D-Algebra for Composing Access Control Policy Decisions[*]

Qun Ni
Purdue University, USA
ni@cs.purdue.edu

Elisa Bertino
Purdue University, USA
bertino@cs.purdue.edu

Jorge Lobo
IBM T.J. Watson, USA
jlobo@us.ibm.com

ABSTRACT

This paper proposes a $\mathfrak{D}$-algebra to compose decisions from multiple access control policies. Compared to other algebra-based approaches aimed at policy composition, $\mathfrak{D}$-algebra is the only one that satisfies both functional completeness (any possible decision matrix can be expressed by a $\mathfrak{D}$-algebra formula) and computational effectiveness (a formula can be computed efficiently given any decision matrix). The $\mathfrak{D}$-algebra has several relevant applications in the context of access control policies, namely the analysis of policy languages decision mechanisms, and the development of tools for policy authoring and enforcement.

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General—*security and protection*; D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Management, Security, Standardization

Keywords

Many-valued Logic, MV-Algebras, Policy Composition, Decision

1. INTRODUCTION

In order to satisfy various access control requirements from different kinds of applications, modern access control policy languages, such as XACML, usually adopt a hierarchical structure to organize policies (e.g. rules $\preceq$ policies $\preceq$ policy sets) and sophisticated methods to answer an access request. Given a request, several rules/policies in an XACML policy set may be applicable. Some applicable rules/policies may evaluate to "permit" (their effects are "permit", referred to as "positive rules"), but others may evaluate to "deny" (their effects are "deny", referred to as "negative rules"). Even more complicated, some rule/policy, including both positive rules and negative rules, may evaluate to "indeterminate", i.e., whether this rule/policy is applicable or not cannot be determined due to missing information or errors, e.g. values of some attributes used in the rule/policy cannot be retrieved during evaluation. To combine these conflicting decisions into one final decision, XACML allows policy authors to specify, in a policy or a policy set, a *rule/policy combining algorithm* from a set of predefined standard algorithms. Examples of such algorithms include "deny-overrides" and "permit-overrides". Although a lot of common sense and practical experience went into the decisions behind the XACML rule and policy combination algorithms, the decision model lacks formal semantics and therefore unintended results may be generated by these standard policy combination algorithms.

As an example consider a policy set ps_1 that consists of two policies p_1 and p_2 (see Figure 1). Moreover assume that r_1 and r_2 are positive rules in p_1 and p_2, respectively. For a given access request, if r_1 evaluates to "permit" and r_2 evaluates to "indeterminate", we would expect the final result of ps_1 for the access request is "permit". If there is no error or no missing information r_2 can only evaluate to either "permit" or "not applicable". If r_2 evaluates to "permit", the final effect is "permit", and if r_2 evaluates to "not applicable", the final effect is "permit" too. However, based on the standard policy combining algorithm provided in XACML 2.0 and 3.0 WD 6, ps_1 returns "deny". The reason for such an unintended result is that the meaning of "indeterminate" is overloaded, which indeed may represent different access decisions. XACML policy combining algorithms cannot distinguish these different "indeterminate", thus may generate such an unintended result. Indeed, as indicated in Section 4.2, all standard policy combining algorithms suffer from some problems.

The lack of formal semantics in a decision model results in not only unintended decision composition but also inappropriate access control policy decision evaluation. For instance, the rule evaluation truth table and policy evaluation truth table in XACML, which determine the decision

[*]The work reported in this paper has been partially supported by IBM under the OCR project "Privacy and Security Policy Management", the NSF grant 0712846 "IPS: Security Services for Healthcare Applications", and MURI award FA9550-08-1-0265 from the Air Force Office of Scientific Research.

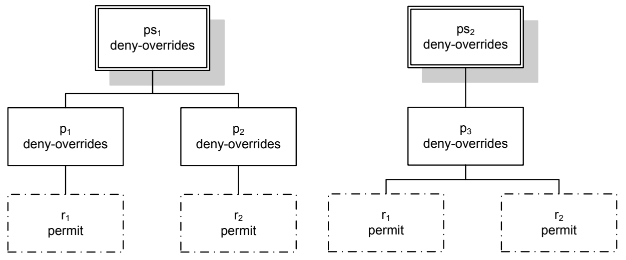


Figure 1: Policy set ps_1 and ps_2

of a rule or a policy given an access request, contain some inappropriate entries (detailed in Section 4.1).

Therefore, there is a strong need to develop a decision formalism suitable for modeling decisions in the context of access control that addresses the above drawbacks. $\mathfrak{D}$-algebra ($\mathfrak{D}$ stands for decision) is presented in this paper to meet the need. The powerset interpretation (P-interpretation) of $\mathfrak{D}$-algebra is able not only to highlight drawbacks from XACML rule/policy evaluation truth tables and policy combining algorithms but also to provide solutions to problems identified. Furthermore, $\mathfrak{D}$-algebra can go beyond XACML by supporting other decision combining algorithms, e.g. majority voting, and expressing and composing decisions other than those defined by XACML.

In this paper we provide a number of other contributions as well:

- We give a proof of the *functional completeness* of the $\mathfrak{D}$-algebras. The importance of functional completeness is that given an arbitrary n dimensional decision matrix, the corresponding decision function can be expressed by operations in $\mathfrak{D}$-algebra.
- We develop a fine-grained method for the composition of rule decisions. Unlike the current XACML composition approach that only allows a single rule combing algorithm per set of rules in a policy, a decision expression can precisely control decision composition of any two rules.
- We show that a rational number interpretation (R-interpretation) of the $\mathfrak{D}$-algebra can be used to process popular majority voting schemes. Majority voting represents a different composition methodology from standard combination algorithms in XACML and is elaborated in Section 6.
- We propose an algebra-based PDP (Policy Decision Point) design. One of the benefits of such design is that under a collaborating environment all PDPs can process user-defined combining algorithms from all participants.

The remaining of this paper is organized as follows: Section 2 discusses the motivation; Section 3 introduces the $\mathfrak{D}$-algebra, its P-interpretation, and the concepts of its functional and computational effectiveness; Section 4 analyzes issues concerning the policy decision mechanism of XACML and presents corresponding solutions; Section 5 discusses a method for expressing fine-grained decision requirements; Section 6 shows that the $\mathfrak{D}$-algebra can handle popular majority voting schemes; Section 7 discusses how the $\mathfrak{D}$-algebra can be used in a system for policy authoring and enforcement; Section 8 discussees related work. Section 9 concludes the paper and outlines future research directions.

2. MOTIVATION

There are intrinsic connections between XACML decisions, many-valued Łukasiewicz logic [15], referred to as $Ł_\infty$, and $\mathfrak{D}$-algebra. In this section, we briefly discuss the motivation of $\mathfrak{D}$-algebra.

$Ł_\infty$ is a many-value logic based on the rational numbers. Because each rule in XACML may evaluate to multiple different values, such as "permit", "deny", "not applicable", or "indeterminate" and these values may have to be further combined to generate a final decision, a $Ł_\infty$ logic would seem a reasonable choice in that we may be able to use different rational numbers to represent different decisions and use connectives of $Ł_\infty$ to express decision algorithms.

Although we can exploit the power of the soundness and completeness theorem [20, 10] for the whole class of $Ł_\infty$ logics, such logics cannot be directly applied to XACML policy decision-making for three reasons. First, the standard $Ł_\infty$ logic is not functionally complete. That means that some theorems of $Ł_\infty$ are not expressible by its logic connectives. In other words, some decision composition algorithms exist that cannot be expressed by $Ł_\infty$ logic formulae. For instance, first-applicable in XACML cannot be expressed by $Ł_\infty$ formulae.

Second, the rational number semantics of $Ł_\infty$ logics is not an appropriate representation for various decisions of XACML policies because the representation defines a total order among decisions prior to any policy composition principle. In order to handle the most general case, a fair treatment of each decision is required before the introduction of any policy composition principle.

Last but not least, $Ł_\infty$ logics is not computationally effective. By *computational effectiveness* we mean that for a functionally complete logics the computational complexity of its formula construction based on any given 2-D decision matrix remains tractable w.r.t. the number of decisions. The importance of computational effectiveness is motivated by the need of developing automatic tools able to assist policy authors when constructing composition formulae given some decision composition requirements. To the best of our knowledge, the computational effectiveness of many-valued logics in the sense described above has not been yet investigated.

These drawbacks of $Ł_\infty$ motivate $\mathfrak{D}$-algebra, an extension to the MV-algebra [9] proposed for describing the algebraic semantics of $Ł_\infty$ logic.

3. D-ALGEBRA

To be general, we use a set, referred to as a decision set, to formalize access control decisions. The decision set, together with some operations on it, forms the $\mathfrak{D}$-algebra.

