{Permit}$, then by Lemma 5, rule $n+1$ must yield Permit making Judgement 15 applicable. If $\hat{c}(q) = \mathsf{Permit}$,, then as above, Judgement 15 holds.

Thus, in either case, Judgement 15 holds, and thus, $\langle p, q\rangle \models \mathsf{Permit}$.

(b) $\hat{f}(q) = \mathsf{Deny}$: For $\hat{f}(q)$ to equal Deny, then both $\hat{e}(q)$ and $\hat{d}(q)$ must be Deny since $\hat{f}$ is $\hat{d}$ with its Deny terminal replaced by $\hat{e}$.

When $\hat{e}(q) = \mathsf{Deny}$, $\hat{c}(q)$ must equal either Deny or NotApplicable since $\hat{e}$ is $\hat{c}$ with its NotApplicable terminal changed to be a Deny terminal. Since $\hat{c}$ is created by n rules, by the inductive hypothesis, $\hat{c}$ must be sound. Thus, since both applicable judgements that yield Deny or NotApplicable (Judgements 17 and 19) require that no rule in the Policy yields Permit, no rule in (*rest children*) may yield Permit.

When $\hat{d}(q) = \mathsf{Deny}$, either $\hat{b}(q) = \mathsf{Deny}$, or $\hat{b}(q) = \mathsf{NotApplicable}$ and $\hat{c}(q) = \mathsf{Deny}$. If $\hat{b}(q) = \mathsf{Deny}$, then by Lemma 5, rule (*first children*) must yield Deny. As above, if $\hat{c}(q) = \mathsf{Deny}$, there must exist a rule that yields deny in (*rest children*) (see Judgement 17). Thus, under either case, $\hat{b}(q) \neq \mathsf{Permit}$ and there exists a rule the yields Deny.

From the implications of $\hat{e}(q) = \mathsf{Deny}$, no rule in (*rest children*) may not yield Permit. From implications of $\hat{d}(q) = \mathsf{Deny}$, the rule (*first children*) may not yield permit and there must exist a rule in the policy that must yields Deny. Thus, Judgement 17 applies and $\langle p, q\rangle \models \mathsf{Deny}$.

(c) $\hat{f}(q) = \mathsf{NotApplicable}$: For $\hat{f}(q)$ to equal NotApplicable, either $\hat{d}(q) = \mathsf{NotApplicable}$, or $\hat{d}(q) = \mathsf{Deny}$ and $\hat{e}(q) = \mathsf{NotApplicable}$. However, $\hat{e}(q)$ cannot equal NotApplicable since $\hat{e}$ has no NotApplicable terminal (it was replaced by a Deny terminal). Thus, $\hat{d}(q)$ must be NotApplicable. For $\hat{d}(q)$ to be NotApplicable, both $\hat{b}(q)$ and $\hat{c}(q)$ must be NotApplicable.

When $\hat{b}(q) = \mathsf{NotApplicable}$, by Lemma 5, (*first children*) must yield NotApplicable.

When $\hat{c}(q) = \mathsf{NotApplicable}$, by the inductive hypothesis, a policy with all the rules in (*rest children*) and on others, must yield NotApplicable. Since the only applicable judgement that yields NotApplicable is Judgement 19, the antecedents of that judgement must hold. Thus, all of the rules in (*rest children*) must yield NotApplicable.

Thus, every rule in p must yield NotApplicable and Judgement 19 is applicable. Thus, $\langle p, q\rangle \models \mathsf{NotApplicable}$.

Completeness:

(a) $\langle p, q\rangle \models \mathsf{Permit}$: Only one judgement can be applied to yield Permit: Judgement 15. According to the antecedent of Judgement 15, there must exist a rule in the policy that yields Permit. Thus, either rule $n+1$ yields Permit or at least on rule in (*rest children*) yields Permit.

In the first case where rule $n+1$ yields Permit, by Lemma 5 $\hat{b}$ will yield Permit. Since the creation of $\hat{f}$ from $\hat{b}$ does not change the Permit terminal or any of the internal nodes, $\hat{f}$ will yield Permit for the same requests as $\hat{b}$. Thus, $\hat{f}$ yields Permit.

In the second case where rule $n+1$ does not yield Permit but another rule in (*rest children*) does, by the inductive hypothesis the MTBDD $\hat{c}$ will yield Permit. Either rule $n+1$ yields NotApplicable or Deny. If it yields NotApplicable, then since in $\hat{f}$ it has its NotApplicable terminal replaced by $\hat{c}$, it will now produce what $\hat{c}$ produces: Permit. If it yields Deny, then since in $\hat{f}$ it has its Deny terminal replaced by $\hat{e}$, it will yields what this MTBDD $\hat{e}$ yields. Since $\hat{e}$ is just $\hat{c}$ with its NotApplicable terminal replaced by Deny and $\hat{c}$ yields Permit, $\hat{e}$ will yield Permit. Thus, $\hat{c}$ will yield Permit.

(b) $\langle p, q\rangle \models \mathsf{Deny}$: Only one judgement could can be applied to yield deny: Judgement 17. According to the antecedent of Judgement 17, there must exist a rule in the policy that yields Deny and no rules that yield Permit on q. Thus, either rule $n+1$ yields Deny or at least on rule in (*rest children*) yields Deny.

