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 formalization based either on Belnap logic or on $\mathcal{D}$-algebra.

1 Introduction

XACML (eXtensible Access Control Markup Language) is an approved OASIS [1] Standard access control language [1,14]. 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.

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 formulae 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

XACML Policy Components		
`PolicySet`	$::= \langle \texttt{Target}, \langle \texttt{PolicySet}_1, \ldots, \texttt{PolicySet}_m \rangle, \theta \rangle$	
	$\mid \langle \texttt{Target}, \langle \texttt{Policy}_1, \ldots, \texttt{Policy}_m \rangle, \theta \rangle$	where $m \geq 0$
`Policy`	$::= \langle \texttt{Target}, \langle \texttt{Rule}_1, \ldots, \texttt{Rule}_m \rangle, \theta \rangle$	where $m \geq 1$
`Rule`	$::= \langle$ *Effect*, `Target`, `Condition` $\rangle$	
`Condition`	::= *propositional formulae*	
`Target`	::= **Null**	
	$\mid \texttt{AnyOf}_1 \wedge \ldots \wedge \texttt{AnyOf}_m$	where $m \geq 1$
`AnyOf`	$::= \texttt{AllOf}_1 \vee \ldots \wedge \texttt{AllOf}_m$	where $m \geq 1$
`AllOf`	$::= \texttt{Match}_1 \wedge \ldots \wedge \texttt{Match}_m$	where $m \geq 1$
`Match`	$::= \Phi(\alpha)$	
Φ	::= **subject** | **action** | **resource** | **enviroment**	
α	::= *attribute value*	
θ	::= $\mathbf{p-o} \mid \mathbf{d-o} \mid \mathbf{f-a} \mid \mathbf{o-1-a}$	
Effect	::= $\mathbf{d} \mid \mathbf{p}$	
XACML Request Component		
`Request`	$::= \{ A_1, \ldots, A_m \}$	where $m \geq 1$
A	::= $\Phi(\alpha)$ | *external state*	

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 formulae 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 $\mathcal{L}_3 = \langle V_3, \leq \rangle$ 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. $\mathcal{L}_3$) by $\bigsqcap S$ and $\bigsqcup S$, respectively. Recall that $\bigsqcap \emptyset = \top$ and $\bigsqcup \emptyset = \bot$.

We use $[\![.]\!]$ 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 2. Mapping V_3 into XACML Evaluation Values

V_3	`Match` and `Target` value	`Condition` value	`Rule`, `Policy` and `PolicySet` value
$\top$	match	true	applicable (either deny or permit)
$\bot$	not match	false	not applicable
I	indeterminate	indeterminate	indeterminate

2.2.2 Match Evaluation

A `Match` element $\mathcal{M}$ is an attribute value that the request should fulfill. Given a `Request` component $\mathcal{Q}$, the evaluation of `Match` element is as follows:

$$[\![\mathcal{M}]\!](\mathcal{Q}) = \begin{cases} \top & \mathcal{M} \in \mathcal{Q} \\ \bot & \mathcal{M} \notin \mathcal{Q} \\ I & \text{there is an error during the evaluation} \end{cases} \tag{1}$$

2.2.3 Target Evaluation

Let $\mathcal{M}$ be a `Match`, $\mathcal{A} = \mathcal{M}_1 \wedge \ldots \wedge \mathcal{M}_m$ be an `AllOf`, $\mathcal{E} = \mathcal{A}_1 \vee \ldots \vee \mathcal{A}_n$ be an `AnyOf`, $\mathcal{T} = \mathcal{E}_1 \wedge \ldots \wedge \mathcal{E}_o$ be a `Target` and $\mathcal{Q}$ be a `Request`. Then, the evaluations of `AllOf`, `AnyOf`, and `Target` are as follows:

$$[\![\mathcal{A}]\!](\mathcal{Q}) = \bigsqcap_{i=1}^{m} [\![\mathcal{M}_i]\!](\mathcal{Q}) \tag{2}$$

$$[\![\mathcal{E}]\!](\mathcal{Q}) = \bigsqcup_{i=1}^{n} [\![\mathcal{A}_i]\!](\mathcal{Q}) \tag{3}$$

$$[\![\mathcal{T}]\!](\mathcal{Q}) = \bigsqcap_{i=1}^{o} [\![\mathcal{E}_i]\!](\mathcal{Q}) \tag{4}$$

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

$$[\![\mathcal{T}]\!](\mathcal{Q}) = \bigsqcap \bigsqcup \bigsqcap [\![\mathcal{M}]\!](\mathcal{Q}) \tag{5}$$

An empty `Target` – indicated by $\mathbf{Null}$ – is always evaluated to $\top$.