Definition 1. Let D be a nonempty set of elements, 0 be a constant element of D, $\neg$ be a unary operation on elements in D, and $\oplus, \otimes$ be binary operations on elements in D. A $\mathfrak{D}$-algebra is an algebraic structure $\langle D, \neg, \oplus, \otimes, 0\rangle$ closed on $\neg, \oplus, \otimes$ and satisfying the following axioms:

1. $x \oplus y = y \oplus x$
2. $(x \oplus y) \oplus z = x \oplus (y \oplus z)$

3. $x \oplus 0 = x$

4. $\neg\neg x = x$

5. $x \oplus \neg 0 = \neg 0$

6. $\neg(\neg x \oplus y) \oplus y = \neg(\neg y \oplus x) \oplus x$

7. $x \otimes y = \begin{cases} \neg 0 & : \quad x = y \\ 0 & : \quad x \neq y \end{cases}$ □

Some readers may be curious about the meaning of $\neg 0$. If there is an order, either a total order or a partial order, on D, 0 usually represents the minimal or the least element in D. $\neg 0$ is another constant of D with a "complementary" meaning. For instance, if element 0 represents number 0, where $D = [0, 1]$, then $\neg 0$ represents number 1. If element 0 represents an empty set in a powerset, then $\neg 0$ represents the largest member in the powerset.

The binary operation $\oplus$ can be understood as a strong disjunction (the negation of the strong conjunction in $Ł_\infty$). One natural question is why $\mathfrak{D}$-algebra does not support absorption law, that is $x \oplus x = x$. At first sight it seems reasonable that two equal decisions yield the same decision. However, disjunction in a many-valued logic does not always follow the absorption law. For instance, in majority voting, the truth degree of the disjunction of two permits is greater than each independent permit even if each permit has the same truth degree. Assuming the absorption law actually limits the expressiveness of $\mathfrak{D}$-algebra.

The differences between a $\mathfrak{D}$-algebra and a MV-algebra are the new operation $\otimes$ and axiom 7. The intuition underlying axiom 7 is that there is a need for an equality test on two decisions when composing decisions. For instance, in order to realize the permit-overrides we may have to test whether one of the operands is permit or not, and $\otimes$ is introduced for this purpose. If two decisions are equal, $\neg 0$, the largest or biggest element in D. is returned. Otherwise, 0, the smallest element, is returned. The motivation of such a definition comes from axiom 3 and 5 which play key roles in the functional completeness of $\mathfrak{D}$-algebra.

We will see that even if all basic operations supported by $\mathfrak{D}$-algebra are commutative, it does not mean that only policy combining algorithms that are commutative are expressible by $\mathfrak{D}$-algebra. Later, we will see how $\mathfrak{D}$-algebra is used to express a first-applicable algorithm.

In order to write formulas and proofs in a compact form we also introduce the $\odot$ and $\ominus$ operations in a $\mathfrak{D}$-algebra.

Definition 2. Let $\langle D, \neg, \oplus, \otimes, 0\rangle$ be a $\mathfrak{D}$-algebra. $x \odot y = \neg(\neg x \oplus \neg y)$, and $x \ominus y = x \odot \neg y$ where $x, y \in D$. □

The interpretation of a $\mathfrak{D}$-algebra depends on the application and relevant composition principles. We take XACML as an example. If all relevant attribute values are retrievable and there are no errors, all policy rules in XACML, given a request, must evaluate to only one of these decisions: "permit", "deny", or "not applicable" (due to the no-match of a rule target or an unsatisfied rule condition). We refer to such decisions as *deterministic decisions.* To write XACML decisions in a compact form, we use the following set notations: $\{p\}$ denotes "permit", '$\{d\}$ denotes "deny", and $\{\frac{n}{a}\}$ denotes "not applicable".

During the evaluation of a positive rule, it is possible that the value of a subject attribute used in the rule condition cannot be retrieved. Therefore, the decision of this rule is indeterminate, in that it could be either permit or not applicable; we denote this case by the set $\{p, \frac{n}{a}\}$ and refer to it as a *non-deterministic decision.* A similar set $\{d, \frac{n}{a}\}$ denotes the non-deterministic decision of "deny" rules. Now suppose that two non-deterministic decisions $\{p, \frac{n}{a}\}$ and $\{d, \frac{n}{a}\}$ are returned for a request and the "deny takes precedence" principle is used. It is obvious that the final decision should be the set of all possible combinations between elements in these two decisions, that is, $\{p, d, \frac{n}{a}\}$[1].

Based on this observation, one possible interpretation of a $\mathfrak{D}$-algebra on XACML decisions, referred to as a *P-interpretation* (P stands for "power set"), is as follows:

- D is represented by the power set of the set of deterministic decisions $\{p, d, \frac{n}{a}\}$.
- 0 is represented by $\emptyset$.
- $\neg x$ is represented by $\{p, d, \frac{n}{a}\} - x$ where $x \in D$.
- $x \oplus y$ is represented by $x \cup y$ where $x, y \in D$.
- $\otimes$ is defined by axiom 7.

Someone may be confused by the definition of 0 and $\neg 0$ as access decisions in the P-interpretation. Basically, 0 here represents the simplest decision that is nothing, for instance, if there is no rule in a XACML policy, referred to as an empty policy, the policy decision should be an empty set [2]. There is a difference between an empty set decision and a "not applicable" decision. An empty set decision for a policy represents the situation in which there is simply no rule in the policy while a not applicable decision represents the situation in which there is at least one rule but not applicable.

On the contrary, $\neg 0$ represents the most complicated decision in D, that is $\{p, d, \frac{n}{a}\}$. The decision, though non-deterministic, is the most complicated decision an XACML policy can make, which means the decision can be one of "permit", "deny", or "not applicable" but not sure which of them should be taken. The P-interpretation obeys all axioms for a $\mathfrak{D}$-algebra and all operations are closed on D.

In the P-interpretation, the equality test of two permits yields a non-deterministic decision $\{p, d, \frac{n}{a}\}$. Such a situation may looks strange but indeed is meaningful because only the non-deterministic decision ensures axiom 5 in the P-interpretation.

In what follows we will show how the P-interpretation of $\mathfrak{D}$-algebras can be used to analyze the XACML standard decision-making mechanisms. Sometimes the P-interpretation of $\mathfrak{D}$-algebras may be called a 8-valued $\mathfrak{D}$-algebra because the cardinality of D is 8.

Since in our $\mathfrak{D}$-algebra only three operations $\oplus$, $\otimes$ and $\neg$ are introduced, a natural question that arises is: "can the three operations express all possible decision composition principles?" This question is equivalent to the question about the functional completeness of logical connectives for a corresponding many-valued logic system, that is: "do more connectives increase the expressiveness of a logic system?" The following theorem answers the question.

[1]Interestingly, if the "permit takes precedence" principle is adopted, the final decision is the same!

[2]XACML does not give a definition for this situation.

Definition 3. The operation set of an algebra is functionally complete if and only if any function in the algebra, given its truth table, is expressible by operations in the set.

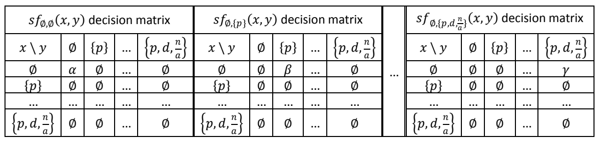
THEOREM 1. *$\{\otimes, \oplus, \neg\}$ is functionally complete.*

The proof of the functional completeness of $\{\otimes, \oplus, \neg\}$ is discussed in the next section.[3] Theorem 1 shows the importance of introducing $\otimes$ and axiom 7 in the $\mathfrak{D}$-algebra. The reader is reminded that $\{\neg, \oplus\}$ is functionally complete for a Boolean algebra. This is because $\otimes$ can be expressed using $\{\neg, \oplus\}$: $x \otimes y = \neg(\neg(\neg x \oplus y) \oplus \neg(x \oplus \neg y))$.

3.1 Computational Effectiveness

As mentioned in the introduction, the computational effectiveness of a $\mathfrak{D}$-algebra means that a tractable algorithm exists for efficiently constructing a formula corresponding to any given 2-D decision matrix. A decision matrix for an algebra formula is similar to a truth table for a logic formula. When used to express composition principles, a formula sometime may be a better choice than a decision matrix. For instance, if the number of decisions is very large and the relevant decision matrix is very sparse, a formula may be a better choice. Moreover, when decision formulas have been embedded in policies, a formula usually is a better choice than its corresponding decision matrix.

In this section, we propose a method for constructing a decision formula directly from an 8-valued $\mathfrak{D}$-algebra. The construction procedure, albeit not formal, can be considered as a proof of the first part (the functional completeness of $\{\otimes, \oplus, \neg\}$) of Theorem 1 because all construction steps are interpretation-independent. In other words, for any interpretation of the $\mathfrak{D}$-algebra, the approach for constructing the proof is the same.



