The Logic of XACML

Ramli, Carroline Dewi Puspa Kencana; Nielson, Hanne Riis; Nielson, Flemming

Published in: Proceedings of FACS 2011

Publication date: 2011

The Logic of XACML

Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, Flemming Nielson

Department of Informatics and Mathematical Modelling
Danmarks Tekniske Universitet
Lyngby, Denmark
{cdpu,riis,nielson}@imm.dtu.dk

Abstract We study the international standard XACML 3.0 for describing security access control policy in a compositional way. Our main contribution is to derive a logic that precisely captures the idea behind the standard and to formally define the semantics of the policy combining algorithms of XACML. To guard against modelling artifacts we provide an alternative way of characterizing the policy combining algorithms and we formally prove the equivalence of these approaches. This allows us to pinpoint the shortcoming of previous approaches to formalisation based either on Belnap logic or on $D$-algebra.

1 Introduction

XACML (eXtensible Access Control Markup Language) is an approved OASIS Standard access control language [1][2]. XACML describes both an access control policy language and a request/response language. The policy language is used to express access control policies (who can do what when) while the request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses).

In order to manage modularity in access control, XACML constructs policies into several components, namely PolicySet, Policy and Rule. A PolicySet is a collection of others PolicySets or Policies whereas a Policy consists of one or more Rules. A Rule is the smallest component of XACML policy and each Rule only either grants or denies an access. As an illustration, suppose we have access control policies used within the National Health Care System. The system is composed of several access control policies of local hospitals. Each local hospital has its own policies such as patient policy, doctor policy, administration policy, etc. Each policy contains one or more particular rules, for example, in patient policy there is a rule that only the designated patient can read his or her record. In this illustration, both the National Health Care System and local hospital policies are PolicySets. However the patient policy is a Policy and one of its rules is the patient record policy. Every policy is only applicable to a certain target and a policy is applicable when a request matches to its target, otherwise, it is not

1 OASIS (Organization for the Advancement of Structured Information Standard) is a non-for-profit, global consortium that drives the development, convergence, and adoption of e-business standards. Information about OASIS can be found at [http://www.oasis-open.org](http://www.oasis-open.org)
applicable. The evaluation of composing policies is based on a combining algorithm – the procedure for combining decisions from multiple policies. There are four standard combining algorithms in XACML i.e., (i) permit-overrides, (ii) deny-overrides, (iii) first-applicable and (iv) only-one-applicable.

The syntax of XACML is based on XML format [2], while its standard semantics is described normatively using natural language in [14]. Using English paragraphs in standardization leads to misinterpretation and ambiguity. In order to avoid this drawback, we define an abstract syntax of XACML 3.0 and a formal XACML components evaluation based on XACML 3.0 specification in Section 2. Furthermore, the evaluation of the XACML combining algorithms is explained in Section 3.

Recently there are some approaches to formalizing the semantics of XACML. In [8], Halpern and Weissman show XACML formalization using First Order Logic (FOL). However, their formalization does not capture whole XACML specification. It is too expensive to express XACML combining algorithms in FOL. Kolovski et al. in [10,11] maps a large fragment of XACML to Description Logic (DL) – a subset of FOL – but they leave out the formalization of only-one-applicable combining algorithm. Another approach is to represent XACML policies in term of Answer Set Programming (ASP). Although Ahn et al. in [3] show a complete XACML formalization in ASP, their formalization is based on XACML 2.0, which is out-of-date nowadays. More particular, the combining algorithms evaluation in XACML 2.0 is simpler than XACML 3.0. Our XACML 3.0 formalization is closer to multi-valued logic approach such as Belnap logic [4] and $\mathcal{D}$-algebra [13]. Bruns et al. in [5,6] and Ni et al. in [13] define a logic for XACML using Belnap logic and $\mathcal{D}$-algebra, respectively. In some cases, both works show different results from the XACML standard specification. We discuss the shortcoming of formalization based either on Belnap logic or on $\mathcal{D}$-algebra in Section 4 and we conclude in Section 5.

2 XACML Components

XACML syntax is describe verbosely in XML format. For our analysis purpose, we do abstracting XACML components. From the abstraction XACML, we show how XACML evaluates policies. We give an example how XACML policies can be described in our abstraction and the components evaluation at the end of this section.

2.1 Abstracting XACML Components

There are three main policy components in XACML, namely PolicySet, Policy and Rule. PolicySet is the root of all XACML policies. A PolicySet is composed of a sequence of others PolicySet or Policy components along with a policy combining algorithm ID and a Target. A Policy is composed of a sequence of Rule, a Target and a rule combining algorithm ID. A Rule is a single entity that defines the individual rule in the policy. A Rule is composed of a Target, a Condition and its effect, i.e., either deny or permit. A Target is an XACML component that indicates under which categories an XACML policy is applicable. A Target consists of conjunction of AnyOf component with each AnyOf consists of
disjunction of AllOf components and each AllOf consists of conjunction of Match. Each Match contains only one particular category to be matched with the request. Typical categories of XACML attributes are subject category (e.g. human user, workstation, etc) action category (e.g. read, write, delete, etc), resource category (e.g. database, server, etc) and environment category (e.g. SAML, J2SE, CORBA, etc). A Condition is a set of propositional formulæ that refines the applicability of a Rule.

A Request contains a set of available informations on desired access request such as subject, action, resource and environment categories. A Request also contains additional information about external state, e.g. the current time, the temperature, etc.

We present in Table 1 a succinct syntax of XACML 3.0 that is faithful to the more verbose syntax used in the standard [14].

