Canonical Completeness in Lattice-Based Languages for Attribute-Based Access Control

Jason Crampton
Royal Holloway University of London
Egham, TW20 0EX, United Kingdom
jason.crampton@rhul.ac.uk

Conrad Williams
Royal Holloway University of London
Egham, TW20 0EX, United Kingdom
conrad.williams.2010@live.rhul.ac.uk

ABSTRACT

The study of canonically complete attribute-based access control (ABAC) languages is relatively new. A canonically complete language is useful as it is functionally complete and provides a "normal form" for policies. However, previous work on canonically complete ABAC languages requires that the set of authorization decisions is totally ordered, which does not accurately reflect the intuition behind the use of the allow, deny and not-applicable decisions in access control. A number of recent ABAC languages use a fourth value and the set of authorization decisions is partially ordered. In this paper, we show how canonical completeness in multi-valued logics can be extended to the case where the set of truth values forms a lattice. This enables us to investigate the canonical completeness of logics having a partially ordered set of truth values, such as Belnap logic, and show that ABAC languages based on Belnap logic, such as PBel, are not canonically complete. We then construct a canonically complete four-valued logic using connections between the generators of the symmetric group (defined over the set of decisions) and unary operators in a canonically suitable logic. Finally, we propose a new authorization language $\text{PTaCL}_4^{\leqslant}$, an extension of PTaCL, which incorporates a lattice-ordered decision set and is canonically complete. We then discuss how the advantages of $\text{PTaCL}_4^{\leqslant}$ can be leveraged within the framework of XACML.

CCS Concepts

•Security and privacy → Access control; Authorization; *Security requirements;* **•Software and its engineering → Specialized application languages;**

Keywords

XACML, PTaCL, decision operators, combining algorithms, functional completeness, canonical completeness

CODASPY'17, March 22-24, 2017, Scottsdale, AZ, USA

DOI: http://dx.doi.org/10.1145/3029806.3029808

1. INTRODUCTION

Access control is one of the most important security services in multi-user computer systems, providing a mechanism for constraining the interaction between (authenticated) users and protected resources. Generally, access control is implemented by an authorization service, which includes an *authorization decision function* for deciding whether a user request to access a resource (an "access request") should be permitted or not. In its simplest form an authorization decision function either returns an allow or deny decision.

Most implementations of access control use *authorization policies*, where a user request to access a resource is evaluated with respect to a policy that defines which requests are authorized. Many recent languages for the specification of authorization policies are designed for "open", distributed systems (rather than the more traditional "closed", centralized systems in which the set of users was assumed to be known in advance). Such languages do not necessarily rely on user identities to specify policies; instead, policies are defined in terms of other user and resource attributes. The most widely used attribute-based access control (ABAC) language is XACML [14, 17]. However, XACML suffers from poorly defined and counterintuitive semantics [10, 15], and is inconsistent in its articulation of policy evaluation. PTaCL is a more formal language for specifying authorization policies [6], providing a concise syntax for policy targets and precise semantics for policy evaluation.

Crampton and Williams [7] recently introduced the notion of *canonical completeness* for ABAC languages, showing that XACML and PTaCL are not canonically complete and developing a variant of PTaCL that is canonically complete. These results apply to languages that support three decision values, which are assumed to be totally ordered. However, there are certain situations where it is useful to have four decisions available, and some languages, such as PBel [4], BelLog [18] and Rumpole [12], use four decisions, which are partially ordered.

In this paper, we extend existing results to languages that support four decision values, which need not be totally ordered. We show that PBel [4], perhaps the best-known four-valued ABAC language, is not canonically complete. We then develop a canonically complete ABAC language, based on PTaCL syntax and semantics. The language is abstract, but its operators could be implemented as combining algorithms in XACML, thereby leveraging the features that XACML provides for specifying attribute-based

requests and targets, the evaluation of targets with respect to requests, and the storage and evaluation of policies.

In Section 2 we discuss background material and related work, which provides us with the primary motivation for this paper: to develop a canonically complete 4-valued logic to support a tree-structured authorization language. The main contributions of this work are:

- to extend Jobe's work on canonical completeness in multi-valued logics to the case where the set of truth values forms a lattice (Section 3.2);
- to establish that existing 4-valued logics are not canonically complete (Section 3.3);
- to construct a canonically complete 4-valued logic (Section 4);
- to construct a 4-valued, canonically complete authorization language for ABAC (Section 5).

We conclude the paper with a summary of our contributions and a discussion of future work.