$sf_{\emptyset,\emptyset}(x,y)$ decision matrix					$sf_{\emptyset,\{p\}}(x,y)$ decision matrix					...	$sf_{\emptyset,\{p,d,\frac{n}{a}\}}(x,y)$ decision matrix				
$x \setminus y$	$\emptyset$	$\{p\}$	...	$\{p,d,\frac{n}{a}\}$	$x \setminus y$	$\emptyset$	$\{p\}$	...	$\{p,d,\frac{n}{a}\}$		$x \setminus y$	$\emptyset$	$\{p\}$	...	$\{p,d,\frac{n}{a}\}$
$\emptyset$	α	$\emptyset$	...	$\emptyset$	$\emptyset$	$\emptyset$	β	...	$\emptyset$		$\emptyset$	$\emptyset$	$\emptyset$	...	γ
$\{p\}$	$\emptyset$	$\emptyset$	...	$\emptyset$	$\{p\}$	$\emptyset$	$\emptyset$	...	$\emptyset$		$\{p\}$	$\emptyset$	$\emptyset$	...	$\emptyset$
...	...	...	...	...	...	...	...	...	...		...	...	...	...	...
$\{p,d,\frac{n}{a}\}$	$\emptyset$	$\emptyset$	...	$\emptyset$	$\{p,d,\frac{n}{a}\}$	$\emptyset$	$\emptyset$	...	$\emptyset$		$\{p,d,\frac{n}{a}\}$	$\emptyset$	$\emptyset$	...	$\emptyset$

Figure 2: Decision Matrices for Subfunctions

Consider the matrix represented in Table 1. Each row and each column in the matrix is associated with a unique label from the set of decisions. For example, an entry (i, j) with the i_{th} row label and the j_{th} column label in such a matrix represents the decision taken according to the permit-overrides combination principle when the inputs are the i_{th} row label and the j_{th} column label. The idea for constructing the corresponding compositing function $f(x, y)$ is to construct 64 subfunctions $sf_{i,j}(x, y)$, one for each cell first, where i and j represent the row label and the column label of the cell, respectively. The composed decision of each subfunction $sf_{i,j}(x, y)$ is the decision in the cell when its input (x, y) corresponds to the cell position, i.e. when $i = x$ and $j = y$. Otherwise the composed decision is $\emptyset$. Figure 2 shows decision matrices for subfunction $sf_{\emptyset,\emptyset}(x, y)$, $sf_{\emptyset,\{p\}}(x, y)$, and $sf_{\emptyset,\{p,d,\frac{n}{a}\}}(x, y)$. In these matrices, only when the input (x, y) corresponds to the relevant position, the result is different from $\emptyset$. We denote such values with Greek letters such as α, β, or γ. If we can construct these 64 subfunctions, the final decision function is as follows:

$$f(x, y) = sf_{\emptyset,\emptyset}(x, y) \oplus sf_{\emptyset,\{p\}}(x, y) \oplus .... \oplus sf_{\{p,q,\frac{n}{a}\},\{p,q,\frac{n}{a}\}}(x, y)$$

Table 1: Permit-Overrides Decision Matrix

$x \setminus y$	$\emptyset$	$\{p\}$	$\{d\}$	$\{\frac{n}{a}\}$	$\{p, d\}$	$\{p, \frac{n}{a}\}$	$\{d, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$
$\emptyset$	$\emptyset$	$\{p\}$	$\{d\}$	$\{\frac{n}{a}\}$	$\{p, d\}$	$\{p, \frac{n}{a}\}$	$\{d, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$
$\{p\}$	$\{p\}$	$\{p\}$	$\{p\}$	$\{p\}$	$\{p\}$	$\{p\}$	$\{p\}$	$\{p\}$
$\{d\}$	$\{d\}$	$\{p\}$	$\{d\}$	$\{d\}$	$\{p, d\}$	$\{p, d\}$	$\{d\}$	$\{p, d\}$
$\{\frac{n}{a}\}$	$\{\frac{n}{a}\}$	$\{p\}$	$\{d\}$	$\{\frac{n}{a}\}$	$\{p, d\}$	$\{p, \frac{n}{a}\}$	$\{d, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$
$\{p, d\}$	$\{p, d\}$	$\{p\}$	$\{p, d\}$	$\{p, d\}$	$\{p, d\}$	$\{p, d\}$	$\{p, d\}$	$\{p, d\}$
$\{p, \frac{n}{a}\}$	$\{p, \frac{n}{a}\}$	$\{p\}$	$\{p, d\}$	$\{p, \frac{n}{a}\}$	$\{p, d\}$	$\{p, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$
$\{d, \frac{n}{a}\}$	$\{d, \frac{n}{a}\}$	$\{p\}$	$\{d\}$	$\{d, \frac{n}{a}\}$	$\{p, d\}$	$\{p, d, \frac{n}{a}\}$	$\{d, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$
$\{p, d, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$	$\{p\}$	$\{p, d\}$	$\{p, d, \frac{n}{a}\}$	$\{p, d\}$	$\{p, d, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$	$\{p, d, \frac{n}{a}\}$

Now the problem becomes how to construct each subfunction. Assume that the decision of a cell (ϕ, φ) is ψ, we can construct a corresponding subfunction as follows:

$$sf_{\phi,\varphi}(x, y) = \psi \odot (\phi \otimes x) \odot (\varphi \otimes y)$$

$(\phi \otimes x) \odot (\varphi \otimes y)$ evaluates to $\neg\emptyset$ if and only if $x = \phi$ and $y = \varphi$ and to be $\emptyset$ otherwise. Therefore, $\psi \odot (\phi \otimes x) \odot (\varphi \otimes y)$ evaluates to ψ if and only if $x = \phi$ and $y = \varphi$ and $\emptyset$ otherwise. For instance, the subfunction of cell $(\emptyset, \{p\})$ in Figure 2 $sf_{\emptyset,\{p\}}(x, y) = \beta \odot (\emptyset \otimes x) \odot (\{p\} \otimes y)$. It is easy to see that the complexity of the construction only depends on the number of cells in a decision matrix, that is, $O(n^2)$ assuming n to be the number of decisions.

It should be noted that in practice we construct the composition function by considering the patterns of the decision matrix. Usually we can obtain a much terser function than the lengthy function built by the mechanical steps (see Section 4.3). The mechanical steps are useful for some worst situation, for example when the distribution of decisions in the matrix is totally random, and for automatically computing composition functions (see Section 7.1).

Up to now we only discuss composition functions based on 2-D decision matrices. The same idea can be applied to higher dimensional decision matrices as well, e.g. 3-D decision matrices and even m-D decision matrices. The number of arguments of corresponding functions are the dimensions of relevant matrices. It is straightforward for us to show that the complexity of such a function construction is $O(n^m)$ assuming n to be the number of decisions and m to be the number of dimensions.

2-D decision matrices are suitable for some general decision combination principles, e.g. permit-overrides, which can be realized by pair-wisely combining decisions. Higher dimensional matrices might be useful for some special cases. For instance, a special combination principle may require that each decision combination must include 3 rules simultaneously and the final decision is "permit" if and only if the first rule and the second rule evaluate to "permit" and the third rule evaluates to "deny".

In this paper, we only focus on composition functions based on 2-D decision matrices because higher dimensional decision matrices do not provide obvious benefits. First, popular combination principles like all standard combination algorithms in XACML and majority voting can be realized by pair-wisely combining decisions. We actually cannot

[3]The proof that $\{\oplus, \neg\}$ is not functionally complete is discussed in the appendix.

find a popular combination principle that cannot be realized pair-wisely. Second, a pair-wise combination might be the most natural and popular way for human beings to aggregate decisions and can help avoid some tricky issues. For instance, if a decision combination principle must include 3 rules simultaneously like the example mentioned above, we have to define some additional rule in order to use this principle to combine decisions from 4 rules.

4. CASE STUDY: XACML

We now show the application of the $\mathfrak{D}$-algebra to XACML [4] based on a P-interpretation. We discuss how we can apply a P-interpretation to accurately express a rule evaluation truth table and how decision matrices and composition functions are used to address the problems in standard policy combining algorithms. The advantage of the $\mathfrak{D}$-algebra becomes clear especially when handling the composition of non-deterministic decisions.

4.1 Evaluation of Rule, Policy, and Policy Set

To answer a request w.r.t. a XACML policy set ps, all or some of applicable inner policy sets, policies, and rules contained in ps first evaluate to some intermediate decisions. Those intermediate decisions are then combined according to the policy and rule combining algorithms defined in ps to return the final decision. In what follows, we first present the XACML evaluation truth tables for rules and policies, then identify issues in these tables using the P-interpretation, and finally propose a revised truth table based on the P-interpretation. XACML specifies that rule evaluation involves a separate evaluation of the rule's target and condition; the rule truth table used by XACML is shown in Table 2.

Table 2: Rule Truth Table

Target	Condition	Rule Value
Match	True	Effect
Match	False	Not Applicable
Match	Indeterminate	Indeterminate
No-match	Don't care	Not Applicable
Indeterminate	Don't care	Indeterminate

A target defines the set of requests to which the rule is intended to apply in the form of a logical expression on attributes in the request. The condition element may further refine the applicability established by the target. "Don't Care" in the table means that the condition is simply ignored and thus is not evaluated. There are two missing cases in Table 2. The cases of an empty target and an empty condition in a rule are allowed by the XACML standard but the truth table in the XACML standard does not show how to process an empty target and an empty condition. The XACML standard says that if there is no target in a rule, the value of the empty target should be the same as the value of the target of the policy containing the rule. If the condition is missing, its evaluation is true based on the semantics of the XACML language model. There are two problems in the truth table. First, even if the target value is "indeterminate", the rule value need not be "indeterminate". If some attribute values in the target cannot be retrieved during evaluation, the target evaluates to "indeterminate". However, if the value of the condition is false, the rule value indeed is "not applicable" because whatever the target value is, either match or no-match, the rule value is always "not applicable" in this situation. Inappropriate "indeterminate" here may result in a wrong final composed decision. Another problem is that "indeterminate" represents *too many* different values which is the key reason behind the inappropriate decision result from the policy combining algorithms (discussed in Section 4.2).