2.2.4 Condition Evaluation

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

$$[\![\mathcal{C}]\!](\mathcal{Q}) = eval(\mathcal{C}, \mathcal{Q}) \tag{6}$$

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 $\top_{\mathbf{p}}$ and $\top_{\mathbf{d}}$, 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 ($I_{\mathbf{d}}$): 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 ($I_{\mathbf{p}}$): 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 ($I_{\mathbf{dp}}$): an indeterminate from a policy which could have effect either deny or permit.

We extend the set V_3 to $V_6 = \{ \top_{\mathbf{p}}, \top_{\mathbf{d}}, I_{\mathbf{d}}, I_{\mathbf{p}}, I_{\mathbf{dp}}, \bot \}$ and we use V_6 for for XACML policies evaluations.

2.2.6 Rule Evaluation

Let $\mathcal{R} = \langle *, \mathcal{T}, \mathcal{C} \rangle$ be a `Rule` and $\mathcal{Q}$ be a `Request`. Then, the evaluation of `Rule` is determined as follows:

$$[\![\mathcal{R}]\!](\mathcal{Q}) = \begin{cases} \top_* & [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \top \\ \bot & \big([\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } [\![\mathcal{C}]\!](\mathcal{Q}) = \bot\big) \text{ or } [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \\ I_* & \text{otherwise} \end{cases} \tag{7}$$

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

$$F \rightsquigarrow G = \begin{cases} G & F = \top \\ F & \text{otherwise} \end{cases} \tag{8}$$

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

$$\sigma(X, *) = \begin{cases} X & X = \bot \\ X_* & \text{otherwise} \end{cases} \tag{9}$$

Proposition 1. *Let $\mathcal{R} = \langle *, \mathcal{T}, \mathcal{C} \rangle$ be a* `Rule` *and $\mathcal{Q}$ be a* `Request`*. Then, the following equation holds*

$$[\![\mathcal{R}]\!](\mathcal{Q}) = \sigma\left([\![\mathcal{T}]\!](\mathcal{Q}) \rightsquigarrow [\![\mathcal{C}]\!](\mathcal{Q}), *\right) \tag{10}$$

2.2.7 Policy Evaluation

The standard evaluation of Policy element taken from [14] is as follows

`Target` value	`Rule` value	`Policy` Value
match	At least one `Rule` value is applicable	Specified by the combining algorithm
match	All `Rule` values are not applicable	not applicable
match	At least one `Rule` value is indeterminate	Specified by the combining algorithm
not match	Don't care	not applicable
indeterminate	Don't care	indeterminate

Let $\mathcal{P} = \langle \mathcal{T}, \mathbb{R}, \theta \rangle$ be a `Policy` where $\mathbb{R} = \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$. Let $\mathcal{Q}$ be a `Request` and $\mathbb{R}' = \langle [\![\mathcal{R}_1]\!](\mathcal{Q}), \ldots, [\![\mathcal{R}_n]\!](\mathcal{Q}) \rangle$. The evaluation of `Policy` is defined as follows:

$$[\![\mathcal{P}]\!](\mathcal{Q}) = \begin{cases} I_* & [\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_\theta(\mathbb{R}') \in \{\ \top_*,\ I_*\ \} \\ \bot & [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \text{ or} \\ & [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \forall \mathcal{R}_i : [\![\mathcal{R}_i]\!](\mathcal{Q}) = \bot \\ \bigoplus_\theta(\mathbb{R}') & \text{otherwise} \end{cases} \tag{11}$$

Note 1. The combining algorithms denoted by $\bigoplus$ is explained in Section 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 $\mathcal{PS} = \langle \mathcal{T}, \mathbb{P}, \theta \rangle$ be a `PolicySet` where $\mathbb{P} = \langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$. Let $\mathcal{Q}$ be a `Request` and $\mathbb{P}' = \langle [\![\mathcal{P}_1]\!](\mathcal{Q}), \ldots, [\![\mathcal{P}_n]\!](\mathcal{Q}) \rangle$. The evaluation of `PolicySet` is defined as follows:

$$[\![\mathcal{PS}]\!](\mathcal{Q}) = \begin{cases} I_* & [\![\mathcal{T}]\!](\mathcal{Q}) = I \text{ and } \bigoplus_\theta(\mathbb{P}') \in \{\ \top_*,\ I_*\ \} \\ \bot & [\![\mathcal{T}]\!](\mathcal{Q}) = \bot \text{ or} \\ & [\![\mathcal{T}]\!](\mathcal{Q}) = \top \text{ and } \forall \mathcal{P}_i : [\![\mathcal{P}_i]\!](\mathcal{Q}) = \bot \\ \bigoplus_\theta(\mathbb{P}') & \text{otherwise} \end{cases} \tag{12}$$