Table 1. Abstraction of XACML 3.0 Components

<table>
<thead>
<tr>
<th>XACML Policy Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolicySet := ⟨Target, ⟨PolicySet₁, ..., PolicySetₘ⟩, θ⟩</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Policy := ⟨Target, ⟨Rule₁, ..., Ruleₘ⟩, θ⟩</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Rule := ⟨Effect, Target, Condition⟩</td>
</tr>
<tr>
<td>Condition := propositional formulæ</td>
</tr>
<tr>
<td>Target := Null</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>AnyOf := AllOf₁ ∨ ... ∨ AllOfₘ</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>AllOf := Match₁ ∧ ... ∧ Matchₘ</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Match := Φ(α)</td>
</tr>
<tr>
<td>Φ := subject</td>
</tr>
<tr>
<td>α := attribute value</td>
</tr>
<tr>
<td>θ := p − o</td>
</tr>
<tr>
<td>Effect := d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XACML Request Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request := {A₁, ..., Aₘ}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>A := Φ(α)</td>
</tr>
</tbody>
</table>

2.2 XACML Evaluation

The evaluation of XACML components starts from Match evaluation and it is continued iteratively until the PolicySet evaluation. The Match, AllOf, AnyOf, and Target values are either match, not match or indeterminate. The value can be indeterminate if there is an error during the evaluation so that the decision cannot be made at that moment. The Rule evaluation depends on Target evaluation and Condition evaluation. The Condition component is a set of propositional formulæ which each formula is evaluated to either true, false or indeterminate. An empty Condition is always evaluated to true. The result of Rule is either applicable, not applicable or indeterminate. An applicable Rule has effect either deny or permit. Finally, the evaluation of Policy and PolicySet are based on a combining algorithm of which the result can be either applicable (with its effect either deny or permit), not applicable or indeterminate.
2.2.1 Three-Valued Lattice

We use three-valued logic to determine XACML evaluation value. We define \( L_3 = \langle V_3, \leq \rangle \) be three-valued lattice where \( V_3 \) is the set \{ \( \top, I, \bot \) \} and \( \bot \leq I \leq \top \). Given a subset \( S \) of \( V_3 \), we denote the greatest lower bound (glb) and the least upper bound (lub) at \( S \) (w.r.t. \( L_3 \)) by \( \bigwedge S \) and \( \bigvee S \), respectively. Recall that \( \bigwedge \emptyset = \top \) and \( \bigvee \emptyset = \bot \).

We use \( \llbracket \rrbracket \) notation to map XACML elements into their evaluation values. The evaluation of XACML components to values in \( V_3 \) is summarized in Table 2.

<table>
<thead>
<tr>
<th>( V_3 )</th>
<th>Match and Target value</th>
<th>Condition value</th>
<th>Rule, Policy and PolicySet value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \top )</td>
<td>match</td>
<td>true</td>
<td>applicable (either deny or permit)</td>
</tr>
<tr>
<td>( \bot )</td>
<td>not match</td>
<td>false</td>
<td>not applicable</td>
</tr>
<tr>
<td>( I )</td>
<td>indeterminate</td>
<td>indeterminate</td>
<td>indeterminate</td>
</tr>
</tbody>
</table>

2.2.2 Match Evaluation

A Match element \( M \) is an attribute value that the request should fulfill. Given a Request component \( Q \), the evaluation of Match element is as follows:

\[
\llbracket M \rrbracket(Q) = \begin{cases} 
  \top & M \in Q \\
  \bot & M \notin Q \\
  I & \text{there is an error during the evaluation}
\end{cases}
\]  

(1)

2.2.3 Target Evaluation

Let \( M \) be a Match, \( A = M_1 \land \ldots \land M_m \) be an AllOf, \( E = A_1 \lor \ldots \lor A_n \) be an AnyOf, \( T = E_1 \land \ldots \land E_o \) be a Target and \( Q \) be a Request. Then, the evaluations of AllOf, AnyOf, and Target are as follows:

\[
\llbracket A \rrbracket(Q) = \bigwedge_{i=1}^{m} \llbracket M_i \rrbracket(Q) 
\]  

(2)

\[
\llbracket E \rrbracket(Q) = \bigvee_{i=1}^{n} \llbracket A_i \rrbracket(Q) 
\]  

(3)

\[
\llbracket T \rrbracket(Q) = \bigwedge_{i=1}^{o} \llbracket E_i \rrbracket(Q) 
\]  

(4)

In summary, we can simplify the Target evaluation as follows:

\[
\llbracket T \rrbracket(Q) = \bigwedge \bigvee \bigwedge \llbracket M \rrbracket(Q) 
\]  

(5)

An empty Target – indicated by \( \text{Null} \) – is always evaluated to \( \top \).
2.2.4 Condition Evaluation

We define the conditional evaluation function \( \text{eval} \) as an arbitrary function to evaluate Condition to value in \( V_3 \) given a Request component \( Q \). The evaluation of Condition is defined as follows:

\[
\llbracket C \rrbracket(Q) = \text{eval}(C, Q)
\]  

(6)

2.2.5 Extended Values

In order to distinguish an applicable policy to permit an access from applicable policy to deny an access, we extend \( \top \) in \( V_3 \) to \( \top_p \) and \( \top_d \), respectively. The same case also applies to indeterminate value. The extended indeterminate value contains the potential effect values which could have occurred if there would not have been an error during a evaluation. The possible extended indeterminate values are [14]:

- Indeterminate Deny (\( I_d \)): an indeterminate from a policy which could have evaluated to deny but not permit, e.g., a Rule which evaluates to indeterminate and its effect is deny.
- Indeterminate Permit (\( I_p \)): an indeterminate from a policy which could have evaluated to permit but not deny, e.g., a Rule which evaluates to indeterminate and its effect is permit.
- Indeterminate Deny Permit (\( I_{dp} \)): an indeterminate from a policy which could have effect either deny or permit.

We extend the set \( V_3 \) to \( V_6 = \{ \top_p, \top_d, I_d, I_p, I_{dp}, \bot \} \) and we use \( V_6 \) for XACML policies evaluations.