In the first case where rule $n+1$ yields Deny, by Lemma 5 $\hat{b}$ yields Deny. The creation of $\hat{f}$ from $\hat{b}$ changes the Deny terminal to be $\hat{e}$. Thus, $\hat{f}$ yields what $\hat{e}$ yields. Since no rules may yield Permit, by the inductive hypothesis, $\hat{c}$ does not yield Permit. Thus, $\hat{c}$ either yields Deny or NotApplicable. $\hat{e}$ is $\hat{c}$ with its NotApplicable terminal replaced with a Deny terminal. Thus, $\hat{e}$ always yields Deny. Thus, $\hat{f}$ yields Deny as required.

In the second case where rule $n+1$ does not yield Deny but another rule in (*rest children*) does, by the inductive hypothesis MTBDD $\hat{c}$ will yield Deny. Since rule $n+1$ may yield neither Permit nor Deny, it must yield NotApplicable. Since $\hat{f}$ is $\hat{b}$ with its NotApplicable terminal changed to $\hat{c}$, $\hat{f}$ will yield what $\hat{c}$ yields. Thus, $\hat{f}$ will yield Deny.

Thus, in either case, $\hat{f}$ under q will yield Deny if $\langle p, q\rangle \models \mathsf{Deny}$.

(c) $\langle p, q\rangle \models \mathsf{NotApplicable}$: Given that the target matches, only one judgement can be applied to yield deny: Judgement 19. According to the antecedent of Judgement 19, every rule with in the policy must yields NotApplicable. Thus, by Lemma 5, $\hat{b}(q) = \mathsf{NotApplicable}$. By the inductive hypothesis, $\hat{c}(q) = \mathsf{NotApplicable}$. Since $\hat{f}$ is $\hat{b}$ with its NotApplicable terminal replaced by $\hat{c}$, $\hat{f}(q) = \hat{c}(q) = \mathsf{NotApplicable}$.

Thus, in every case, $\langle p, q\rangle \models d \implies f(q) = d$

3. Deny-Overrides: Almost the same as Permit-Overrides.

□

5.4 PolicySets

Theorem 7 (Equivalence).

$$\forall p \in Pol, \forall q \in \mathcal{Q}, [(\textsc{Mar}(p))(q) = d \iff \langle p, q\rangle \models d]. \tag{64}$$

Proof. Let the *height* of a PolicySet be defined as 1 for any PolicySet that has only Policies as children (no PolicySets) and as one more than the maximum height of any of children for PolicySets that do have other PolicySets are children. Theorem will proved by induction over the height of the PolicySet.

Base Case: height 1, that is, a PolicySet with all its children being Policies. Given a PolicySet of height 1, it can have any number of children. By Lemma 6, we know that each of these children are both sound and complete. Now we must show that Margrave combines them correctly to yield a MTBDD that is sound and complete. Margrave uses the same function to combine Policies into PolicySets as it does to combine Rules into Policies. Inspecting the judgements for PolicySets, one can see that every rule for a PolicySet has a corresponding rule for Policy (listed right after it). The only differences in each pair of corresponding judgements are the syntactic category of children and the use of $\models$ instead of $\models_r$. The proof above for Lemma 6 only makes use of these two facts in using Lemma 5; it does not make use of any internal structure of rules or $\models_r$, only that Margrave is sound and complete with respect to rule and the relation $\models_r$. Lemma 6 shows that Margrave is sound and complete with respect to Policies and the relation $\models$ when used with Policies. Thus, Lemma 6 can be substituted for Lemma 5 everywhere in the proof of Lemma 6 to prove the base case of this theorem.

Inductive Case: Assume the inductive hypothesis that for all PolicySets of height n or less, $\forall p \in \mathcal{P}, \forall q \in \mathcal{Q}, [(\textsc{Mar}(p))(q) = d \iff \langle p, q\rangle \models d]$. Note that all the children of a PolicySet of height $n + 1$ has a height of n or less. Thus, the inductive hypothesis proves Margrave to be equivalent on all the children. Thus, as Lemma 6 stood in place of Lemma 5 in proof of the base case, the inductive hypothesis can stand in place of Lemma 5 in proof of the inductive case.

Thus, it has been proven that for a PolicySet of any finite height. Since only PolicySets of finite height are permissible by the grammar, the theorem is proven. □

6 Conclusion

This paper has presented a subset of XACML, provided a natural semantics for that subset, formalize an algorithm evaluating policies written in this subset, and proven the soundness and completeness of this algorithm.

The reader should bear in mind that full XACML is a much more rich set of operators. For example, full XACML results in not simply a decision being produced by each policy-request pair, but also a

list of obligations, actions that must be fulfilled upon the rendering of the decision (*e.g.*, notifying an auditor). How these lists of obligations are produced from the policy introduces non-determinism into the language. Future work shall have to provide a semantics for full XACML.

The algorithm presented is incapable of running on full XACML. Furthermore, due to the limitations of MTBDD construction, no way to extend the algorithm to full XACML presents itself. Future work should produce a tool capable of the analysis of XACML policies written in full XACML.

Acknowledgements

This paper was produced in the fulfillment of the requirements of the course CS 296 section 01 in Department of Computer Science, Brown University. The authors wish to thank the instructor Shriram Krishnamurthi for offering this course.

References

[1] Fisler, K., Krishnamurthi, S., Meyerovich, L. A., and Tschantz, M. C. Verification and change impact analysis of access-control policies. In *27th International Conference on Software Engineering ICSE '05* (St. Louis, Missouri, May 2005). To appear.

[2] Moses, T. eXtensible Access Control Markup Language (XACML) version 1.0. Tech. rep., OASIS, Feb. 2003.