2. BACKGROUND AND RELATED WORK

In this section, we summarize background material and related work, including tree-structured ABAC languages, canonical completeness, and four-valued languages for ABAC, thereby providing motivation for the work in the remainder of the paper.

2.1 Completeness in Multi-valued Logics

Let V be a set of truth values. The set of formulae $\Phi(L)$ that can be written in a (multi-valued) propositional logic $L = (V, \mathsf{Ops})$ is defined by V and the set of operators Ops. For brevity, we will write L when V and Ops are obvious from context.

Let V be a totally ordered set of m truth values, $\{0, \ldots, m-1\}$, with $0 < 1 < \cdots < m-1$. Then we say $L = (V, \mathsf{Ops})$ is *canonically suitable* if and only if there exist two formulas $\phi_{\max}$ and $\phi_{\min}$ of arity 2 in $\Phi(L)$ such that $\phi_{\max}(x, y)$ returns $\max\{x, y\}$ and $\phi_{\min}(x, y)$ returns $\min\{x, y\}$. We will usually write $\phi_{\max}$ and $\phi_{\min}$ using the infix operators $\curlyvee$ and $\curlywedge$ respectively.

Example 1. *Standard propositional logic with truth values 0 and 1, and operators $\vee$ and $\neg$, representing disjunction and negation, respectively, is canonically suitable: $\phi_{\max}(x, y)$ is simply $x \vee y$, while $\phi_{\min}(x, y)$ is $\neg(\neg x \vee \neg y)$ (that is, conjunction).*

A function $f : V^n \to V$ is completely specified by a truth table containing n columns and m^n rows. However, not every truth table can be represented by a formula in a given logic $L = (V, \mathsf{Ops})$. L is said to be *functionally complete* if for every function $f : V^n \to V$, there is a formula $\phi \in \Phi(L)$ of arity n whose evaluation corresponds to the truth table. In Section 2.2, we explain why we may regard a tree-structured authorization language as a logic defined by a set of decisions and the set of policy-combining operators. In this sense, XACML is not functionally complete [7], while PTaCL [6] and PBel are [4].

A *selection operator* $S^j_{(a_1,\ldots,a_n)}$ is an n-ary operator defined as follows:

$$S^j_{(a_1,\ldots,a_n)}(x_1, \ldots, x_n) = \begin{cases} j & \text{if } (x_1, \ldots, x_n) = (a_1, \ldots, a_n), \\ 0 & \text{otherwise.} \end{cases}$$

We will write $\mathbf{a}$ to denote the tuple $(a_1, \ldots, a_n) \in V^n$ when no confusion can occur. Note that $S^0_{\mathbf{a}}$ is the same for all $\mathbf{a} \in V^n$, and $S^0_{\mathbf{a}}(\mathbf{x}) = 0$ for all $\mathbf{x} \in V^n$. Illustrative examples of binary selection operators (for a 4-valued logic) are shown in Figure 1.

$S^1_{(0,2)}$	0	1	2	3
0	0	0	1	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

$S^2_{(1,1)}$	0	1	2	3
0	0	0	0	0
1	0	2	0	0
2	0	0	0	0
3	0	0	0	0

$S^3_{(3,0)}$	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	3	0	0	0

Figure 1: Selection operators $S^1_{(0,2)}$, $S^2_{(1,1)}$ and $S^3_{(3,0)}$

Selection operators play a central role in the development of canonically complete logics because an arbitrary function $f : V^n \to V$ can be expressed in terms of selection operators. Consider, for example, the function

$$f(x, y) = \begin{cases} 1 & \text{if } x = 0,\ y = 2, \\ 2 & \text{if } x = y = 1, \\ 3 & \text{if } x = 3,\ y = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Then it is easy to confirm that

$$f(x, y) \equiv S^1_{(0,2)}(x, y) \curlyvee S^2_{(1,1)}(x, y) \curlyvee S^3_{(3,0)}(x, y).$$

Moreover, $S^c_{(a,b)}(x, y) \equiv S^c_a(x) \curlywedge S^c_b(y)$ for any $a, b, c, x, y \in V$. Thus,

$$f(x, y) \equiv (S^1_0(x) \curlywedge S^1_2(y)) \curlyvee (S^2_1(x) \curlywedge S^2_1(y)) \curlyvee (S^3_3(x) \curlywedge S^3_0(y))$$

In other words, we can express f as the "disjunction" ($\curlyvee$) of "conjunctions" ($\curlywedge$) of unary selection operators.