2.2.6 Rule Evaluation

Let \( R = \langle *, T, C \rangle \) be a Rule and \( Q \) be a Request. Then, the evaluation of Rule is determined as follows:

\[
\llbracket R \rrbracket(Q) = \begin{cases} 
\top & \llbracket T \rrbracket(Q) = \top \text{ and } \llbracket C \rrbracket(Q) = \top \\
\bot & (\llbracket T \rrbracket(Q) = \bot \text{ and } \llbracket C \rrbracket(Q) = \bot) \text{ or } \llbracket T \rrbracket(Q) = \bot \\
I_* & \text{otherwise}
\end{cases}
\]  

(7)

Let \( F \) and \( G \) be two values in \( V_3 \). We define a new operator \( \sim \): \( V_3 \times V_3 \to V_3 \) as follows:

\[
F \sim G = \begin{cases} 
G & F = \top \\
F & \text{otherwise}
\end{cases}
\]  

(8)

We define a function \( \sigma : V_3 \times \{ p, d \} \to V_6 \) that maps a value in \( V_3 \) into a value in \( V_6 \) given a particular Rule’s effect as follows:

\[
\sigma(X, *) = \begin{cases} 
X & X = \bot \\
X_* & \text{otherwise}
\end{cases}
\]  

(9)
Proposition 1. Let $R = (\ast, T, C)$ be a Rule and $Q$ be a Request. Then, the following equation holds

$$\|R\|(Q) = \sigma (\|T\|(Q) \sim \|C\|(Q), \ast)$$  \hspace{1cm} (10)

### 2.2.7 Policy Evaluation

The standard evaluation of Policy element taken from \cite{14} is as follows

<table>
<thead>
<tr>
<th>Target value</th>
<th>Rule value</th>
<th>Policy Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>match</td>
<td>At least one Rule value is applicable</td>
<td>Specified by the combining algorithm</td>
</tr>
<tr>
<td>match</td>
<td>All Rule values are not applicable</td>
<td>not applicable</td>
</tr>
<tr>
<td>not match or</td>
<td>At least one Rule value is indeterminate</td>
<td>Specified by the combining algorithm</td>
</tr>
<tr>
<td>indeterminate</td>
<td>Don’t care</td>
<td>Not applicable</td>
</tr>
<tr>
<td></td>
<td>Don’t care</td>
<td>Indeterminate</td>
</tr>
</tbody>
</table>

Let $P = (T, R, \theta)$ be a Policy where $R = (R_1, \ldots, R_n)$. Let $Q$ be a Request and $R' = (\|R_1\|(Q), \ldots, \|R_n\|(Q))$. The evaluation of Policy is defined as follows:

$$\|P\|(Q) = \begin{cases} I_*, & \|T\|(Q) = I \text{ and } \bigoplus_{\theta} (R') \in \{ I*, I_* \} \\ \bot, & \|T\|(Q) = \bot \text{ or } \|T\|(Q) = I \text{ and } \forall R_i: \|R_i\|(Q) = \bot \\ \bigoplus_{\theta} (R'), & \text{otherwise} \end{cases}$$  \hspace{1cm} (11)

Note 1. The combining algorithms denoted by $\bigoplus$ is explained in Section \cite{3}.

### 2.2.8 PolicySet Evaluation

The evaluation of PolicySet is similar to Policy evaluation. However, the input of the combining algorithm is a sequence of either PolicySet or Policy components.

Let $PS = (T, P, \theta)$ be a PolicySet where $P = (P_1, \ldots, P_n)$. Let $Q$ be a Request and $P' = (\|P_1\|(Q), \ldots, \|P_n\|(Q))$. The evaluation of PolicySet is defined as follows:

$$\|PS\|(Q) = \begin{cases} I_*, & \|T\|(Q) = I \text{ and } \bigoplus_{\theta} (P') \in \{ I*, I_* \} \\ \bot, & \|T\|(Q) = \bot \text{ or } \|T\|(Q) = I \text{ and } \forall P_i: \|P_i\|(Q) = \bot \\ \bigoplus_{\theta} (P'), & \text{otherwise} \end{cases}$$  \hspace{1cm} (12)

### 2.3 Example

The following example simulates briefly how a policy is built using the abstraction. The example is motivated by \cite{7,8} which presents a health information system for a small nursing home in New South Wales, Australia.

Example 1 (Patient Policy). The general policy in the hospital in particular:
1. Patient Record Policy
   - RP1: only designated patient can read his or her patient record except that if
     the patient is less than 18 years old, the patient’s guardian is permitted also
     read the patient’s record,
   - RP2: patients may only write patient surveys into their own records
   - RP3: both doctors and nurses are permitted to read any patient records,

2. Medical Record Policy
   - RM1: doctors may only write medical records for their own patients and
   - RM2: may not write any other patient records,

The XACML policies for this example is shown in Figure 1. The topmost policy in
this example is the Patient Policy that contains two policies, namely the Patient
Record Policy and the Medical Record Policy. The access is granted if either one of the Patient
Record Policy or the Medical Record Policy gives a permit access. Thus in this case,
we use permit-overrides combining algorithm to combine those two policies. In order
to restrict the access, each policy denies an access if there is a rule denies it. Thus, we
use deny-overrides combining algorithms to combine the rules.

Suppose now there is an emergency situation and a doctor D asks permission to
read patient record P. The Request is as follows:

```plaintext
{ subject(doctor), action(read), resource(patient_record),
  doctor(id,d), patient(id,p), patient_record(id,p) }
```
Only Target RP3 matches for this request and the effect of RP3 is permit. Thus, the final result is doctor $D$ is allowed to read patient record $P$. Now, suppose that after doing some treatment, the doctor wants to update the medical record. A request is sent

```
{ subject(doctor), action(write), resource(medical_record),
  doctor(id,d), patient(id,p), medical_record(id,p) }
```

The Target RM1 and the Target RM2 match for this request, however because doctor $D$ is not registered as patient $P$’s doctor thus Condition RM1 is evaluated to false while Condition RM2 is evaluated to true. In consequence, Rule RM1 is not applicable while Rule RM2 is applicable with effect deny.