The revised truth table based on the P-interpretation is shown in Table 3. Values of a target and a condition are two P-interpretations represented by deterministic decision sets $\{m, n\}$ and $\{t, f\}$ respectively, and an effect is a Boolean algebra $\{p, d\}$[5]. As discussed before, the rule value, that is, its decision, is a P-interpretation on a deterministic decision set $\{p, d, \frac{n}{a}\}$. $m, n, t, f, p, d, \frac{n}{a}$ represent "match", "no-match", "true", "false", "permit", "deny", and "not applicable" respectively. "$-$" in Table 3 denotes "don't care". "," in Table 3 denotes "OR" relation between different condition evaluation results.

Table 3: Revised Rule Truth Table

Target	Condition	Effect	Rule Value
$\{n\}$	$-$	$-$	$\{\frac{n}{a}\}$
$-$	$\{f\}$	$-$	$\{\frac{n}{a}\}$
$\{m\}$	$\emptyset, \{t\}$	$\{p\}$	$\{p\}$
$\{m\}$	$\emptyset, \{t\}$	$\{d\}$	$\{d\}$
$\{m\}$	$\{t, f\}$	$\{d\}$	$\{d, \frac{n}{a}\}$
$\{m, n\}$	$\emptyset, \{t\}, \{t, f\}$	$\{d\}$	$\{d, \frac{n}{a}\}$
$\{m\}$	$\{t, f\}$	$\{p\}$	$\{p, \frac{n}{a}\}$
$\{m, n\}$	$\emptyset, \{t\}, \{t, f\}$	$\{p\}$	$\{p, \frac{n}{a}\}$

The semantics of Table 3 is easily understood. The case in which a target value equals to $\emptyset$ is not included in Table 3 because in this situation the target value of its containing policy has already been evaluated and the PDP can simply follow the corresponding entry to evaluate the rule[6].

Compared to Table 2, Table 3 is more precise and different "indeterminate" values are clearly distinguished. Careful readers may notice that some rule values are missing from Table 3, e.g. $\{p, d\}$, $\{p, d, \frac{n}{a}\}$. These values cannot be directly generated by the rule evaluation but can be generated by rule composition functions.

XACML provides a policy truth table (Table 4). It is unnecessary to list the 3 cases which all refer to the case of the target value equal to "match". The second case directly specifies the policy value to be "not applicable" by skipping the rule combining algorithm. This is not a good design. On the one hand, to verify all rule values, a PDP has to evaluate each rule, which is not much different from running a rule-combining algorithm. On the other hand, the table prevents a non-standard combining algorithm from adopting other combining strategies, e.g., "deny". Another problem is that a policy with an "indeterminate" target value does not necessarily evaluate to "indeterminate", for instance, when all rule conditions evaluate to "false", the policy should re-

[4]Issues discussed here apply to both XACML 2.0 and XACML 3.0 WD 6.

[5]We assume that effects of rules cannot be empty or some wrong values and are always retrievable from a PDP.

[6]XACML does not clearly define the relation between the target of a containing policies and targets of inner rules. For simplicity, we assume in this paper that the scope of a policy target contains the union of scopes of all inner rules.

turn "not applicable" instead of "indeterminate". A revised policy truth table is shown in Table 5.

Table 4: Policy Truth Table

Target	Rule Value	Policy Value
Match	At least one rule value is its Effect	Specified by the rule-combining algorithm
Match	All rule values are "Not Applicable"	Not Applicable
Match	At least one rule value is "Indeterminate"	Specified by the rule -combining algorithm
No-match	Don't care	Not Applicable
Indeterminate	Don't care	Indeterminate

Table 5: Revised Policy Truth Table

Target	Policy Value
$\{m\}, \{m, n\}$	Specified by the rule-combining algorithm
$\{n\}$	$\{\frac{n}{a}\}$

XACML also provides a policy set truth table which is the same as Table 4 except that the term "rule" is replaced by the term "policy". Both cases, revised according to our approach, are covered in Table 5.

4.2 Rule/Policy Combining Algorithms

XACML defines five standard rule combining algorithms and six standard policy combining algorithms. However, due to the overloaded meaning of "indeterminate", all policy combining algorithms, including deny-overrides, permit-overrides, first-applicable, and only-one-applicable, can generate inappropriate decisions. The deny-overrides rule combining algorithm and permit-overrides rule combining algorithm behave as expected because different "indeterminate"s are distinguishable at rule level. Unfortunately, "indeterminate"s becomes indistinguishable at policy level.

As mentioned in the previous section, the key reason behind inappropriate decisions is that the "indeterminate" decision adopted in XACML represents *too many* different non-deterministic decisions. Clearly, "indeterminate" $\{p, \frac{n}{a}\}$ is different from "indeterminate" $\{d, \frac{n}{a}\}$. Even though at rule level the combining algorithms can distinguish between these two different "indeterminate"s by using local information, the local information is lost at policy level because of the vague "indeterminate". The problem of the deny-overrides algorithm is shown in the introduction. The reason behind this problem is that a policy combining algorithm cannot distinguish indeterminate $\{p, \frac{n}{a}\}$ from indeterminate $\{d, \frac{n}{a}\}$ and has to adopt a conservative approach by assuming all "indeterminate" can possibly take the d ("deny") value which is not sufficiently precise and leads to inappropriate decisions. Because the problem of permit-overrides is similar to that of the deny-overrides algorithm, its discussion is omitted.

The problem of first-applicable algorithm is that when it encounters an "indeterminate" the algorithm immediately returns "indeterminate". However, if the first "indeterminate" is $\{p, \frac{n}{a}\}$ and a next rule returns $\{p\}$, it is reasonable that "permit" instead of "indeterminate" should be returned in this case. If no error happens and the first "indeterminate" is actually $\{p\}$, then "permit" should be returned. If no error happens and the first "indeterminate" is actually $\{\frac{n}{a}\}$, then "permit" should be returned as well.

Even without considering the inappropriate representation of "indeterminate", the only-one applicable policy combining algorithm does not have a good design. The main problem is that the algorithm verifies whether a policy is applicable or not only by the target of the policy: "In the entire set of policies in the policy set, if no policy is considered applicable by virtue of its target, then the result of the policy-composition algorithm SHALL be *not applicable*. If more than one policy is considered applicable by virtue of its target, then the result of the policy-composition algorithm SHALL be *Indeterminate*." [18]

There is one potential problem with such a target-based strategy. Assume that policy p_1 contains rules r_1, r_2 and r_3; suppose that those rules have the scopes shown in Figure 3 (such scopes are allowed by XACML). Let q denote an access request. Obviously, based on the algorithm, p_1 is applicable to q. However, no rule in p_1 is really applicable to q. Moreover, even if there is a rule target matching q, if the rule condition is false, the rule is still not applicable to q. Thus the correct answer of p_1 in these two cases should be "not applicable" instead of "applicable". If there is a policy p_2 following p_1 which evaluates to "permit", the composition decision from p_1 and p_2 is "indeterminate" based on the XACML only-one applicable algorithm but should be "permit" based on our analysis. The key observation is that a policy target can only be used to exclude the not applicable case (if a policy target evaluates to "no match") and cannot be used to identify any other cases. In order to obtain an appropriate answer, the algorithm should look at each inner rule in the policy: a policy is applicable to a request if and only if there exists at least one rule in the policy applicable to the request. The XACML version certainly has better efficiency, but we believe that for some situation users may want to get a more precise answer.

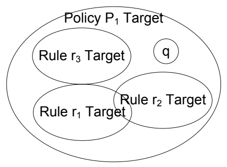


Figure 3: Target Scope

Another problem of the algorithm appears in the return value when two or more applicable policies exist. The algorithm returns "indeterminate". In this case a PDP cannot distinguish a policy set that returns a regular "indeterminate" decision such as $\{p, \frac{n}{a}\}$ from a policy set that returns an "indeterminate" decision due to the violation of a composition principle that is two or more policies in the policy set are applicable. These two "indeterminate" values really have very different semantics. To precisely capture the subtle difference, we need a new deterministic decision "error" in the 8-valued P-interpretation, which in turn becomes a 16-valued P-interpretation [7]. One possible effect of the "error" decision during rule and policy composition is that the "error" decision overrides any other decisions and propagates up to the highest level PDP. The behavior of the "'error" decision is similar to the exception concept in modern programming languages. In what follows, we stick to the 8-valued

[7]Due to space reason we do not adopt a 16-valued P-interpretation. The structure is very similar to the 8-valued interpretation.

P-interpretation.

4.3 Standard Composition Functions

A systematic solution, based on the P-interpretation, is presented for each standard combining algorithm. Instead of writing pseudo algorithms, we construct composition functions. The benefits of this approach are discussed in Section 7. For space reasons, we only use the permit-overrides as an example to show the procedure for constructing a composition function. For other composition algorithms, we only present the composition function without giving a detailed discussion.

The permit-overrides principle specifies the following decision precedence (order) for each deterministic decision: $\{p\} > \{d\} > \{\frac{n}{a}\}$. If any two of these deterministic decisions have to be combined, the stronger one will replace the weaker one. For non-deterministic decisions, all possible conditions should be considered. For instance: the composition result of decision $\{p, \frac{n}{a}\}$ and decision $\{d, \frac{n}{a}\}$ is $\{p, d, \frac{n}{a}\}$. Based on this discussion, it is straightforward to generate the decision matrix (see Table 1) for the permit-overrides principle.