More generally, given the truth table of function $f : V^n \to V$, we can write down an equivalent function in terms of selection operators. Specifically, let

$$A = \{\mathbf{a} \in V^n : f(\mathbf{a}) > 0\};$$

then, for all $\mathbf{x} \in V^n$,

$$f(\mathbf{x}) = \bigvarcurlyvee_{\mathbf{a} \in A} S^{f(\mathbf{x})}_{\mathbf{a}}(\mathbf{x}).$$

Jobe established a number of results connecting the functional completeness of a logic with the unary selection operators, summarized in the following theorem.

Theorem 1 (Jobe [9, Theorems 1, 2; Lemma 1]). *A logic L is functionally complete if and only if each unary selection operator is equivalent to some formula in L.*

The *normal form* of formula ϕ in a canonically suitable logic is a formula ϕ' that has the same truth table as ϕ and has the following properties:

- the only binary operators it contains are $\curlyvee$ and $\curlywedge$;
- no binary operator is included in the scope of a unary operator;
- no instance of $\curlyvee$ occurs in the scope of the $\curlywedge$ operator.

In other words, given a canonically suitable logic L containing unary operators $\sharp_1, \ldots, \sharp_\ell$, a formula in normal form has the form

$$\bigvee_{i=1}^{r} \bigwedge_{j=1}^{s} \sharp_{i,j} x_{i,j}$$

where $\sharp_{i,j}$ is a unary operator defined by composing the unary operators in $\sharp_1, \ldots, \sharp_\ell$. In the usual 2-valued propositional logic with a single unary operator (negation) this corresponds to disjunctive normal form.

A canonically suitable logic is *canonically complete* if every unary selection operator can be expressed in normal form. It is known that there are canonically suitable 3-valued logics that are: (i) not functionally complete [9, 11]; (ii) functionally complete but not canonically complete [9, Theorem 4]; and (iii) canonically complete (and hence functionally complete) [9, Theorem 6].

Jobe defined a canonically complete 3-valued logic [9]. The operators and the construction of the unary selection operators using these operators are given in Appendix A. The expression for f above could be expressed in normal form, providing we could find suitable unary operators for a 4-valued logic. In Section 4.3 we explain how to produce a suitable set of unary operators for an m-valued logic.

2.2 Tree-structured Languages for ABAC

Let D be a set of *authorization decisions*. Typically, we assume D contains the values 0, 1 and $\perp$ representing "deny", "allow" and "not-applicable", respectively. We call 0 and 1 *conclusive* decisions. Let $\oplus$ be an associative binary operator defined on D and $-$ be a unary operator defined on D. Then

- an *atomic policy* is a pair (t, d), where d is a decision in D and t is a *target predicate*;
- an atomic policy is a *policy*;
- if p and p' are policies, then $(t, p \oplus p')$ and $(t, -p)$ are policies.

We will write p to denote the policy (true, p).

The first stage in policy evaluation for a request q is to determine whether a policy is "applicable" to q or not. Every (well-formed) request q, allows us to assign a truth value to the target t. Specifically, t may evaluate to true, in which case the associated policy is *applicable*; otherwise the policy is *not applicable*.[1] Then, writing $\nu_q(t)$ to denote the truth value assigned to t by q and $\delta_q(p)$ to denote the decision assigned to policy p for request q, we define:

$$\delta_q(t,p) = \begin{cases} \delta_q(p) & \text{if } \nu_q(t) = 1, \\ \perp & \text{otherwise;} \end{cases}$$
$$\delta_q(d) = d;$$
$$\delta_q(-p) = -\delta_q(p);$$
$$\delta_q(p \oplus p') = \delta_q(p) \oplus \delta_q(p').$$

It is easy to see that we may represent a policy as a tree. Hence, we describe policy languages of this nature as *tree-structured*. The first stage in policy evaluation corresponds to labeling the nodes of tree applicable or not applicable. We then compute a decision for non-leaf nodes in the tree by combining the decisions assigned to their respective children. Figure 2 shows the tree for the policy

$$\big(t_6, (t_4, -(t_3, ((t_1, 1) \oplus_1 (t_2, 0)))) \oplus_2 (t_5, 0)\big)$$

and the evaluation of that policy for a request q such that $\nu_q(t_i) = \mathsf{true}$ for all i except $i = 2$.