2.3 Example

The following example simulates briefly how a policy is built using the abstraction. The example is motivated by [7,9] 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.

```
PS_patient = <Null, <P_patient_record, P_medical_record>, po>
P_patient_record = <Null, <RP1, RP2, RP3>, do>
P_medical_record = <Null, <RM1, RM2>, do>

RP1 =
< p,
  subject(patient) /\ action(read) /\ resource(patient_record),
  patient(id,X) /\ patient_record(id,Y) /\
  (X = Y \/ (age(Y) < 18 /\ guardian(X,Y))>

RP2 =
< p,
  subject(patient) /\ action(write) /\ resource(patient_survey),
  patient(id,X) /\ patient_survey(id, X)>

RP3=
< p,
  (subject(doctor) \/ subject(nurse)) /\ action(read) /\ resource(patient_record),
  true>

RM1 =
< p,
  subject(doctor) /\ action(write) /\ resource(medical_record),
  doctor(id,X) /\ patient(id,Y) /\ medical_record(id, Y) /\ patient_doctor(Y,X)>

RM1 =
< d,
  subject(doctor) /\ action(write) /\ resource(medical_record),
  doctor(id,X), patient(id,Y), medical_record(id, Y), not patient_doctor(Y,X)>
```

Figure 1. The XACML Policy for Patient Policy

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

```
{ 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.[2]

3.1 Pairwise Policy Values

In V_6 we define the truth values of XACML components by extending $\top$ to $\top_{\mathbf{p}}$ and $\top_{\mathbf{d}}$ and I to $I_{\mathbf{d}}, I_{\mathbf{p}}$ and $I_{\mathbf{dp}}$. 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 $\mathbf{1}$ represents an applicable value (either deny or permit), $\mathbf{\frac{1}{2}}$ represents indeterminate value and $\mathbf{0}$ means there is no applicable value. In each tuple, the first element represents the `Deny` value ($\top_{\mathbf{d}}$) and the later represents `Permit` value ($\top_{\mathbf{p}}$). We can say $[0,0]$ for not applicable ($\bot$) because neither `Deny` nor `Permit` is applicable, $[1,0]$ for applicable with deny effect ($\top_{\mathbf{d}}$) because only `Deny` value is applicable, $[\frac{1}{2},0]$ for $I_{\mathbf{d}}$ because the `Deny` part is indeterminate, $[\frac{1}{2},\frac{1}{2}]$ for $I_{\mathbf{dp}}$ because both `Deny` and `Permit` have indeterminate values. The conversion applies also for `Permit`.

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

[2] 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 $\delta : V_6 \rightarrow \mathbf{P}$ as a mapping function that maps V_6 into $\mathbf{P}$ as follows:

$$\delta(X) = \begin{cases} [0,0] & X = \bot \\ [1,0] & X = \top_{\mathbf{d}} \\ [0,1] & X = \top_{\mathbf{p}} \\ [\frac{1}{2},0] & X = I_{\mathbf{d}} \\ [0,\frac{1}{2}] & X = I_{\mathbf{p}} \\ [\frac{1}{2},\frac{1}{2}] & X = I_{\mathbf{dp}} \end{cases} \tag{13}$$

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

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

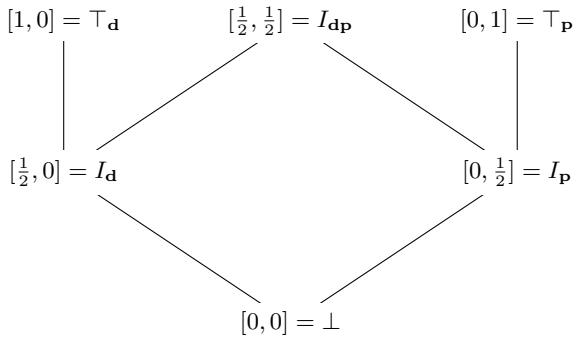

Figure 2. The Partial Ordered Set $\boldsymbol{P}_{\mathbf{P}}$ for Pairwise Policy Values

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

$$Max_{\sqsubseteq_{\mathbf{P}}}(S) = [max(\{\, d(X) \mid X \in S \,\}), max(\{\, p(X) \mid X \in S \,\})] \tag{14}$$

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

$$Min_{\sqsubseteq_{\mathbf{P}}}(S) = [min(\{\, d(X) \mid X \in S \,\}), min(\{\, p(X) \mid X \in S \,\})] \tag{15}$$

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 $\top_{\mathbf{p}}$ then the result is $\top_{\mathbf{p}}$,