By analyzing the decision matrix, we can find that the composition function $f_{po}(x, y)$ of permit-overrides principle is similar to that of the $\oplus$ operation except for two cases:

- If either x or y is $\{p\}$, we must subtract by using operation $\ominus$ the set $\{d, \frac{n}{a}\}$ from the final decision set because $\{p\}$ is the "strongest" decision.
- If neither x nor y contains $\{\frac{n}{a}\}$, we must subtract the set $\{\frac{n}{a}\}$ from the final decision set because $\{\frac{n}{a}\}$ is the "weakest" decision.
  - However, there is an exception of this case. If either x or y is $\emptyset$, we should not subtract the set $\{\frac{n}{a}\}$ from the final result because no decision is even weaker than the "weakest" decision.

The composition function of permit-overrides (f_{po}) is thus as follows:

$$\begin{aligned} f_{po}(x,y) &= (x \oplus y) \\ &\ominus(((x \otimes \{p\}) \oplus (y \otimes \{p\})) \odot \{d, \frac{n}{a}\}) \\ &\ominus(\neg((x \odot y) \otimes \{\frac{n}{a}\}) \odot \{\frac{n}{a}\} \odot \neg((x \otimes \emptyset) \oplus (y \otimes \emptyset))) \end{aligned}$$

Because the difference between permit-overrides and deny-overrides is a permutation of "permit" and "deny", the composition function f_{do} of deny-overrides is as follows:

$$\begin{aligned} f_{do}(x,y) &= (x \oplus y) \\ &\ominus(((x \otimes \{d\}) \oplus (y \otimes \{d\})) \odot \{p, \frac{n}{a}\}) \\ &\ominus(\neg((x \odot y) \otimes \{\frac{n}{a}\}) \odot \{\frac{n}{a}\} \odot \neg((x \otimes \emptyset) \oplus (y \otimes \emptyset))) \end{aligned}$$

Because of the existence of non-deterministic decisions, the first-applicable composition function f_{fa} actually is much more complicated than what it may look like. Two special values that may be generated intermediately are $\{p, d\}$, e.g., $f_{fa}(\{p, \frac{n}{a}\}, \{d\})$, and $\{p, d, \frac{n}{a}\}$, e.g. $f_{fa}(\{p, \frac{n}{a}\}, \{d, \frac{n}{a}\})$. $\{p, d\}$ and $\{p, d, \frac{n}{a}\}$ can only appear as left operands (an evaluation precedence must strictly follow the order of each rule in a policy) and are similar to $\{p\}$ and $\{d\}$: they simply override any possible right operands. Indeed, f_{fa} looks daunting and is really a good example to show that a decision matrix sometime is a better way to go about the construction of composition functions.

$$\begin{aligned} f_{fa}(x,y) &= (x \odot (x \otimes y)) \oplus (y \odot (x \otimes \{\frac{n}{a}\})) \oplus (x \odot (y \otimes \{\frac{n}{a}\})) \\ &\oplus(x \odot (x \otimes \{p\})) \oplus (x \odot (x \otimes \{d\})) \\ &\oplus(x \odot (x \otimes \{p, d\})) \oplus (x \odot (x \otimes \{p, d, \frac{n}{a}\})) \\ &\oplus(\{p\} \odot (x \otimes \{p, \frac{n}{a}\}) \odot (y \otimes \{p\})) \\ &\oplus(\{p, d\} \odot (x \otimes \{p, \frac{n}{a}\}) \odot ((y \otimes \{d\}) \oplus (y \otimes \{d, \frac{n}{a}\}))) \\ &\oplus(\{p, d, \frac{n}{a}\} \odot (x \otimes \{p, \frac{n}{a}\}) \odot ((y \otimes \{d, \frac{n}{a}\})) \\ &\oplus(\{d\} \odot (x \otimes \{d, \frac{n}{a}\}) \odot (y \otimes \{d\})) \\ &\oplus(\{p, d\} \odot (x \otimes \{d, \frac{n}{a}\}) \odot ((y \otimes \{p\}) \oplus (y \otimes \{p, \frac{n}{a}\}))) \\ &\oplus(\{p, d, \frac{n}{a}\} \odot (x \otimes \{d, \frac{n}{a}\}) \odot ((y \otimes \{p, \frac{n}{a}\})) \end{aligned}$$

The only-one applicable principle requires that one of the two operands must evaluate to "not applicable". As mentioned earlier, the current 8-valued P-interpretation cannot precisely capture such a situation if there are two applicable policies. For space reasons, we will not expand our 8-valued P-interpretation to a 16-valued one in this paper. By contrast, we introduce a "shortcut" version to handle this situation. If we reasonably assume that each policy must contain at least one rule, then we may assign decision value $\emptyset$ to this situation. If each policy must contain at least one rule, no policy can evaluate to $\emptyset$ and thus the PDP becomes aware that there is a principle violation that there are two applicable policies. Noticing that $\emptyset$ has the highest precedence (representing an error status) and the composition must be $\emptyset$ once both x and y are not $\{\frac{n}{a}\}$, we have the following composition function for the only-one applicable principle.

$$f_{oo}(x,y) = (x \odot (y \otimes \{\frac{n}{a}\})) \oplus (y \odot (x \otimes \{\frac{n}{a}\}))$$

One reason to introduce ordered-deny-overrides and order-permit-overrides is to guarantee a deterministic result when obligations are involved. The ordered versions of deny-overrides and permit-overrides are not new composition operations; they are just guidelines on the order according to which a PDP applies composition operations to the policies. In fact, the pseudo algorithms provided by the XACML standard are for the ordered versions, even if their names do not indicate that they are ordered.

4.4 User-defined Composition Function

XACML supports user-defined rule/policy combining algorithms. It is straightforward to construct a user-defined composition function using P-interpretation because the procedure for standard composition functions is not different from that for a user-defined composition function. For instance, a strong AND principle requires that given a set of rules, only if all rules are evaluated to the same decision, the final decision is such a decision, otherwise the final decision is an error (a violation of the principle). If we use $\emptyset$ to represent the error decision and do not allow an empty policy, the following concise composition function implements the strong AND principle.

$$f_{sa}(x,y) = x \odot (x \otimes y)$$

It is usually the case that a composition function built by carefully observing the patterns in the final decision matrix or investigating the "strength" relation between deterministic decisions can be terse and elegant compared to functions built by mechanical steps. Since a combining principle is often described by some special decision patterns or some "strength" relations, a concise composition function can usually be built without much effort. However, a clear and simple description of a combination principle may not always generate a terse function such as first-applicable. In this situation, we may need a formal language to help users to generate functions or decision matrices (detailed in Section 7.1).

5. COMPLEX DECISION EXPRESSIONS

Unlike XACML which can only specify a rule combining algorithm for a total rule set, our $\mathfrak{D}$-algebra can model a much more fine-grained manipulation on each rule in a rule set by a decision expressions, e.g. $(r_1 \otimes \neg(r_2 \ominus r_3) \oplus (r_4 \odot \neg r_5))$. In practice, specific applications or contexts may require a special composition methodology that is different from the application of the same composition function to all rules. For instance, the Chief Security and Privacy Officer (CSPO) of a company may define a binding set of base regulations for all departments in the company. These regulations can in turn be refined by the various departments [19]. One way to address this requirement is to let the CSPO define some applicable (mandatory) policies to be followed by all the departments. However, these mandatory policies cannot authorize access requests themselves because they are not sufficiently fine-grained. Each department can then define its own fine-grained terminal policies. The decision for a request is then obtained by composing the decisions from all applicable policies with at least one decision from the terminal policies in one department. In other words, if all applicable mandatory policies allow the access request and at least one terminal policy allows the request, the request is authorized. The $\mathfrak{D}$-algebra is flexible enough to construct a decision expression to express such requirement:

$$p_{m1} \wedge p_{m2} \wedge ... \wedge p_{mn} \wedge ((p_{td11} * p_{td12} ...) \vee (p_{td21} * p_{td22} ...) \vee ...)$$

where $\wedge$ is the infix version of the deny-overrides function, $\vee$ is the infix version of the permit-override function, $*$ represents any valid operation or composition function, p_{mi} represents mandatory policies, and p_{tdij} represents the j-th terminal policy in the i-th department. The expression specifies that in order to obtain a final "permit" decision all mandatory policies must evaluate to "permit" or "not applicable" and at least one departmental decision is "permit". Another good example is represented by positive norms and negative norms [4], the discussion of which is omitted for space reasons. It should be noted that XACML can support the CSPO scenario through the use of multi-level policy sets. However, the approach is inefficient and cannot be applied at a rule level in XACML. The ability to manipulate decision composition at a rule level enables departmental policy authors to write more flexible policies without assistance by a CSPO and increases the modularity of policy specifications.

6. MAJORITY VOTING

Majority voting is a popular decision making mechanism, including simple majority voting, absolute majority voting, and super majority voting. The simple majority voting is a form of voting in which, given two options, the option receiving more votes than the other wins. The absolute majority voting usually requires that more than half of all the members of a group (including those absent and those present but not voting) must vote in favour of a decision in order for it to be passed. A super majority or a qualified majority is a requirement for a decision to obtain a specified level or type of support which exceeds a simple majority in order to have effect, e.g. two-thirds majority.