3 Combining Algorithms

Currently, there are four basic combining algorithms in XACML, namely (i) permit-overrides, (ii) deny-overrides, (iii) first-applicable, and (iv) only-one-applicable. The input of a combining algorithm is a sequence of Rule, Policy or PolicySet values. In this section we give formalizations of the XACML 3.0 combining algorithms based on [14]. To guard against modelling artifacts we provide an alternative way of characterizing the policy combining algorithms and we formally prove the equivalence of these approaches.

3.1 Pairwise Policy Values

In $V_0$ we define the truth values of XACML components by extending $\top$ to $\top_p$ and $\top_d$ and $I$ to $I_d$. $I_p$ and $I_{dp}$. This approach shows straightforwardly the status of XACML component. However, it is easier if we use numerical encoding when we need to do a computation, especially for computing policies compositions. Thus, we encode all the values returned by algorithms as pairs of natural numbers.

In this numerical encoding, the value $1$ represents an applicable value (either deny or permit), $\frac{1}{2}$ represents indeterminate value and $0$ means there is no applicable value. In each tuple, the first element represents the Deny value ($\top_d$) and the later represents Permit value ($\top_p$). We can say $[0, 0]$ for not applicable (⊥) because neither Deny nor Permit is applicable, $[1, 0]$ for applicable with deny effect ($\top_d$) because only Deny value is applicable, $[\frac{1}{2}, 0]$ for $I_d$ because the Deny part is indeterminate, $[\frac{1}{2}, \frac{1}{2}]$ for $I_{dp}$ because both Deny and Permit have indeterminate values. The conversion applies also for Permit.

A set of pairwise policy values is $P = \{ [0, 0], [\frac{1}{2}, 0], [0, \frac{1}{2}], [1, 0], [\frac{1}{2}, \frac{1}{2}], [0, 1] \}$. Let $[D, P]$ be an element on $P$. We denote $d([D, P]) = D$ and $p([D, P]) = P$ for the function that returns the Deny value and Permit value, respectively.

\[ An extended version of this paper with all the proofs is available at http://www2.imm.dtu.dk/~cdpu/Papers/the_logic_of_XACML-extended.pdf \]
We define $\delta : V_6 \to P$ as a mapping function that maps $V_6$ into $P$ as follows:

$$\delta(X) = \begin{cases} [0, 0] & X = \perp \\ [1, 0] & X = \top_d \\ [0, 1] & X = \top_p \\ \left[\frac{1}{2}, 0\right] & X = I_d \\ \left[\frac{1}{2}, \frac{1}{2}\right] & X = I_{dp} \end{cases}$$ (13)

We define $\delta$ over a sequence $S$ as $\delta(S) = \langle \delta(s) | s \in S \rangle$.

We use pairwise comparison for the order of $P$. We define an order $\preceq_P$ for $P$ as follows $[D_1, P_1] \preceq_P [D_2, P_2]$ iff $D_1 \leq D_2$ and $P_1 \leq P_2$ with $0 \leq \frac{1}{2} \leq 1$. We write $P_P$ for the partial ordered set (poset) $(P, \preceq_P)$ illustrated in Figure 2.

Let $\max : 2^{\mathbb{R}} \to \mathbb{R}$ be a function that returns the maximum value of a set of rational numbers and let $\min : 2^{\mathbb{R}} \to \mathbb{R}$ be a function that returns the minimum value of a set of rational numbers. We define $Max_{\preceq_P} : 2^P \to P$ as a function that returns the maximum pairwise policy value which is defined as follows:

$$Max_{\preceq_P}(S) = [\max(\{d(X) | X \in S\}), \max(\{p(X) | X \in S\})]$$ (14)

and $Min_{\preceq_P} : 2^P \to P$ as a function that return the minimum pairwise policy value which is defined as follows:

$$Min_{\preceq_P}(S) = [\min(\{d(X) | X \in S\}), \min(\{p(X) | X \in S\})]$$ (15)

3.2 Permit-Overrides Combining Algorithm

The permit-overrides combining algorithm is intended for those cases where a permit decision should have priority over a deny decision. This algorithm (taken from [14]) has the following behaviour:

1. If any decision is $\top_p$ then the result is $\top_p$. 

![Figure 2. The Partial Ordered Set $P_P$ for Pairwise Policy Values](image-url)
Figure 3. The Lattice $L_{p-o}$ for The Permit-Overrides Combining Algorithm (left), The Lattice $L_{d-o}$ for The Deny-Overrides Combining Algorithm (middle) and The Lattice $L_{o-1-a}$ for The Only-One-Applicable Combining Algorithm (right)

2. otherwise, if any decision is $I_{dp}$ then the result is $I_{dp}$.
3. otherwise, if any decision is $I_p$ and another decision is $I_d$ or $\top_d$, then the result is $I_{dp}$.
4. otherwise, if any decision is $I_p$ then the result is $I_p$.
5. otherwise, if decision is $\top_d$ then the result is $\top_d$.
6. otherwise, if any decision is $I_d$ then the result is $I_d$.
7. otherwise, the result is $\bot$.

We call $L_{p-o} = (V_6, \sqsubseteq_{p-o})$ for the lattice using the permit-overrides combining algorithm where $\sqsubseteq_{p-o}$ is the ordering depicted in Figure 3. The least upper bound operator for $L_{p-o}$ is denoted by $\bigcup_{p-o}$.