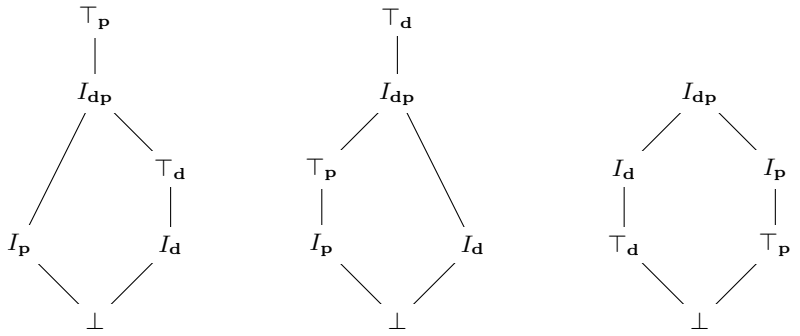

Figure 3. The Lattice $\mathcal{L}_{\mathbf{p-o}}$ for The Permit-Overrides Combining Algorithm (left), The Lattice $\mathcal{L}_{\mathbf{d-o}}$ for The Deny-Overrides Combining Algorithm (middle) and The Lattice $\mathcal{L}_{\mathbf{o-1-a}}$ for The Only-One-Applicable Combining Algorithm (right)

2. otherwise, if any decision is $I_{\mathbf{dp}}$ then the result is $I_{\mathbf{dp}}$,
3. otherwise, if any decision is $I_{\mathbf{p}}$ and another decision is $I_{\mathbf{d}}$ or $\top_{\mathbf{d}}$, then the result is $I_{\mathbf{dp}}$,
4. otherwise, if any decision is $I_{\mathbf{p}}$ then the result is $I_{\mathbf{p}}$,
5. otherwise, if decision is $\top_{\mathbf{d}}$ then the result is $\top_{\mathbf{d}}$,
6. otherwise, if any decision is $I_{\mathbf{d}}$ then the result is $I_{\mathbf{d}}$,
7. otherwise, the result is $\bot$.

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

Definition 1. *The permit-overrides combining algorithm $\bigoplus_{\mathbf{p-o}}^{V_6}$ 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:*

$$\bigoplus_{\mathbf{p-o}}^{V_6}(S) = \bigsqcup_{\mathbf{p-o}} S' \tag{16}$$

The permit-overrides combining algorithm can also be expressed under $\mathbf{P}$. 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 $I_{\mathbf{dp}}$ 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 $\bigoplus_{\mathbf{p-o}}^{\mathbf{P}}$ is a mapping function from a sequence of $\mathbf{P}$ elements into an element in $\mathbf{P}$ 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 $\mathbf{P}$ *as follows:*

$$\bigoplus_{\mathbf{p-o}}^{\mathbf{P}}(S) = \begin{cases} [0,1] & Max_{\sqsubseteq_{\mathbf{P}}}(S') = [\_,1] \\ [\frac{1}{2},\frac{1}{2}] & Max_{\sqsubseteq_{\mathbf{P}}}(S') = [D,\frac{1}{2}], D \geq \frac{1}{2} \\ Max_{\sqsubseteq_{\mathbf{P}}}(S') & \textit{otherwise} \end{cases} \tag{17}$$

Proposition 2. *Let* S *be a sequence of policy values in* V_6*. Then*

$$\delta(\bigoplus_{\mathbf{p-o}}^{V_6}(S)) = \bigoplus_{\mathbf{p-o}}^{\mathbf{P}}(\delta(S))$$

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 $\top_{\mathbf{d}}$ then the result is $\top_{\mathbf{d}}$,
2. otherwise, if any decision is $I_{\mathbf{dp}}$ then the result is $I_{\mathbf{dp}}$,
3. otherwise, if any decision is $I_{\mathbf{d}}$ and another decision is $I_{\mathbf{p}}$ or $\top_{\mathbf{p}}$, then the result is $I_{\mathbf{dp}}$,
4. otherwise, if any decision is $I_{\mathbf{d}}$ then the result is $I_{\mathbf{d}}$,
5. otherwise, if decision is $\top_{\mathbf{p}}$ then the result is $\top_{\mathbf{p}}$,
6. otherwise, if any decision is $I_{\mathbf{p}}$ then the result is $I_{\mathbf{p}}$,
7. otherwise, the result is $\bot$.

We call $\mathcal{L}_{\mathbf{d-o}} = (V_6, \sqsubseteq_{\mathbf{d-o}})$ for the lattice using the deny-overrides combining algorithm where $\sqsubseteq_{\mathbf{d-o}}$ is the ordering depicted in Figure 3. The least upper bound operator for $\mathcal{L}_{\mathbf{d-o}}$ is denoted by $\bigsqcup_{\mathbf{d-o}}$.

Definition 3. *The deny-overrides combining algorithm* $\bigoplus_{\mathbf{d-o}}^{V_6}$ *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 deny-overrides combining algorithm under* V_6 *as follows:*

$$\bigoplus_{\mathbf{d-o}}^{V_6}(S) = \bigsqcup_{\mathbf{d-o}} S' \tag{18}$$

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