A notable feature of all aforementioned non-majority voting combining principles is that they can be handled without memorizing decisions from the processed rules. In other words if we have multiple rules we can handle two of them first and process the third with the result from the first two, and so on. During this procedure, results from rules that are overwritten by other rules are discarded because these discarded results have no effect on the final result. The final result of majority voting, however, requires that the final result must take into account the decisions of each calculation step. In other words decisions that result from all rules are to be memorized because all of them may impact the final result.

6.1 Majority Voting and Rational Number Interpretation

The simple majority voting has the following two properties. First, it assigns the same unambiguous value to every logically possible list of individual decisions. In the context of rule composition, each rule has the same weight on a final decision, referred to a truth degree. Second, the final decision depends on the accumulation of truth degrees on individual decisions. In other words, all votes matter and we need to memorize them during counting. Thus we need an operation able to accumulate evaluation results[8].

The rational number interpretation (R-interpretation) for $\mathfrak{D}$-algebra is the right tool that satisfies these two properties. The rational number interpretation is actually the standard semantics of Łukasiewicz logics $Ł_m$.

- D_m is represented by some finite set $\{k/(m-1) | 0 \leq k \leq m-1\}$, where $k, m \in \mathbb{N}$ and $m > 1$.
- 0 is represented by 0.
- $\neg x$ is represented by 1 - x.
- $x \oplus y$ is represented by $\min\{1, x+y\}$ where $x, y \in D_m$.
- $\otimes$ is defined by axiom 7.

$\oplus$ here represents strong disjunction in $Ł_m$ while standard $Ł_m$ also supports weak disjunction ($\vee$), that is $x \vee y = \max\{x, y\}$, and implication ($\rightarrow$), that is $x \rightarrow y = \min\{1, 1 - x + y\}$. Due to the functional completeness of $\mathfrak{D}$-algebra, these two operations are not necessary because they can be represented by using only the $\neg$ and $\oplus$ operations. For instance, $x \rightarrow y = \neg x \oplus y = \min\{1, 1 - x + y\}$.

Now we can see that two important properties of the simple majority voting can be satisfied by such a R-interpretation. The same unambiguous value is assigned to each yes/no voting. Given n Yes/No/Abstain voting, each vote has the same truth degree of $1/n$ for either "Yes", "Abstain", or "No". The required accumulation operation is provided by the strong disjunction $\oplus$. The final decision depends on the comparison between the final truth degree of "Yes" votes (V_y) and the truth degree of "No" votes (V_n), e.g. if $V_y > V_n$, then "Yes". It is trivial to implement comparison like $>$ by using implication $\rightarrow$ operation.

[8]Such operation seems trival but among many popular many-valued logics only Łukasiewicz logics $Ł_m$ supports it.

The difference between a strong majority voting and a simple majority voting is the last step. The final decision in a strong majority voting depends on the comparison between the final truth degrees of different preferences with a constant truth value 0.5, e.g., if $V_y > 0.5$, then "Yes". Similarly, to handle super majority voting like three-fifths majority or two-thirds majority, we just need different thresholds (constant truth values) to make a final decision.

We are investigating an interpretation that combines both the powerset interpretation and the rational number interpretation and prepare to discuss it in our future work.

6.2 Simple Majority Voting on XACML Policies/Rules

As a popular judgment aggregation method, the simple majority voting can be applied to combining XACML rules as well. However, the distinguishing feature of XACML rule composition is that we have multiple non-deterministic votes other than deterministic Yes(permit) / No(deny) / Abstain(not applicable) votes. Later, we show that non-deterministic intermediate results make tricky final decisions. For space reason, we only discuss the simple majority voting at rule level. The idea, however, can be applied to policy level and to other majority voting such as absolute majority voting or super majority voting.

Since results from the evaluation of XACML rules can only be one of $\{p\}$, $\{d\}$, $\{\frac{n}{a}\}$, $\{p, \frac{n}{a}\}$, $\{d, \frac{n}{a}\}$, we use a 5-tuple $(v_0, v_1, v_2, v_3, v_4)$ to represent a truth degree of each rule, where v_0, v_1, v_2, v_3, v_4 represent the truth value of $\{p\}$, $\{d\}$, $\{\frac{n}{a}\}$, $\{p, \frac{n}{a}\}$, or $\{d, \frac{n}{a}\}$ respectively. For instance, consider 10 rules. Given an access request, if rule r_1 evaluates to permit $\{p\}$, the truth degree of r_1 is $(1/10, 0, 0, 0, 0)$. If rule r_2 evaluates to $\{d, \frac{n}{a}\}$, its truth degree is $(0, 0, 0, 0, 1/10)$.

An operation between two truth tuples is extended by performing the operation on the elements of these two truth tuples, e.g. $(v_0, v_1, v_2, v_3, v_4) \oplus (v_0', v_1', v_2', v_3', v_4') = (v_0 \oplus v_0', v_1 \oplus v_1', v_2 \oplus v_2', v_3 \oplus v_3', v_4 \oplus v_4')$. To represent the simply majority voting on XACML rules, all rules are connected by strong disjunction $\oplus$. For instance, $r_1 \oplus r_2 = (1/10, 0, 0, 0, 1/10)$. Since such an extension of operations still follows all axioms of Definition 1, it is just an extended R-interpretation.

One way to make a final decision for either "permit" or "deny" is simply comparing v_0 with v_1. It, however, may result in some problem. For example, assume the final truth tuple is $(4/10, 3/10, 1/10, 0, 2/10)$, there are two possibly applicable deny rules evaluated to non-deterministic values $\{d, \frac{n}{a}\}$ due to reasons like irretrievable attribute values. What if they evaluate to "deny" when all relevant attribute values can be retrieved? In this case, the final decision should be "deny" rather than "permit". Thus, the final decision must also take into account truth values from non-deterministic votes. A possible way to make a final decision is as follows:

- A final decision is "permit" if and only if $v_0 > v_1 + v_4$.
- A final decision is "deny" if and only if $v_1 > v_0 + v_3$.
- Otherwise a final decision is some non-deterministic value depending on the votes on non-deterministic decisions, e.g. $\{p, \frac{n}{a}\}$.

The truth value of v_3, "not applicable", is useful in some special situation. For instance, in a sensitive context, a rule combining principle may require that non-deterministic decisions be disallowed and an effective decision be made only if at least 80% rules are applicable. In this situation $v_3 < 0.2$ is a necessary condition to make a final decision. Indeed the possibility of manipulating truth values of different elements in a final truth tuple makes it possible to satisfy a quite large range of decision aggregation mechanisms.

Note that our approach makes it possible to use majority voting on some level of XACML and use different composition principles at other level, e.g., majority voting at rule level and deny-override at policy level or vice versa.

6.3 Generalized Majority Voting

A generalized majority voting may introduce weights on different rules. Different weights on different rules are essentially represented by different truth degree on their evaluation results. Different choices of weights may need different interpretations. For instance, assume that three rules r_1, r_2 and r_3 have weights 1, 2, and 3 respectively. The current R-interpretation can handle such a case well: the truth degrees of r_1, r_2 and r_3 are 1/6, 2/6, and 3/6. Basically the truth degree of r_x is "weight$_x$ / sum of weights". However, if real number weights are used, we need a real number interpretation of $\mathfrak{D}$-algebra that is similar to R-interpretation.

7. PRACTICAL CONSIDERATIONS

When we adopt $\mathfrak{D}$-algebra as a formal tool for the aggregation of access control decisions, we may encounter some practical issues. For instance, we may want to improve decision combination efficiency. Since in our approach different rule/policy composition principles are represented by algebraic expressions, various mature evaluation strategies from programming languages can be applied directly to improve evaluation efficiency. For instance, short-circuit evaluation can be used in first-applicable, permit-override, deny-override, only-one-applicable, and majority voting etc. A detailed discussion of possible optimizations is omitted due to space reason. In this section, we focus on two other issues: methods to help policy authors write composition functions and an algebra-based PDP design.

7.1 Constructing Composition Functions

An access control policy language designed based on a $\mathfrak{D}$-algebra should be accompanied by a composition function library, which consists of functionally complete operations, standard composition functions, and user-defined composition functions. To write a correct policy, policy authors have to choose an appropriate composition function. There will be situations in which standard functions, such as permit-overrides, might not be sufficient. Therefore we need methodologies to assist policy authors in constructing a user-defined composition function correctly and efficiently. We thus suggest three such different methodologies.

- Experienced policy authors can write composition functions directly by using the functionally complete operation set or standard composition functions. Usually it is not hard to obtain a terse function if there exists some pattern in the decision requirements.
- If the decision matrix of the user-defined function has a reasonable size, it is usually better for policy authors to specify a decision matrix rather than directly writing the function. By the mechanical step introduced

in Section 3.1, a corresponding function can be constructed automatically.

- To help constructing a composition function for a large-size decision set, a Decision Specification Language (DSL), currently under development, allows policy authors to specify decision composition rules. Based on these rules, a decision matrix can be automatically constructed and a composition function can be in turn defined based on this matrix.

DSL is a declarative language that allows one to specify decisions. A decision specification consists of a set of DSL statements which are order-insensitive. The current version of the language includes the following statements.