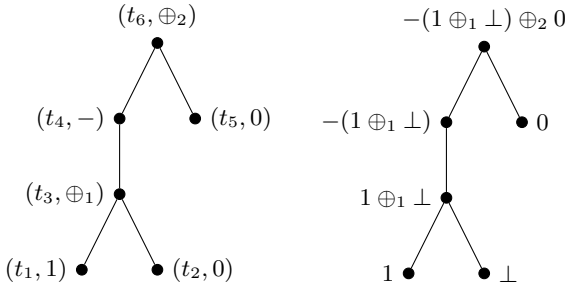

Figure 2: A policy tree and its evaluation

There are several tree-structured ABAC languages in the literature, including the OASIS standard XACML, PBel and PTaCL [17, 6, 4].[2] These languages differ to some extent in the choices of D and the set of operators that are used. XACML, for example, defines several rule- and policy-combining algorithms (which may be regarded as binary operators), but no unary operators.[3] PBel and PTaCL prefer to define a rather small set of operators: PTaCL defines a single binary operator and two unary operators, whereas PBel defines two binary operators and a single unary operator. XACML and PTaCL use a three-valued decision set comprising 0, 1 and $\perp$, to which PBel adds $\top$, which represents "conflict".

The main difference between existing languages, however, is the extent to which they are complete in the senses defined in Section 2.1 [7]. We summarize these differences in Table 1, where CS, FC and CC denote canonically suitable, functionally complete and canonically complete, respectively. In Section 3.3, we prove that PBel is canonically suitable but not canonically complete.

2.3 The Value of Canonical Completeness

One of the main difficulties with using a tree-structured language is writing the desired policy using the operators provided by the language. In particular, if it is not possible to express a policy using a single target and decision, the policy author must engineer the desired policy by combining sub-policies using the set of operators specified in the given language. This is a non-trivial task, in general. Moreover, in XACML it may be impossible to write the desired policy due to its functional incompleteness. Thus, a policy author may be forced to write a policy that approximates the desired policy, which may lead to unintended or undesirable decisions for certain requests.

[1] The evaluation of requests is not relevant to the exposition of this paper. XACML provides a means of specifying requests and targets and an evaluation architecture for determining whether a target is applicable or not.

[2] A number of policy algebras have also been defined, which have some similarities with tree-structured languages. The semantics of a policy are defined in terms of sets of authorized and denied requests [3, 19, 15, 16], and policy operators are defined in terms of set operations such as intersection and union.

[3] An XACML rule is equivalent to an atomic policy.

Language	Decisions	Unary Ops	Binary Ops
XACML	$\{0, 1, \bot\}$	0	12
PTaCL	$\{0, 1, \bot\}$	2	1
PTaCL(E)	$\{0, 1, \bot\}$	2	1
PBel	$\{0, 1, \bot, \top\}$	1	2

Language	CS?	FC?	CC?
XACML	No	No	No
PTaCL	Yes	Yes	No
PTaCL(E)	Yes	Yes	Yes
PBel	?	Yes	?

Table 1: Properties of ABAC languages

An alternative approach, supported by XACML, is to define custom combining algorithms. However, there is no guarantee that the addition of a new combining algorithm will make XACML functionally complete. Thus, more and more custom algorithms may be required over time. This, in turn, will make the design decisions faced by policy authors ever more complicated, thereby increasing the chances of errors and misconfigurations.

In other words, we believe it is preferable to define a small number of operators having unambiguous semantics and providing functional completeness. A functionally complete ABAC language, such as PTaCL, can be used to construct any conceivable policy using the operators provided by the language. However, policy authors still face the challenge of finding the correct way to combine sub-policies using those operators to construct the desired policy.

For example, PTaCL defines three policy operators $\wedge_{\mathrm{p}}, \neg$ and $\sim$. To express XACML's deny- and permit-overrides in PTaCL requires significant effort. For convenience, we introduce the operator $\vee_{\mathrm{p}}$:

$$d \vee_{\mathrm{p}} d' \stackrel{\text{def}}{=} \neg((\neg d) \wedge_{\mathrm{p}} (\neg d')).$$

It is then possible to show that

$$d \,\mathsf{po}\, d' \equiv (d \vee_{\mathrm{p}} (\sim d')) \wedge_{\mathrm{p}} ((\sim d) \vee_{\mathrm{p}} d'), \text{ and}$$
$$d \,\mathsf{do}\, d' \equiv \neg((\neg d) \,\mathsf{po}\, (\neg d')).$$

The operators po and do are equivalent to the permit- and allow-overrides policy-combining algorithms in XACML. As can be seen, the definitions of these operators in terms of the PTaCL operators are complex, and, more generally, it is a non-trivial task to derive such formulae.

Disjunctive normal form in propositional logic makes it trivial to write down a logical formula, using only conjunction, disjunction and negation, that is equivalent to an arbitrary Boolean function expressed in the form of a truth table. Similarly, a canonically complete ABAC language, such as PTaCL(E) [7], makes it possible to write down a policy in normal form from its decision table. In this paper, we show that there exist 4-valued canonically complete logics in which the set of truth values forms a lattice. We discuss why and how this can simplify policy generation in Section 5.2.