**Definition 1.** The permit-overrides combining algorithm $\bigoplus_{p-o}^{V_6}$ is a mapping function from a sequence of $V_6$ elements into an element in $V_6$ as the result of composing policies. Let $S = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values in $V_6$ and $S' = \{ s_1, \ldots, s_n \}$. We define the permit-overrides combining algorithm under $V_6$ as follows:

$$
\bigoplus_{p-o}^{V_6}(S) = \bigcup_{p-o} S'
$$

(16)

The permit-overrides combining algorithm can also be expressed under $P$. The idea is that we inspect the maximum value of Deny and Permit in the set of pairwise policy values. We conclude that the decision is permit if the Permit is applicable (i.e. it has value 1). If the Permit is indeterminate (i.e. it has value $\frac{1}{2}$) then the decision is $I_{dp}$ if the Deny is either indeterminate (i.e. it has value $\frac{1}{2}$) or applicable (i.e. it has value 1). Otherwise we take the maximum value of Deny and Permit from the set of pairwise policy values as the result of permit-overrides combining algorithm.

**Definition 2.** The permit-overrides combining algorithm $\bigoplus_{p-o}^{P}$ is a mapping function from a sequence of $P$ elements into an element in $P$ as the result of composing policies. Let $S = \langle s_1, \ldots, s_n \rangle$ be a sequence of pairwise policy values and $S' = \{ s_1, \ldots, s_n \}$. 


We define the permit-overrides combining algorithm under $\mathbb{P}$ as follows:

$$
\mathbb{P} \oplus \bigotimes (S) = \begin{cases} 
[0, 1] & \text{Max}_{\mathbb{P}}(S') = [\perp, 1] \\
[\frac{1}{2}, \frac{1}{2}] & \text{Max}_{\mathbb{P}}(S') = [D, \frac{1}{2}], D \geq \frac{1}{2} \\
\text{otherwise} & \text{Max}_{\mathbb{P}}(S')
\end{cases}
$$

(17)

**Proposition 2.** Let $S$ be a sequence of policy values in $\mathbb{V}_6$. Then

$$
\delta_{\mathbb{P}} \bigotimes (\delta_0(S)) = \bigotimes (\delta(S))
$$

### 3.3 Deny-Overrides Combining Algorithm

The deny-overrides combining algorithm is intended for those cases where a deny decision should have priority over a permit decision. This algorithm (taken from [14]) has the following behaviour:

1. If any decision is $\top_d$ then the result is $\top_d$.
2. otherwise, if any decision is $I_{dp}$ then the result is $I_{dp}$.
3. otherwise, if any decision is $I_d$ and another decision is $I_p$ or $\top_p$, then the result is $I_{dp}$.
4. otherwise, if any decision is $I_d$ then the result is $I_d$.
5. otherwise, if decision is $\top_p$ then the result is $\top_p$.
6. otherwise, if any decision is $I_p$ then the result is $I_p$.
7. otherwise, the result is $\bot$.

We call $\mathcal{L}_{d-o} = (\mathbb{V}_6, \sqsubseteq_{d-o})$ for the lattice using the deny-overrides combining algorithm where $\sqsubseteq_{d-o}$ is the ordering depicted in Figure 3. The least upper bound operator for $\mathcal{L}_{d-o}$ is denoted by $\bigsqcup_{d-o}$.

**Definition 3.** The deny-overrides combining algorithm $\bigoplus_{d-o}^{\mathbb{V}_6}$ is a mapping function from a sequence of $\mathbb{V}_6$ elements into an element in $\mathbb{V}_6$ as the result of composing policies. Let $S = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values in $\mathbb{V}_6$ and $S' = \{ s_1, \ldots, s_n \}$. We define the deny-overrides combining algorithm under $\mathbb{V}_6$ as follows:

$$
\bigoplus_{d-o}^{\mathbb{V}_6}(S) = \bigsqcup_{d-o} S'
$$

(18)

The deny-overrides combining algorithm can also be expressed under $\mathbb{P}$. The idea is similar to permit-overrides combining algorithm by symmetry.

**Definition 4.** The deny-overrides combining algorithm $\bigoplus_{d-o}^{\mathbb{P}}$ is a mapping function from a sequence of $\mathbb{P}$ elements into an element in $\mathbb{P}$ as the result of composing policies. Let $S = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values in $\mathbb{P}$ and $S' = \{ s_1, \ldots, s_n \}$. We define the deny-overrides combining algorithm under $\mathbb{P}$ as follows:

$$
\bigoplus_{d-o}^{\mathbb{P}}(S) = \begin{cases} 
[1, 0] & \text{Max}_{\mathbb{P}}(S') = [1, \perp] \\
[\frac{1}{2}, \frac{1}{2}] & \text{Max}_{\mathbb{P}}(S') = [\frac{1}{2}, P], P \geq \frac{1}{2} \\
\text{otherwise} & \text{Max}_{\mathbb{P}}(S')
\end{cases}
$$

(19)
Proposition 3. Let $S$ be a sequence of policy values in $V_6$. Then
\[
\delta(\bigoplus_{d-o}(S)) = \bigoplus_{d-o}(\delta(S))
\]

3.4 First-Applicable Combining Algorithm

The result of first-applicable algorithm is the first Rule, Policy or PolicySet element in the sequence whose Target and Condition is applicable. The pseudo-code of the first-applicable combining algorithm in XACML 3.0 [14] shows that the result of this algorithm is the first Rule, Policy or PolicySet that is not "not applicable". The idea is that there is a possibility an indeterminate policy could return to be an applicable policy. The first-applicable combining algorithm under $V_6$ and $P$ are defined below.