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

$$\bigoplus_{\mathbf{d-o}}^{\mathbf{P}}(S) = \begin{cases} [1,0] & Max_{\sqsubseteq_{\mathbf{P}}}(S') = [1,\_] \\ [\frac{1}{2},\frac{1}{2}] & Max_{\sqsubseteq_{\mathbf{P}}}(S') = [\frac{1}{2},P], P \geq \frac{1}{2} \\ Max_{\sqsubseteq_{\mathbf{P}}}(S') & \textit{otherwise} \end{cases} \tag{19}$$

Proposition 3. *Let S be a sequence of policy values in V_6. Then*

$$\delta(\bigoplus_{\mathbf{d-o}}^{V_6}(S)) = \bigoplus_{\mathbf{d-o}}^{\mathbf{P}}(\delta(S))$$

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 $\mathbf{P}$ are defined below.

Definition 5 (First-Applicable Combining Algorithm). *The first-applicable combining algorithm $\bigoplus_{\mathbf{f-a}}^{V_6}$ 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:*

$$\bigoplus_{\mathbf{f-a}}^{V_6}(S) = \begin{cases} s_i & \exists i : s_i \neq \bot \textit{ and } \forall j < i : s_j = \bot \\ \bot & \textit{otherwise} \end{cases} \tag{20}$$

Definition 6. *The first-applicable combining algorithm $\bigoplus_{\mathbf{f-a}}^{\mathbf{P}}$ is a mapping function from a sequence of $\mathbf{P}$ elements into an element in $\mathbf{P}$ as the result of composing policies. Let $S = \langle s_1, \ldots, s_n \rangle$ be a sequence of policy values in $\mathbf{P}$. We define the first applicable combining algorithm under $\mathbf{P}$ as follows:*

$$\bigoplus_{\mathbf{f-a}}^{\mathbf{P}}(S) = \begin{cases} s_i & \exists i : s_i \neq [0,0] \textit{ and } \forall j < i : s_j = [0,0] \\ [0,0] & \textit{otherwise} \end{cases} \tag{21}$$

Proposition 4. *Let S be a sequence of policy values in V_6. Then*

$$\delta(\bigoplus_{\mathbf{f-a}}^{V_6}(S)) = \bigoplus_{\mathbf{f-a}}^{\mathbf{P}}(\delta(S))$$

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 $\mathcal{L}_{\mathbf{o-1-a}} = (V_6, \sqsubseteq_{\mathbf{o-1-a}})$ for the lattice using the only-one-applicable combining algorithm where $\sqsubseteq_{\mathbf{o-1-a}}$ is the ordering depicted in Figure 3. The least upper bound operator for $\mathcal{L}_{\mathbf{o-1-a}}$ is denoted by $\bigsqcup_{\mathbf{o-1-a}}$.

Definition 7. *The only-one-applicable combining algorithm* $\bigoplus^{V_6}_{\mathbf{o-1-a}}$ *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 only-one-applicable combining algorithm under* V_6 *as follows*

$$\bigoplus_{\mathbf{o-1-a}}^{V_6}(S) = \begin{cases} I_{\mathbf{d}} & \exists i,j : i \neq j, s_i = s_j = \top_{\mathbf{d}} \textit{ and} \\ & \forall k : s_k \neq \top_{\mathbf{d}} \rightarrow s_k = \bot \\ I_{\mathbf{p}} & \exists i,j : i \neq j, s_i = s_j = \top_{\mathbf{p}} \textit{ and} \\ & \forall k : s_k \neq \top_{\mathbf{p}} \rightarrow s_k = \bot \\ \bigsqcup_{\mathbf{o-1-a}} S' & \textit{otherwise} \end{cases} \tag{22}$$

The only-one-applicable combining algorithm also can be expressed under $\mathbf{P}$. 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 $I_{\mathbf{dp}}$ (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 $I_{\mathbf{d}}$ (or $I_{\mathbf{p}}$). 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* $\bigoplus^{\mathbf{P}}_{\mathbf{o-1-a}}$ *is a mapping function from a sequence of* $\mathbf{P}$ *elements into an element in* $\mathbf{P}$ *as the result of composing policies. Let* $S = \langle s_1, \ldots, s_n \rangle$ *be a sequence of policy values in* $\mathbf{P}$ *and* $S' = \{ s_1, \ldots, s_n \}$*. We define only-one-applicable combining algorithm under* $\mathbf{P}$ *as follows*