- declare statement, e.g., choose an interpretation and define the semantics of basic operations.
- override statement, e.g., if decision x overrides decision y, the composed decision is x.
- contradiction statement, e.g., if decision x contradicts decision y, the final decision is an error decision.
- containment/order statement (can be conditional), e.g., if decision x contains decision y, the final relation is z.

Two simple statements that describe the permit-overrides principle are presented as follows:

```
<statement type = "override">
    <source>permit</source>
    <destination>deny</destination>
<\statement>
<statement type = "override">
    <source>deny</source>
    <destination>not applicable</destination>
<\statement>
```

7.2 An Algebra-based PDP Design

XACML allows a party to define its own combining algorithm. Since the algorithm is not a standard one, in a distributed but collaborative environment the party might not be able to share this policy with other parties since not all PDPs might understand the algorithm's identifier in the file. One possibility is to keep standardizing more algorithms and adding more algorithm identifiers, but by using the $\mathfrak{D}$-algebra we can standardize the mechanism to describe the combining algorithms once and eliminate the need of standardizing new algorithms. The key points of our design are as follows (See Figure 4):

- We can revise the XACML policy structure by replacing the rule combining algorithm identifier with the followings.
  - Either a function id to compose decisions from the rules in the policy or a composition expression constructed by composition functions applied on all rules in the policy. One of them must be specified. It should be noted that the composition expression is provided only for experienced policy authors.
  - If a user-defined function is provided, it is initially either specified as DSL statements or defined by a decision matrix. In both cases, it will be compiled into an algebraic expression before being stored in a policy repository.
- A PDP consists of a target and rule evaluator and a decision composer. The target and rule evaluator is the same as that used in an XACML PDP. The decision composer replaces the software modules that represent the hard-coded rule-combining algorithms in an XACML PDP. The composer only needs to "understand" three basic operations, that is, $\{\neg, \oplus, \otimes\}$.

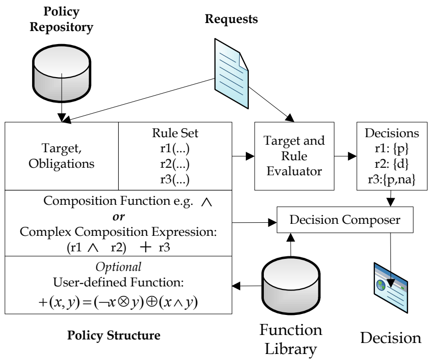


Figure 4: An Algebra-based PDP Design

There are several advantages in such a design. Since standard functions are built by these three simple and elegant operations, the composer only needs to know these three operations and how to evaluate an expression based on these operations. The implementation of the composer might be more complex than that of one combining algorithm; however the implementation is much simpler than that of several standard combining algorithms. Accordingly the correctness of its implementation is easier to verify than that of the implementation of complex standard combing algorithms. More importantly, the capability of a composer is much more flexible than the capability of hand-coded combining algorithms. When a new standard composition function has to be introduced and used, there is no need to modify the code of the composer. What we need is just to specify a new composition function definition in policies. We also need not to worry that an updated PDP or a third party PDP cannot understand new standard composition functions or user-defined functions because all these new functions will be described in terms of well understood standard operations. Last but not least, unlike standard algorithms provided by XACML, the algebra based composer can help reduce errors during policy composition. Deterministic and complete obligations, once required, can be guaranteed as well. The correctness of algebra-based composition is provable.

Some careful reader may argue that the efficiency of evaluating an expression during policy enforcement can be a problem for time crucial tasks. However, efficiency when dealing with a new composition function can be achieved by immediately building a decision matrix based on the function. Rule/policy composition now becomes a table lookup operation which usually is much faster than the execution of standard rule/policy combining algorithms.

8. RELATED WORK

The notion of many-valued logic (MVL) as a research topic was introduced by the Polish logician and philosopher Łukasiewicz [14] in 1920. The 1950s saw a sudden boom of invaluable research results on MVL. An analytical characterization of the class of truth degree functions definable in the infinite valued propositional Łukasiewicz system, $Ł_\infty$, was proposed by McNaughton [17]. The first proof of completeness on $Ł_\infty$ [17] was presented by Rose et al. [20]. Chang [9] devised the algebraic semantics, MV-algebras, of the $Ł_\infty$, based on which an elegant proof of completeness theorem was then given [10]. The first proof of functional completeness for $Ł_m$ $(m > 2)$ using rational number semantics that added a unary operation $\mathbf{T}$ to the logic was given by Rosser et al. [21]. Several issues on the functional completeness of $Ł_m$ $(m > 2)$ were later investigated by Jobe [13]. The properties of Sheffer stroke functions for $Ł_3$ and $Ł_\infty$ were investigated by Martin [16] and Graham [12], respectively.

In this paper, we provide another proof of functional completeness for $Ł_\infty$ by adding a binary operation $\otimes$. The proof is based on MV-algebras. Compared to the proof by Rosser et al. [21], the value of our proof is that the new binary operation $\otimes$ and the P-interpretations are more natural for handling policy decision problems by treating them fairly. More importantly, our proof itself provides the basis for the computational effectiveness of $\mathfrak{D}$-algebra. It seems to be known that $Ł_m$ $(m > 2)$ is functionally incomplete but we were not able to find a formal proof. Therefore, a formal proof regarding MV-algebra is given in this paper as well. Our P-interpretation is closely related to Fitting's work [11] on the generalization of Kleene's 3-value Logic.

Using algebras as formal tools to formalize policy composition is not a new idea. Bonatti et al. [6] formalized policies as customized Horn clauses and introduced six operators, namely addition, subtraction, closure, scoping restriction, overriding, and template, to describe policy expressions. Because some operators are closely intertwined with predicates that are similar to the predicates used in policies, the final decisions cannot be generated by only manipulating intermediate decisions from each policy. To address this issue, they adopted the approach of translating a policy expression into a logic program in order to compute the final authorization decisions. Such an approach has one major limitation. Only positive decisions can be supported which greatly limits the expressiveness and practical application of their approach. considered. All their operators, except the template operator, can be described naturally by our 3 operations under the XACML context. The function intended to be achieved by the template operator can be achieved more naturally by decision expressions (see Section 5).

Inspired by [5, 6], Wijesekera et al. [23, 22] proposed another algebra to compose decisions from access control policies. Policies in such approaches are described as transformations of non-deterministic assignments of permission sets to subjects, that is, if a subject s is assigned a set x of permission sets by a policy, only one permission set in x can be really assigned to s but how to choose the permission set is non-deterministic. This semantics is quite different from the standard notion of access control policies and is more typical of meta-policies on access control policies or constraints on permissions assignments, e.g., separation of duty constraints. Although we can point out that this work suffers from limitation on its expressiveness, e.g. only positive and negative decisions are supported, we cannot directly compare their proposal with our approach due to the different notion of policies.

Backes et al. [3, 2] proposed an algebra for composing policies written in E-P3P [1]. Such an algebra supports positive decisions, negative decisions, and obligations fulfilled on decisions. The algebra has some limitations. It does not address conflicts in that it requires policies to be conflict-free, referred to as well-formed policies. Moreover, the algebra is not functionally complete though permit, deny, and not applicable decisions are supported. Raub et al. [19] extended the algebra by Backes et al. [2] by a better obligation formalization, but their approach still suffers from similar problems.

Bruns et al. [7, 8] proposed a policy language PBel based on Belnap logic, a four-valued Belnap logic. PBel probably is the closest proposal to $\mathfrak{D}$-algebra. "grant", "deny", "undefined", and "conflict", four values in PBel, correspond to $\{p\}$, $\{d\}$, $\{\frac{n}{a}\}$, and $\{p, d\}$, respectively, in our P-interpretation. The intention of PBel is similar to that of $\mathfrak{D}$-algebra, that is to provide a functionally complete set of logical connectives to combine access control policy decisions. However, due to its limited decision set, PBel can neither express decisions resulted from uncertainties (e.g. "indeterminate" from XACML) due to errors nor identify subtle issues in XACML, not even to mention solutions to these issues. Limited by the expressiveness of Belnap logic, PBel cannot support majority voting either.

9. CONCLUSIONS

In this paper, we have proposed a many-valued decision algebra which is both functionally complete and computationally effective. The importance of the algebra is justified by its application to policy combination and PDP design. We are working on implementing the decision specification language and developing a toolkit for evaluating policy expressions based on the algebra. Issues concerning decision composition for hierarchies are also being investigated.

10. REFERENCES

[1] P. Ashley, S. Hada, G. Karjoth, and M. Schunter. E-p3p privacy policies and privacy authorization. In *WPES*, pages 103–109, 2002.

[2] M. Backes, M. Dürmuth, and R. Steinwandt. An algebra for composing enterprise privacy policies. In P. Samarati, P. Y. A. Ryan, D. Gollmann, and R. Molva, editors, *ESORICS*, volume 3193 of *Lecture Notes in Computer Science*, pages 33–52. Springer, 2004.

[3] M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. In E. Snekkenes and D. Gollmann, editors, *ESORICS*, volume 2808 of *Lecture Notes in Computer Science*, pages 162–180. Springer, 2003.

[4] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 184–198, Washington, DC, USA, 2006. IEEE Computer Society.

[5] P. A. Bonatti, S. D. C. di Vimercati, and P. Samarati. A modular approach to composing access control policies. In *ACM Conference on Computer and Communications Security*, pages 164–173, 2000.