Definition 5 (First-Applicable Combining Algorithm). The first-applicable combining algorithm $\bigoplus_{f-a}^{V_6}$ is a mapping function from a sequence of $V_6$ elements into an element in $V_6$ as the result of composing policies. Let $S = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values in $V_6$. We define the first-applicable combining algorithm under $V_6$ as follows:
\[
\bigoplus_{f-a}^{V_6}(S) = \begin{cases} 
  s_i & \exists i : s_i \neq \bot \text{ and } \forall j < i : s_j = \bot \\
  \bot & \text{otherwise}
\end{cases}
\]
(20)

Definition 6. The first-applicable combining algorithm $\bigoplus_{f-a}^{P}$ is a mapping function from a sequence of $P$ elements into an element in $P$ as the result of composing policies. Let $S = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values in $P$. We define the first applicable combining algorithm under $P$ as follows:
\[
\bigoplus_{f-a}^{P}(S) = \begin{cases} 
  s_i & \exists i : s_i \neq [0, 0] \text{ and } \forall j < i : s_j = [0, 0] \\
  [0, 0] & \text{otherwise}
\end{cases}
\]
(21)

Proposition 4. Let $S$ be a sequence of policy values in $V_6$. Then
\[
\delta(\bigoplus_{f-a}^{V_6}(S)) = \bigoplus_{f-a}^{P}(\delta(S))
\]

3.5 Only-One-Applicable Combining Algorithm

The result of the only-one-applicable combining algorithm ensures that one and only one policy is applicable by virtue of their Target. If no policy applies, then the result is not applicable, but if more than one policy is applicable, then the result is indeterminate. When exactly one policy is applicable, the result of the combining algorithm is the result of evaluating the single applicable policy.

We call $\mathcal{O}_{-1-a} = (V_0, \sqsubseteq_{-1-a})$ for the lattice using the only-one-applicable combining algorithm where $\sqsubseteq_{-1-a}$ is the ordering depicted in Figure 3. The least upper bound operator for $\mathcal{O}_{-1-a}$ is denoted by $\bigcup_{-1-a}$. 
Definition 7. The only-one-applicable combining algorithm $\bigoplus_{o-1-a}^V$ is a mapping function from a sequence of $V_6$ elements into an element in $V_6$ as the result of composing policies. Let $S = (s_1, \ldots, s_n)$ be a sequence of policy values in $V_6$ and $S' = \{s_1, \ldots, s_n\}$. We define only-one-applicable combining algorithm under $V_6$ as follows

$$
\bigoplus_{o-1-a}^V (S) = \begin{cases} 
I_d & \exists i, j : i \neq j, s_i = s_j = \top_d \text{ and } \\
I_p & \exists i, j : i \neq j, s_i = s_j = \top_p \text{ and } \\
\bigcup_{o-1-a} S' & \text{otherwise}
\end{cases}
$$

The only-one-applicable combining algorithm also can be expressed under $P$. The idea is that we inspect the maximum value of Deny and Permit returned from the given set of pairwise policy values. By inspecting the maximum value for each element, we know exactly the combination of pairwise policy values i.e., if we find that both Deny and Permit are not 0, it means that the Deny and the Permit are either applicable (i.e. it has value 1) or indeterminate (i.e. it has value $\frac{1}{2}$). Thus, the result of this algorithm is $I_{dp}$ (based on the XACML 3.0 Specification [14]). However if only one element is not 0 then there is a possibility that many policies have the same applicable (or indeterminate) values. If there are at least two policies with the Deny (or Permit) are either applicable or indeterminate value, then the result is $I_d$ (or $I_p$). Otherwise we take the maximum value of Deny and Permit from the given set of pairwise policy values as the result of only-one-applicable combining algorithm.

Definition 8. The only-one-applicable combining algorithm $\bigoplus_{o-1-a}^P$ is a mapping function from a sequence of $P$ elements into an element in $P$ as the result of composing policies. Let $S = (s_1, \ldots, s_n)$ be a sequence of policy values in $P$ and $S' = \{s_1, \ldots, s_n\}$. We define only-one-applicable combining algorithm under $P$ as follows

$$
\bigoplus_{o-1-a}^P (S) = \begin{cases} 
\left[\frac{1}{2}, \frac{1}{2}\right] & \text{Max}_{\sqsubseteq P}(S') = [D, P], D, P \geq \frac{1}{2} \\
\left[\frac{1}{2}, 0\right] & \text{Max}_{\sqsubseteq P}(S') = [D, 0], D \geq \frac{1}{2} \\
[0, \frac{1}{2}] & \text{Max}_{\sqsubseteq P}(S') = [0, P], P \geq \frac{1}{2} \\
\text{otherwise} & \exists i, j : i \neq j, d(s_i), d(s_j) \geq \frac{1}{2} \\
\end{cases}
$$

Proposition 5. Let $S$ be a sequence of policy values in $V_6$. Then

$$
\delta(\bigoplus_{o-1-a}^V (S)) = \bigoplus_{o-1-a}^P (\delta(S))
$$

4 Related Work

We will focus the discussion on the formalization of XACML using Belnap logic [4] and D-Algebra [13] – those two have a similar approach to the pairwise policy values
approach explained in Section 3. We show the shortcoming of the formalization on Bruns et al. work in [6] and Ni et al. work in [13].

4.1 XACML Semantics under Belnap Four-Valued Logic

Belnap in his paper [4] defines a four-valued logic over \( \text{four} = \{ \top\top, \top\bot, \bot\top, \bot\bot \} \). There are two orderings in Belnap logic, i.e., the knowledge ordering \((\leq_k)\) and the truth ordering \((\leq_t)\) (see Figure 4).

\[
\begin{align*}
\text{glb} &= \otimes_B \top\top \\
\text{lub} &= \oplus_B \top\bot \\
\text{glb} &= \wedge \\
\text{lub} &= \vee \\
\leq_k &= \top\bot \\
\leq_t &= \bot\top \\
\otimes &= \land \\
\oplus &= \lor
\end{align*}
\]

knowledge ordering \hspace{2cm} truth ordering

Figure 4. Bi-lattice of Belnap Four-Valued Logic

Bruns et al. in PBel [5,6] and also Hankin et al. in AspectKB [9] use Belnap four-valued logic to represent the composition of access control policies. The responses of an access control system are \( \top\bot \) when the policy is granted or access permitted, \( \bot\top \) when the policy is not granted or access is denied, \( \bot\bot \) when there is no applicable policy and \( \top\top \) when conflict arises, i.e., an access is both permitted and denied. Additional operators are added as follows [6]:

- overwriting operator \([y \mapsto z]\) with \(y, z \in \text{four}\). Expression \(x[y \mapsto z]\) yields \(x\) if \(x \neq y\), and \(z\) otherwise.
- priority operator \(x > y\); it is a syntactic sugar of \(x[\bot\bot \mapsto y]\).