$$\bigoplus_{\mathbf{o-1-a}}^{\mathbf{P}}(S) = \begin{cases} [\frac{1}{2}, \frac{1}{2}] & Max_{\sqsubseteq_{\mathbf{P}}}(S') = [D,P], D, P \geq \frac{1}{2} \\ [\frac{1}{2}, 0] & Max_{\sqsubseteq_{\mathbf{P}}}(S') = [D,0], D \geq \frac{1}{2} \textit{ and} \\ & \exists i,j : i \neq j, d(s_i), d(s_j) \geq \frac{1}{2} \\ [0, \frac{1}{2}] & Max_{\sqsubseteq_{\mathbf{P}}}(S') = [0,P], P \geq \frac{1}{2} \textit{ and} \\ & \exists i,j : i \neq j, p(s_i), p(s_j) \geq \frac{1}{2} \\ Max_{\sqsubseteq_{\mathbf{P}}}(S') & \textit{otherwise} \end{cases} \tag{23}$$

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

$$\delta(\bigoplus_{\mathbf{o-1-a}}^{V_6}(S)) = \bigoplus_{\mathbf{o-1-a}}^{\mathbf{P}}(\delta(S))$$

4 Related Work

We will focus the discussion on the formalization of XACML using Belnap logic [4] and $\mathcal{D}$-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 $\mathbf{four} = \{ \top\top, \mathbf{tt}, \mathbf{ff}, \bot\bot \}$. There are two orderings in Belnap logic, i.e., the knowledge ordering ($\leq_k$) and the truth ordering ($\leq_t$) (see Figure 4).

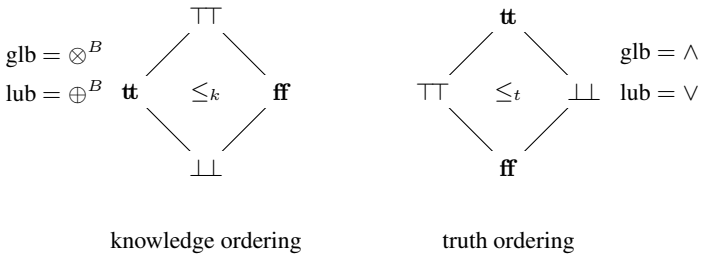

Figure 4. Bi-lattice of Belnap Four-Valued Logic

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 **tt** when the policy is granted or access permitted, **ff** 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 $y, z \in \mathbf{four}$. 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**: $(p \oplus^B q)[\top\top \mapsto \mathbf{ff}]$
- **first-applicable**: $p > q$
- **only-one-applicable**: $(p \oplus^B q) \oplus^B ((p \oplus^B \neg p) \otimes^B (q \oplus^B \neg q))$

Bruns *et al.* suggested that the indeterminate value is treated as $\top\top$. However, with indeterminate as $\top\top$, 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 $(p \oplus^B q)[\top\top \mapsto \mathbf{ff}] = (\mathbf{tt} \oplus^B \top\top)[\top\top \mapsto \mathbf{ff}] = \top\top[\top\top \mapsto \mathbf{ff}] = \mathbf{ff}$. Based on the XACML 2.0 [12] and the XACML 3.0 [14], the result of permit-overrides combining algorithm should be permit (**tt**). However, based on Belnap four-valued logic, the result is deny (**ff**).

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 $\perp\!\!\perp$ and sometimes as $\top\!\!\top$ [5], but there is no explanation about under which circumstances that indeterminate is treated as $\top\!\!\top$ or as $\perp\!\!\perp$. The treatment of indeterminate as $\top\!\!\top$ is too strong because indeterminate does not always contains information about deny and permit in the same time. Only $I_{\mathbf{dp}}$ contains information both deny and permit. However, $I_{\mathbf{d}}$ and $I_{\mathbf{p}}$ only contain information only about deny and permit, respectively. Even so, the value $\perp\!\!\perp$ 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. $\top\!\!\top$).

4.2 XACML Semantics under $\mathcal{D}$-Algebra

Ni *et al.* in [13] define $\mathcal{D}$-algebra as a decision set together with some operations on it.

Definition 9 ($\mathcal{D}$-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 $\mathcal{D}$, and $\oplus^{\mathcal{D}}, \otimes^{\mathcal{D}}$ be binary operations on elements in D. A $\mathcal{D}$-algebra is an algebraic structure $\langle D, \neg, \oplus^{\mathcal{D}}, \otimes^{\mathcal{D}}, 0\rangle$ closed on $\neg, \oplus^{\mathcal{D}}, \otimes^{\mathcal{D}}$ and satisfying the following axioms:*