In addition, policies in normal form may be more efficient to evaluate. Given a formula in a 3-valued logic expressed in normal form, any literal that evaluates to 0 causes the entire clause to evaluate to 0, while any clause evaluating to 1 means the entire formula evaluates to 1. In short, the time required for policy evaluation may be reduced in many cases. (This is similar to the way in which algorithms such as first-applicable in XACML work: once an applicable policy is found policy evaluation terminates, even if there are additional policies that could be evaluated.)

2.4 The Value of a Fourth Decision

The XACML 2.0 standard includes a fourth authorization decision "indeterminate" [14]. This is used to indicate errors have occurred during policy evaluation, meaning that a decision could not be reached. The XACML 3.0 standard extends the definition of the indeterminate decision to indicate decisions that might have been reached, had evaluation been possible [17]. However, the indeterminate decision is used in XACML 3.0 for more than reporting errors. It is also used as a decision in the "only-one-applicable" combining algorithm, which returns indeterminate if two or more sub-policies are applicable.

More generally, a conflict decision is used in PBel (and languages such as Rumpole and BelLog) to indicate that two sub-policies return different conclusive decisions. PBel is functionally complete. In the remainder of this paper, we show that PBel is not canonically complete and then develop a canonically complete 4-valued ABAC language.

3. LATTICE-BASED MULTI-VALUED LOGICS

We first recall the definition of a lattice. Suppose $(X, \leqslant)$ is a partially ordered set. Then for a subset Y of X, we say u is an *upper bound* of Y if $y \leqslant u$ for all $y \in Y$. We say u' is a *least upper bound* or *supremum* of Y if $u' \leqslant u$ for all upper bounds u of Y. Note that a least upper bound of Y (if it exists) is unique. We define *greatest lower bound* or *infimum* in an analogous way. A lattice $(X, \leqslant)$ is a partially ordered set such that for all $x, y \in X$ there exists a least upper bound of x and y, denoted $\sup\{x, y\}$, and a greatest lower bound of x and y, denoted by $\inf\{x, y\}$. The least upper bound of x and y is written as $x \vee y$ (the "join" of x and y) and the greatest lower bound is written as $x \wedge y$ (the "meet" of x and y). If $(X, \leqslant)$ is a finite lattice, as we will assume henceforth, then $(X, \leqslant)$ has a maximum element (that is, a unique maximal element) and a minimum element.

In the remainder of this section we (i) describe Belnap logic [2], a well-known 4-valued lattice-based logic; (ii) extend the definitions of canonical suitability, selection operators and canonical completeness to lattices; and (iii) show that Belnap logic and PBel are not canonically complete.

3.1 Belnap Logic

Belnap logic was developed with the intention of defining ways to handle inconsistent and incomplete information in a formal manner. It uses the truth values 0, 1, $\bot$, and $\top$, representing "false", "true", "lack of information" and "too much information", respectively. In the remainder of this paper, we will denote the four valued decision set $\{\bot, 0, 1, \top\}$ by 4.

The truth values 0, 1, $\bot$ and $\top$ have an intuitive interpretation in the context of access control: 0 and 1 are interpreted as the standard "deny" and "allow" decisions, $\bot$ is interpreted as "not-applicable" and $\top$ represents a conflict of decisions. PBel is a 4-valued tree-structured ABAC language [4] based on Belnap logic.

The set of truth values in Belnap logic admits two orderings: a truth ordering $\leqslant_t$ and a knowledge ordering $\leqslant_k$. In the truth ordering, 0 is the minimum element and 1 is the maximum element, while $\bot$ and $\top$ are incomparable indeterminate values. In the knowledge ordering, $\bot$ is the minimum element, $\top$ is the maximum element while 0 and 1 are incomparable. Both $(\mathsf{4}, \leqslant_t)$ and $(\mathsf{4}, \leqslant_k)$ are lattices, forming the interlaced bilattice illustrated in Figure 3.

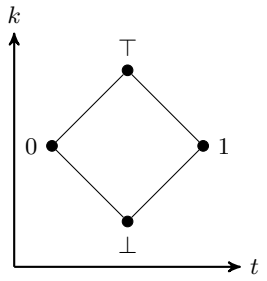

Figure 3: The 4 truth values in Belnap logic

We write the meet and join in $(\mathsf{4}, \leqslant_t)$ as $\wedge_\mathrm{b}$ and $\vee_\mathrm{b}$, respectively; and the meet and join in $(\mathsf{4}, \leqslant_k)$ as $\otimes_\mathrm{b}$ and $\oplus_\mathrm{b}$, respectively. (We use the subscript b to differentiate the Belnap operators from the PTaCL operators $\wedge_\mathrm{p}$ and $\vee_\mathrm{p}$.)