Bruns et al. defined XACML combining algorithms using Belnap four-valued logic as follows [6]:

- **permit-overrides**: \((p \oplus_B q)[\top\top \mapsto \bot\top]\)
- **first-applicable**: \(p > q\)
- **only-one-applicable**: \((p \oplus_B q) \oplus_B ((p \oplus_B \neg p) \oplus_B (q \oplus_B \neg q))\)

Bruns et al. suggested that the indeterminate value is treated as \(\top\top\). However, with indeterminate as \(\top\bot\), the permit-overrides combining algorithm is not defined correctly. Suppose we have two policies: \(p\) and \(q\) where \(p\) is permit and \(q\) is indeterminate. The result of the permit-overrides combining algorithm is as follows \((p \oplus_B q)[\top\top \mapsto \bot\top] = (\top\top \oplus_B \bot\bot)[\top\top \mapsto \bot\top] = \top\top[\top\top \mapsto \bot\top] = \bot\top\). Based on the XACML 2.0 [12] and the XACML 3.0 [14], the result of permit-overrides combining algorithm should be permit (\(\top\bot\)). However, based on Belnap four-valued logic, the result is deny (\(\bot\top\)).

Bruns et al. tried to define indeterminate value as a conflict by formalizing it as \(\top\top\). However, their formulation of permit-overrides combining algorithm is inconsistent based on the standard XACML specification. Moreover, they said that sometimes
indeterminate should be treated as \( \perp \) and sometimes as \( \top \), but there is no explanation about under which circumstances that indeterminate is treated as \( \top \) or as \( \perp \). The treatment of indeterminate as \( \top \) is too strong because indeterminate does not always contains information about deny and permit in the same time. Only \( I_{dp} \) contains information both deny and permit. However, \( I_d \) and \( I_p \) only contain information only about deny and permit, respectively. Even so, the value \( \perp \) for indeterminate is too weak because indeterminate is treated as not applicable despite that there is information contained inside indeterminate value. The Belnap four-valued logic has no explicit definition of indeterminate. In contrast, the Belnap four-valued has a conflict value (i.e. \( \top \)).

4.2 XACML Semantics under \( D \)-Algebra

Ni et al. in [13] define \( D \)-algebra as a decision set together with some operations on it.

Definition 9 (\( D \)-algebra [13]). 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 \( \oplus^D \), \( \otimes^D \) be binary operations on elements in \( D \). A \( D \)-algebra is an algebraic structure \( (D, \neg, \oplus^D, \otimes^D, 0) \) closed on \( \neg \), \( \oplus^D \), \( \otimes^D \) and satisfying the following axioms:

1. \( x \oplus^D y = y \oplus^D x \)
2. \( (x \oplus^D y) \oplus^D z = x \oplus^D (y \oplus^D z) \)
3. \( x \oplus^D 0 = x \)
4. \( \neg \neg x = x \)
5. \( x \oplus^D \neg 0 = \neg 0 \)
6. \( \neg(\neg x \oplus^D y) \oplus^D y = \neg(y \oplus^D x) \oplus^D x \)
7. \( x \otimes^D y = \begin{cases} \neg 0 & : x = y \\ 0 & : x \neq y \end{cases} \)

In order to write formulae in a compact form, for \( x, y \in D \), \( x \otimes^D y = \neg(\neg x \oplus^D y) \) and \( x \oplus^D y = x \oplus^D \neg y \).

Ni et al. [13] show that XACML decisions contain three different value, i.e. permit (\( \{p\} \)), deny (\( \{d\} \)) and not applicable (\( \{n\} \)). Those decision are deterministic decisions. The non-deterministic decisions such as \( I_d \), \( I_p \) and \( I_{dp} \) are denoted by \( \{d, n\} \), \( \{p, n\} \), and \( \{d, p, n\} \), respectively. The interpretation of a \( D \)-algebra on XACML decisions is as follows [13]:

- \( D \) is represented by \( P(\{p,d,n\}) \)
- 0 is represented by \( \emptyset \)
- \( \neg x \) is represented by \( \{p,d,n\} - x \) where \( x \in D \)
- \( x \oplus^D y \) is represented by \( x \cup y \) where \( x, y \in D \)
- \( \otimes^D \) is defined by axiom 7

There are two values which are not in XACML, i.e. \( \emptyset \) and \( \{p,d\} \). Simply we say \( \emptyset \) for empty policy (or there is no policy) and \( \{p,d\} \) for a conflict.
The composition function of permit-overrides using $D$-Algebra is as follows:

$$f_{po}(x, y) = (x \oplus^D y)$$
$$\ominus^D ((x \otimes^D \{ p \}) \otimes^D (y \otimes^D \{ p \})) \odot^D \{ d, \text{na} \}$$
$$\ominus^D (\neg ((x \otimes^D y) \otimes^D \{ \text{na} \}) \odot^D (\neg (x \otimes^D \emptyset) \oplus^D (y \otimes^D \emptyset)))$$

The composition function that Ni et al. proposed is inconsistent with neither the XACML 3.0 [14] nor the XACML 2.0 [12] as they claimed in [13]. Below we show an example that compares all of the results of permit-overrides combining algorithm under the logics discussed in this paper.