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

In order to write formulae in a compact form, for $x, y \in \mathcal{D}$, $x \odot^{\mathcal{D}} y = \neg(\neg x \oplus^{\mathcal{D}} \neg y)$ and $x \ominus^{\mathcal{D}} y = x \odot^{\mathcal{D}} \neg y$.

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

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

There are two values which are not in XACML, i.e. $\emptyset$ and $\{\, \mathbf{p}, \mathbf{d} \,\}$. Simply we say $\emptyset$ for empty policy (or there is no policy) and $\{\, \mathbf{p}, \mathbf{d} \,\}$ for a conflict.

The composition function of permit-overrides using $\mathcal{D}$-Algebra is as follows:

$$\begin{aligned} f_{po}(x,y) = (x \oplus^{\mathcal{D}} y) \\ \ominus^{\mathcal{D}}(((x \otimes^{\mathcal{D}} \{\, \mathbf{p} \,\}) \oplus^{\mathcal{D}} (y \otimes^{\mathcal{D}} \{\, \mathbf{p} \,\})) \odot^{\mathcal{D}} \{\, \mathbf{d}, \tfrac{\mathbf{n}}{\mathbf{a}} \,\}) \\ \ominus^{\mathcal{D}}(\neg((x \odot^{\mathcal{D}} y) \otimes^{\mathcal{D}} \{\, \tfrac{\mathbf{n}}{\mathbf{a}} \,\}) \odot^{\mathcal{D}} \{\, \tfrac{\mathbf{n}}{\mathbf{a}} \,\} \odot^{\mathcal{D}} \neg((x \otimes^{\mathcal{D}} \emptyset) \oplus^{\mathcal{D}} (y \otimes^{\mathcal{D}} \emptyset))) \end{aligned}$$

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 polices under Belnap logic, $\mathcal{D}$-algebra, V_6 and $\mathbf{P}$.

Table 3. Result of Permit-Overrides Combining Algorithm for Composing Two Policies P_1 and P_2 where P_1 is Indeterminate Permit and P_2 is Deny Under Various Logic

Logic	P_1	P_2	Permit-Overrides Function	Result
Belnap logic	$\top\top$	$\mathbf{ff}$	$(\top\top \oplus^{\mathcal{B}} \mathbf{ff})[\top\top \mapsto \mathbf{ff}]$	$\mathbf{ff}$
$\mathcal{D}$-algebra	$\{\, \mathbf{p}, \frac{\mathbf{n}}{\mathbf{a}} \,\}$	$\{\, \mathbf{d} \,\}$	$f_{po}(\{\, \mathbf{p}, \frac{\mathbf{n}}{\mathbf{a}} \,\}, \{\, \mathbf{d} \,\})$	$\{\, \mathbf{p}, \mathbf{d} \,\}$
V_6	$I_{\mathbf{p}}$	$\top_{\mathbf{d}}$	$\bigoplus^{V_6}_{\mathbf{p-o}}(\langle I_{\mathbf{p}}, \top_{\mathbf{d}} \rangle)$	$I_{\mathbf{dp}}$
$\mathbf{P}$	$[0, \frac{1}{2}]$	$[1, 0]$	$\bigoplus^{\mathbf{P}}_{\mathbf{p-o}}(\langle [0, \frac{1}{2}], [1, 0] \rangle)$	$[\frac{1}{2}, \frac{1}{2}]$

The result of permit-overrides combining algorithm under Belnap logic is $\mathbf{ff}$ and under $\mathcal{D}$-algebra is $\{\, \mathbf{p}, \mathbf{d} \,\}$. Under Bruns *et al.* approach using Belnap logic, the access is denied while under Ni *et al.* approach using $\mathcal{D}$-algebra, a conflict occurs. Both Bruns *et al.* and Ni *et al.* claim that their approaches fit with XACML 2.0 [12]. Moreover $\mathcal{D}$-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 $\mathcal{D}$-algebra fits the specifications. We have illustrated that Belnap logic and $\mathcal{D}$-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 $\mathbf{P}$. 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 ([0,1]) if $Min_{\sqsubseteq_{\mathbf{P}}}(S) = [0,1] = Max_{\sqsubseteq_{\mathbf{P}}}(S)$, otherwise, it is deny ([1,0]).

Ni *et al.* proposes a $\mathcal{D}$-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].[3]

Both Belnap four-valued logic and $\mathcal{D}$-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 $\mathbf{P}$ which captures a conflict value in Appendix A.