We may interpret values in V as operators of arity 0 (that is, constants). Then it is known that $L(\mathsf{4}, \{\neg, \wedge_\mathrm{b}, \vee_\mathrm{b}, \otimes_\mathrm{b}, \oplus_\mathrm{b}, \supset_\mathrm{b}, \bot, 0, 1, \top\})$ is functionally complete [1, Theorem 12] and that $\{\neg, \oplus_\mathrm{b}, \supset_\mathrm{b}, \bot\}$ is a minimal functionally complete set of operators [1, Proposition 17]. The truth tables for the binary operators $\wedge_\mathrm{b}$, $\vee_\mathrm{b}$, $\otimes_\mathrm{b}$, $\oplus_\mathrm{b}$ and $\supset_\mathrm{b}$ are shown in Figure 11 (in Appendix B). The unary operator $\neg$ has the effect of switching the values 0 and 1, leaving $\bot$ and $\top$ fixed; in other words, it acts like "classical" negation.

3.2 Canonical Completeness

Jobe's definition of canonical suitability for multi-valued logics assumes a total ordering on the set of truth values. Given that Belnap logic [1], on which PBel is based, is a 4-valued logic in which the set of truth values forms a lattice, we seek to extend the definition of canonical suitability to lattice-based logics.

Let L be a logic associated with a lattice $(V, \leqslant)$ of truth values. Then L is *canonically suitable* if and only if there exist in L two formulas $\phi_{\max}$ and $\phi_{\min}$ of arity 2 such that $\phi_{\max}(x, y)$ returns $\sup\{x, y\}$ and $\phi_{\min}(x, y)$ returns $\inf\{x, y\}$. If a logic is canonically suitable, we will write $\phi_{\max}(x, y)$ and $\phi_{\min}(x, y)$ using infix binary operators as $x \curlyvee y$ and $x \curlywedge y$, respectively.

REMARK 1. *The existence of* $\sup\{x, y\}$ *and* $\inf\{x, y\}$ *is guaranteed in a lattice; this is not true in general for partially ordered sets. And for a totally ordered (finite) set,* $\sup\{x, y\} = \max\{x, y\}$ *and* $\inf\{x, y\} = \min\{x, y\}$*, so our definitions are compatible with those of Jobe's for totally ordered sets of truth values.*

We now extend the definition of selection operators to a lattice-based logic. Let L be a logic associated with a lattice $(V, \leqslant)$ of truth values, with minimum truth value $\underline{v}$. Then, for $\mathbf{a} \in V^n$, the n-ary *selection operator* $S^j_\mathbf{a}$ is defined as follows:

$$S^j_\mathbf{a}(\mathbf{x}) = \begin{cases} j & \text{if } \mathbf{x} = \mathbf{a}, \\ \underline{v} & \text{otherwise.} \end{cases}$$

Note $S^{\underline{v}}_\mathbf{a}(x) = \underline{v}$ for all $\mathbf{a}, \mathbf{x} \in V^n$.

The definitions for normal form and canonically completeness for lattice-based logics are identical to total-ordered logics. Nevertheless, we reiterate the definitions here in the interests of clarity. The *normal form* of formula ϕ in a canonically suitable logic is a formula ϕ' that has the same truth table as ϕ and has the following properties:

- the only binary operators it contains are $\curlyvee$ and $\curlywedge$;
- no binary operator is included in the scope of a unary operator;
- no instance of $\curlyvee$ occurs in the scope of the $\curlywedge$ operator.

A canonically suitable logic is *canonically complete* if every unary selection operator can be expressed in normal form.

3.3 Completeness of Belnap Logic and PBel

Having extended the definitions of canonical suitability, selection operators and canonical completeness to lattices, we now investigate how these concepts can be applied to Belnap logic [2]. The meet and join operators of the two lattices $(\mathsf{4}, \leqslant_t)$ and $(\mathsf{4}, \leqslant_k)$ defined in Belnap logic are different. Canonical suitability for a 4-valued logic, defined as it is in terms of the ordering on the set of truth values, will thus depend on the ordering we choose on $\mathsf{4}$. Consequently, the $\curlywedge$ and $\curlyvee$ operators, along with the selection operators, will differ depending on the lattice that we choose.