[6] P. A. Bonatti, S. D. C. di Vimercati, and P. Samarati. An algebra for composing access control policies. *ACM Trans. Inf. Syst. Secur.*, 5(1):1–35, 2002.

[7] G. Bruns, D. S. Dantas, and M. Huth. A simple and expressive semantic framework for policy composition in access control. In P. Ning, V. Atluri, V. D. Gligor, and H. Mantel, editors, *FMSE*, pages 12–21. ACM, 2007.

[8] G. Bruns and M. Huth. Access-control policies via belnap logic: Effective and efficient composition and analysis. In *CSF*, pages 163–176. IEEE Computer Society, 2008.

[9] C. C. Chang. Algebraic analysis of many valued logics. *Transactions of the American Mathematical Society*, 88(2):467–490, jul 1958.

[10] C. C. Chang. A new proof of the completeness of the lukasiewicz axioms. *Transactions of the American Mathematical Society*, 93(1):74–80, 1959.

[11] M. Fitting. Kleene's logic, generalized. *J. Log. Comput.*, 1(6):797–810, 1991.

[12] R. L. Graham. On n-valued functionally complete truth functions. *The Journal of Symbolic Logic*, 32(2):190–195, 1967.

[13] W. H. Jobe. Functional completeness and canonical forms in many-valued logics. *The Journal of Symbolic Logic*, 27(4):409–422, 1962.

[14] J. Łukasiewicz. O logice trojwartosciowej. *Ruch filozoficzny*, 5:170–171, 1920.

[15] J. Łukasiewicz. *Aristotle's Syllogistic from the Standpoint of Modern Formal Logic.* Garland Pub., New York, USA, first edition, 1987.

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

[17] R. McNaughton. A theorem about infinite-valued sentential logic. *The Journal of Symbolic Logic*, 16(1):1–13, 1951.

[18] OASIS. eXtensible Access Control Markup Language (XACML) 2.0. Available at http://www.oasis-open.org/.

[19] D. Raub and R. Steinwandt. An algebra for enterprise privacy policies closed under composition and conjunction. In *ETRICS*, pages 130–144, 2006.

[20] A. Rose and J. B. Rosser. Fragments of many-valued statement calculi. *Transactions of the American Mathematical Society*, 87(1):1–53, 1958.

[21] J. B. Rosser and A. R. Turquette. *Many-Valued Logics.* North-Holland Publishing Co., Amsterdam, Netherland, first edition, 1952.

[22] D. Wijesekera and S. Jajodia. Policy algebras for access control: the propositional case. In *ACM Conference on Computer and Communications Security*, pages 38–47, 2001.

[23] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. *ACM Trans. Inf. Syst. Secur.*, 6(2):286–325, 2003.

APPENDIX

A. PROOFS OF INCOMLETENESS

$\{\oplus, \neg\}$ is not functionally complete.

Proof. Because a D-algebra is a MV-algebra, we just need to prove that $\{\oplus, \neg\}$ is not functionally complete for a D-algebra. Let $D = \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$, in what follows, we will show that any possible unary operations $f(x)$ constructed by $\{\oplus, \neg\}$ on D can only generate a small set of result decisions, much smaller than $4^4 = 256$, the total number of decision matrices for a unary operation.

The idea of the proof is similar to steps used to build a least Herbrand model for a logic program. In the beginning, we only have one variable x, two operations, a constant sequence set that consists of 4 constant sequences that represent $\emptyset, \{0\}, \{1\}, \{0,1\}$ respectively, and one domain sequence set that consists only one sequence $(\emptyset, \{0\}, \{1\}, \{0,1\})$ in the beginning. Note that order matters for sequences. The elements of the domain sequence set are the input of $f(x)$. Then recursively apply the two operations on the elements in both the constant sequence set and the domain sequence set. In each step, the result sequences will be inserted into the domain sequence set that is used as the input of next step. It is fairly quick to reach a fix point.

The constant sequence set: $\{ (\emptyset, \emptyset, \emptyset, \emptyset), (\{0\}, \{0\}, \{0\}, \{0\}), (\{1\}, \{1\}, \{1\}, \{1\})\}, (\{0,1\}, \{0,1\}, \{0,1\}, \{0,1\}) \}$

Step 0: the domain sequence set $\{(\emptyset, \{0\}, \{1\}, \{0,1\})\}$.

Step 1: the domain sequence set $\{(\emptyset, \{0\}, \{1\}, \{0,1\}), (\emptyset, \emptyset, \emptyset, \emptyset), (\{0\}, \{0\}, \{0\}, \{0\}), (\{1\}, \{1\}, \{1\}, \{1\}), (\{0,1\}, \{0,1\}, \{0,1\}, \{0,1\}), (\{0,1\}, \{1\}, \{0\}, \emptyset), (\{0\}, \{0\}, \{0,1\}, \{0,1\}), (\{1\}, \{0,1\}, \{1\}, \{0,1\})\}$.

Step 2: the domain sequence set $\{(\emptyset, \{0\}, \{1\}, \{0,1\}), (\emptyset, \emptyset, \emptyset, \emptyset), (\{0\}, \{0\}, \{0\}, \{0\}), (\{1\}, \{1\}, \{1\}, \{1\}), (\{0,1\}, \{0,1\}, \{0,1\}, \{0,1\}), (\{0,1\}, \{1\}, \{0\}, \emptyset), (\{0\}, \{0\}, \{0,1\}, \{0,1\}), (\{1\}, \{0,1\}, \{1\}, \{0,1\}), (\{1\}, \{1\}, \emptyset, \emptyset), (\{0\}, \emptyset, \{0\}, \emptyset), (\{0,1\}, \{0,1\}, \{0\}, \{0\}), (\{0,1\}, \{1\}, \{0,1\}, \{1\}) \}$.

Step 3: the domain sequence set $\{(\emptyset, \{0\}, \{1\}, \{0,1\}), (\emptyset, \emptyset, \emptyset, \emptyset), (\{0\}, \{0\}, \{0\}, \{0\}), (\{1\}, \{1\}, \{1\}, \{1\}), (\{0,1\}, \{0,1\}, \{0,1\}, \{0,1\}), (\{0,1\}, \{1\}, \{0\}, \emptyset), (\{0\}, \{0\}, \{0,1\}, \{0,1\}), (\{1\}, \{0,1\}, \{1\}, \{0,1\}), (\{1\}, \{1\}, \emptyset, \emptyset), (\{0\}, \emptyset, \{0\}, \emptyset), (\{0,1\}, \{0,1\}, \{0\}, \{0\}), (\{0,1\}, \{1\}, \{0,1\}, \{1\}) , (\emptyset, \emptyset, \{1\}, \{1\}), (\emptyset, \{0\}, \emptyset, \{0\}) \}$.

Step 4: the domain sequence set $\{(\emptyset, \{0\}, \{1\}, \{0,1\}), (\emptyset, \emptyset, \emptyset, \emptyset), (\{0\}, \{0\}, \{0\}, \{0\}), (\{1\}, \{1\}, \{1\}, \{1\}), (\{0,1\}, \{0,1\}, \{0,1\}, \{0,1\}), (\{0,1\}, \{1\}, \{0\}, \emptyset), (\{0\}, \{0\}, \{0,1\}, \{0,1\}), (\{1\}, \{0,1\}, \{1\}, \{0,1\}), (\{1\}, \{1\}, \emptyset, \emptyset), (\{0\}, \emptyset, \{0\}, \emptyset), (\{0,1\}, \{0,1\}, \{0\}, \{0\}), (\{0,1\}, \{1\}, \{0,1\}, \{1\}) , (\emptyset, \emptyset, \{1\}, \{1\}), (\emptyset, \{0\}, \emptyset, \{0\}), (\{0\}, \emptyset, \{0,1\}, \{1\}), (\{1\}, \{0,1\}, \emptyset, \{0\})\}$.

Step 5: the domain sequence set $\{(\emptyset, \{0\}, \{1\}, \{0,1\}), (\emptyset, \emptyset, \emptyset, \emptyset), (\{0\}, \{0\}, \{0\}, \{0\}), (\{1\}, \{1\}, \{1\}, \{1\}), (\{0,1\}, \{0,1\}, \{0,1\}, \{0,1\}), (\{0,1\}, \{1\}, \{0\}, \emptyset), (\{0\}, \{0\}, \{0,1\}, \{0,1\}), (\{1\}, \{0,1\}, \{1\}, \{0,1\}), (\{1\}, \{1\}, \emptyset, \emptyset), (\{0\}, \emptyset, \{0\}, \emptyset), (\{0,1\}, \{0,1\}, \{0\}, \{0\}), (\{0,1\}, \{1\}, \{0,1\}, \{1\}) , (\emptyset, \emptyset, \{1\}, \{1\}), (\emptyset, \{0\}, \emptyset, \{0\}), (\{0\}, \emptyset, \{0,1\}, \{1\}), (\{1\}, \{0,1\}, \emptyset, \{0\})\}$.

Because the domain sequence set in step 5 is the same as that in step 4, we reach a fix point here. We are done! The number of different decision matrices from all possible unary functions that can be built from $\{\oplus, \neg\}$ indeed is quite small.

Step two: $\{\otimes, \oplus, \neg\}$ is functionally complete, please look at Section 3.1. □