References

1. eXtensible Access Control Markup Language (XACML). `http://xml.coverpages.org/xacml.html`.
2. XML 1.0 specification. w3.org. retrieved 2010-08-22. `http://www.w3.org/TR/xml/`.
3. Gail-Joon Ahn, Hongxin Hu, Joohyung Lee, and Yunsong Meng. Reasoning about xacml policy descriptions in answer set programming (preliminary report). In *13th International Workshop on Nonmonotonic Reasoning (NMR 2010)*, 2010.
4. N.D. Belnap. A useful four-valued logic. In G. Epstein and J.M. Dunn, editors, *Modern Uses of Multiple-Valued Logic*, pages 8–37. D. Reidel, Dordrecht, 1977.
5. Glenn Bruns, Daniel S Dantas, and Michael Huth. A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, FMSE '07, pages 12–21, New York, NY, USA, 2007. ACM.
6. Glenn Bruns and Michael Huth. Access-control via belnap logic: Effective and efficient composition and analysis. In *21st IEEE Computer Security Foundations Symposium*, June 2008.
7. Mark Evered and Serge Bögeholz. A case study in access control requirements for a health information systems. In *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation - Volume 32*, ACSW Frontiers '04, pages 53–61, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.
8. Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. *ACM Transaction on Information and System Security (TISSEC)*, 11(4):1 – 41, 2008.
9. Chris Hankin, Flemming Nielson, and Hanne Riis Nielson. Advice from belnap policies. *Computer Security Foundations Symposium, IEEE*, 0:234–247, 2009.
10. Vladimir Kolovski and James Hendler. Xacml policy analysis using description logics. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, 2007.
11. Vladimir Kolovski, James Hendler, and Bijan Parsia. Formalizing xacml using defeasible description logics. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, 2007.

[3] The detail of all of XACML decisions under $\mathcal{D}$-algebra can be seen in extended paper at `http://www2.imm.dtu.dk/~cdpu/Papers/the_logic_of_XACML-extended.pdf`.

12. Tim Moses. eXtensible Access Control Markup Language (XACML) version 2.0. Technical report, OASIS, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, August 2010.
13. Qun Ni, Elisa Bertino, and Jorge Lobo. D-algebra for composing access control policy decisions. In *ASIACCS '09: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 298–309, New York, NY, USA, 2009. ACM.
14. Erik Rissanen. eXtensible Access Control Markup Language (XACML) version 3.0 (committe specification 01). Technical report, OASIS, http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-03-en.pdf, August 2010.

A Extended Pairwise Policy Values

We add three values into $\mathbf{P}$, 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* $\mathbf{P}_9 = \mathbf{P} \cup \left\{ [1, \frac{1}{2}], [\frac{1}{2}, 1], [1, 1] \right\}$. The extended pairwise policy values shows all possible combination of pairwise policy values. The ordering of $\mathbf{P}_9$ is illustrated in Figure 5.

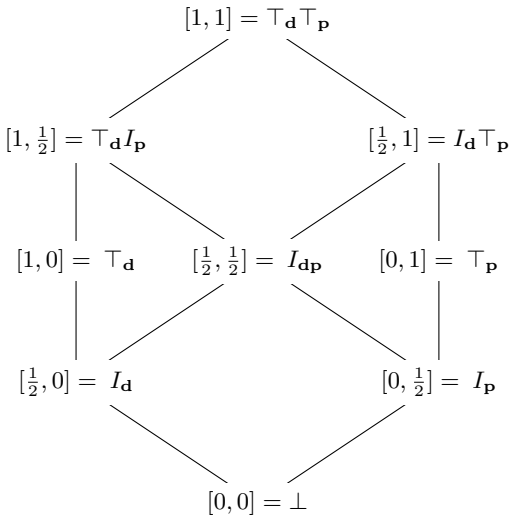

Figure 5. Nine-Valued Lattice

We can see that $\mathbf{P}_9$ 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 $\sqsubseteq_{\mathbf{P}}$ where the greatest lower bound and the least upper bound for $S \subseteq \mathbf{P}_9$ are defined as follows:

$$\bigsqcap_{\mathcal{L}_9} S = Max_{\sqsubseteq_{\mathbf{P}}}(S) \text{ and } \bigsqcup_{\mathcal{L}_9} S = Min_{\sqsubseteq_{\mathbf{P}}}(S)$$