**Example 2.** Given two policies $P_1$ and $P_2$ where $P_1$ is Indeterminate Permit and $P_2$ is Deny. Let us use the permit-overrides combining algorithm to compose those two policies. Table 3 shows the result of combining policies under Belnap logic, $D$-algebra, $V_6$ and $P$.

<table>
<thead>
<tr>
<th>Logic</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>Permit-Overrides Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belnap logic</td>
<td>$\top\top$</td>
<td>$\text{ff}$</td>
<td>($\top\top \oplus^D \text{ff}$)$\mapsto^{\top\top} \text{ff}$</td>
<td>$\text{ff}$</td>
</tr>
<tr>
<td>$D$-algebra</td>
<td>${ p, \text{na} }$</td>
<td>${ d }$</td>
<td>$f_{po}({ p, \text{na} }, { d })$</td>
<td>${ p, d }$</td>
</tr>
<tr>
<td>$V_6$</td>
<td>$I_p$</td>
<td>$\top\alpha$</td>
<td>$\ominus^{\top\alpha} (\ominus (I_p, \top\alpha))$</td>
<td>$I_{\text{ap}}$</td>
</tr>
<tr>
<td>$P$</td>
<td>$[0, \frac{1}{2}]$</td>
<td>$[1, 0]$</td>
<td>$\ominus^{\text{ap}} ([0, \frac{1}{2}], [1, 0])$</td>
<td>$[\frac{1}{2}, \frac{3}{2}]$</td>
</tr>
</tbody>
</table>

The result of permit-overrides combining algorithm under Belnap logic is $\text{ff}$ and under $D$-algebra is $\{ p, d \}$. Under Bruns et al. approach using Belnap logic, the access is denied while under Ni et al. approach using $D$-algebra, a conflict occurs. Both Bruns et al. and Ni et al. claim that their approaches fit with XACML 2.0 [12]. Moreover $D$-algebra claims that it fits with XACML 3.0 [14]. However based on XACML 2.0 the result should be Indeterminate and based on XACML 3.0 the result should be Indeterminate Deny Permit and neither Belnap logic nor $D$-algebra fits the specifications. We have illustrated that Belnap logic and $D$-algebra in some cases give different result with the XACML specification. Conversely, our approach gives consistent result based on the XACML 3.0 [14] and on the XACML 2.0 [12].

## 5 Conclusion

We have shown the formalization of XACML version 3.0 step by step. We believe that with our approach, the user can understand better about how XACML works especially in the behaviour of combining algorithms. We show two approaches to formalizing standard XACML combining algorithms, i.e., using $V_6$ and $P$. To guard against modelling artifacts, we formally prove the equivalence of these approaches.

The pairwise policy values approach is useful in defining new combining algorithms. For example, suppose we have a new combining algorithm “all permit”, i.e., the result
of composing policies is permit if all policies give permit values, otherwise it is deny. Using pairwise policy values approach the result of composing a set of policies values $S$ is permit $\langle 0,1 \rangle$ if $\min_{\leq_p} (S) = \langle 0, 1 \rangle = \max_{\leq_p} (S)$, otherwise, it is deny $\langle 1,0 \rangle$.

Ni et al. proposes a $D$-algebra over a set of decisions for XACML combining algorithms in [13]. However, there are some mismatches between their results and the XACML specifications. Their formulations are inconsistent based both on the XACML 2.0 [12] and on the XACML 3.0 [14].

Both Belnap four-valued logic and $D$-Algebra have a conflict value. In XACML, the conflict will never occur because the combining algorithms do not allow that. Conflict value might be a good indication that the policies are not well design. We propose an extended $P$ which captures a conflict value in Appendix A.

References

2. XML 1.0 specification. w3.org. retrieved 2010-08-22. [http://www.w3.org/TR/xml/]

3 The detail of all of XACML decisions under $D$-algebra can be seen in extended paper at [http://www2.imm.dtu.dk/~cdpu/Papers/the_logic_of_XACML-extended.pdf](http://www2.imm.dtu.dk/~cdpu/Papers/the_logic_of_XACML-extended.pdf)
A Extended Pairwise Policy Values

We add three values into $P$, i.e. deny with indeterminate permit ($[1, \frac{1}{2}]$), permit with indeterminate deny ($[\frac{1}{2}, 1]$) and conflict ($[1, 1]$) and we call the extended pairwise policy values $P_9 = P \cup \{[1, \frac{1}{2}], [\frac{1}{2}, 1], [1, 1]\}$. The extended pairwise policy values shows all possible combination of pairwise policy values. The ordering of $P_9$ is illustrated in Figure 5.

\[ [1, 1] = \top_d \top_p \]
\[ [1, \frac{1}{2}] = \top_d I_p \]
\[ \frac{1}{2}, 1 = I_d \top_p \]
\[ [1, 0] = \top_d \frac{1}{2} \frac{1}{2} = I_d I_p \]
\[ 0, 1 = \top_p \]
\[ [\frac{1}{2}, 0] = I_d \]
\[ 0, \frac{1}{2} = I_p \]
\[ [0, 0] = \perp \]

Figure 5. Nine-Valued Lattice

We can see that $P_9$ forms a lattice (we call this $\mathcal{L}_9$) where the top element is $[1, 1]$ and the bottom element is $[0, 0]$. The ordering of this lattice is the same as $\sqsubseteq_P$ where the greatest lower bound and the least upper bound for $S \subseteq P$ are defined as follows:

\[ \bigwedge_{\mathcal{L}_9} S = \max_{\sqsubseteq_P} (S) \text{ and } \bigvee_{\mathcal{L}_9} S = \min_{\sqsubseteq_P} (S) \]