In Section 5, we will argue in more detail for the use of a lattice-based ordering on $\mathsf{4}$ to support a tree-structured ABAC language. For now, we state that we will use knowledge-ordered lattice $(\mathsf{4}, \leqslant_k)$.[4] The intuition is that the minimum value in this lattice is $\bot$ (rather than 0 in $(\mathsf{4}, \leqslant_t)$) and that this value should be the default value for a policy (being returned when the policy is not applicable to a request). In the interests of brevity, we will henceforth write $\mathsf{4}_k$, rather than $(\mathsf{4}, \leqslant_k)$.

It follows from the functional completeness of $\{\neg, \oplus_\mathrm{b}, \supset_\mathrm{b}, \bot\}$ that $L(\mathsf{4}_k, \{\neg, \oplus_\mathrm{b}, \supset_\mathrm{b}, \bot\})$ is canonically suitable. A similar argument applies to $\{\neg, \wedge_\mathrm{b}, \supset_\mathrm{b}, \bot, \top\}$, the set of operators used in PBel.

As $\bot$ is the minimum truth value in the lattice $\mathsf{4}_k$, the n-ary selection operator $S^j_\mathbf{a}$ for $\mathsf{4}_k$ is defined by the following function:

$$S^j_\mathbf{a}(\mathbf{x}) = \begin{cases} j & \text{if } \mathbf{x} = \mathbf{a}, \\ \bot & \text{otherwise.} \end{cases}$$

Examples of selection operators are shown in Figure 4.

Functional completeness also implies all unary selection operators can be expressed as formulas in the logics $L(\mathsf{4}_k, \{\neg, \oplus_\mathrm{b}, \supset_\mathrm{b}, \bot\})$ and $L(\mathsf{4}_k, \{\neg, \wedge_\mathrm{b}, \supset_\mathrm{b}, \bot, \top\})$. However, we have the following result, from which it follows that neither of these logics is canonically complete.

PROPOSITION 1. $L(\mathsf{4}_k, \{\neg, \wedge_\mathrm{b}, \vee_\mathrm{b}, \otimes_\mathrm{b}, \oplus_\mathrm{b}, \supset_\mathrm{b}, \bot, 0, 1, \top\})$ *is not canonically complete.*

[4] It is worth noting that results analogous to those presented in this paper can be obtained for the truth ordering. Results for a total ordering on $\mathsf{4}$ can be derived using existing methods [7, 9].

x	S_0^0	$S_1^\top$
$\bot$	$\bot$	$\bot$
0	0	$\bot$
1	$\bot$	$\top$
$\top$	$\bot$	$\bot$

$S^0_{(1,\top)}$	$\bot$	0	1	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
0	$\bot$	$\bot$	$\bot$	$\bot$
1	$\bot$	$\bot$	$\bot$	0
$\top$	$\bot$	$\bot$	$\bot$	$\bot$

Figure 4: Examples of selection operators in a logic based on $\mathsf{4}_k$

PROOF. It is impossible to represent all unary selection operators in normal form. The statement follows from the following observations: (i) Belnap logic defines one unary operator $\neg$; (ii) the only binary operators that may be used in normal form are $\oplus_\mathrm{b}$ ($\curlywedge$) and $\otimes_\mathrm{b}$ ($\curlyvee$); and (iii) for any operator $\oplus \in \{\neg, \otimes_\mathrm{b}, \oplus_\mathrm{b}\}$ we have $\bot \oplus \bot = \bot$. Thus it is impossible to construct a unary operator of the form $S^d_\bot$ for any $d \neq \bot$. □

COROLLARY 1. *PBel is not a canonically complete authorization language.*

PROOF. PBel uses the set of operators $\{\neg, \wedge_\mathrm{b}, \supset_\mathrm{b}, \bot, \top\}$, which is a subset of $\{\neg, \wedge_\mathrm{b}, \vee_\mathrm{b}, \otimes_\mathrm{b}, \oplus_\mathrm{b}, \supset_\mathrm{b}, \bot, 0, 1, \top\}$. Thus, by Proposition 1, this is not a canonically complete set of operators, so $L(\mathsf{4}_k, \{\neg, \wedge_\mathrm{b}, \supset_\mathrm{b}, \bot, \top\})$ is not canonically complete either. Hence, we may conclude that PBel is not a canonically complete authorization language. □

4. A CANONICALLY COMPLETE 4-VALUED LOGIC

In the proof of Proposition 1, we were unable to construct all unary selection operators using operators from the set $\{\neg, \otimes_\mathrm{b}, \oplus_\mathrm{b}\}$, because there is no operator in which $\bot \oplus \bot \neq \bot$. This suggests that we will require at least one additional unary operator $-$, say, such that $-\bot \neq \bot$. Accordingly, we start with the unary operator, sometimes called "conflation" [8], such that

$$-\bot = \top,\ -\top = \bot,\ -0 = 0,\ \text{and}\ -1 = 1.$$

Conflation is analogous to negation $\neg$, but inverts knowledge values rather than truth values. In addition to $-$, we include the operator $\otimes_\mathrm{b}$ in our set of operators, since this is the join operator for $\mathsf{4}_k$.

PROPOSITION 2. *$L(\mathsf{4}_k, \{-, \otimes_\mathrm{b}\})$ is canonically suitable.*

Informally, the proof follows from the fact that $-$ and $\otimes_\mathrm{b}$ have exactly the same effect on $\mathsf{4}_k$ as $\neg$ and $\wedge_\mathrm{b}$ have on $(\mathsf{4}, \leqslant_t)$. More formally, the following equivalence holds [1]:

$$d \oplus_\mathrm{b} d' \equiv -(-d \otimes_\mathrm{b} -d').$$

The decision table establishing this equivalence is given in Figure 12 (in Appendix C). Hence, we conclude that the set of operators $\{-, \otimes_\mathrm{b}\}$ is canonically suitable, since $\curlywedge$ corresponds to $\otimes_\mathrm{b}$ and $\curlyvee$ corresponds to $\oplus_\mathrm{b}$.

PROPOSITION 3. *$L(\mathsf{4}_k, \{-, \otimes_\mathrm{b}\})$ is not functionally complete.*

PROOF. The proof follows from the following observations: (i) for the operators $-$ and $\otimes_\mathrm{b}$, $-(0) = 0$ and $0 \otimes_\mathrm{b} 0 = 0$; and (ii) any operator $\circ$ which is a combination of $-$ and $\otimes_\mathrm{b}$, we have $0 \circ 0 = 0$. Thus it is impossible to construct an operator in which $0 \circ 0 \neq 0$. □

To summarize: $L(\mathsf{4}_k, \{\neg, \wedge_\mathrm{b}, \vee_\mathrm{b}, \otimes_\mathrm{b}, \oplus_\mathrm{b}, \supset_\mathrm{b}, \bot, 0, 1, \top\})$ is not canonically complete and $L(\mathsf{4}_k, \{-, \otimes_\mathrm{b}\})$ is not functionally complete. We now investigate what additional operators should be defined to construct a set of operators which is canonically complete (and hence functionally complete).

Given that we cannot use any operators besides $\curlyvee$ and $\curlywedge$ in normal form, we focus on defining additional unary operators on $\mathsf{4}_k$. An important observation at this point is that any permutation (that is, a bijection) $\pi : \mathsf{4} \to \mathsf{4}$ defines a unary operator on $\mathsf{4}$. Accordingly, we now explore the connections between the group of permutations on $\mathsf{4}$ and unary operators on $\mathsf{4}$.

4.1 The Symmetric Group and Unary Operators

The *symmetric group* $(S_X, \circ)$ on a finite set of $|X|$ symbols is the group whose elements are all permutations of the elements in X, and whose group operation $\circ$ is function composition. In other words, given two permutations π_1 and π_2, $\pi_1 \circ \pi_2$ is a permutation such that

$$(\pi_1 \circ \pi_2)(x) \stackrel{\text{def}}{=} \pi_1(\pi_2(x)).$$

We write π^k to denote the permutation obtained by composing π with itself k times.

A *transposition* is a permutation which exchanges two elements and keeps all others fixed. Given two elements a and b in X, the permutation

$$\pi(x) = \begin{cases} b & \text{if } x = a, \\ a & \text{if } x = b, \\ x & \text{otherwise,} \end{cases}$$

is a transposition, which we denote by $(a\,b)$. A *cycle* of *length* $k \geqslant 2$ is a permutation π for which there exists an element x in X such that $x, \pi(x), \pi^2(x), \ldots, \pi^k(x) = x$ are the only elements changed by π. Given a, b and c in X, for example, the permutation

$$\pi(x) = \begin{cases} b & \text{if } x = a, \\ c & \text{if } x = b, \\ a & \text{if } x = c, \\ x & \text{otherwise,} \end{cases}$$

is a cycle of length 3, which we denote by $(a\,b\,c)$. (Cycles of length two are transpositions.) The symmetric group S_X is *generated* by its cycles. That is, every permutation may be represented as the composition of some combination of cycles.

In fact, stronger results are known. We first introduce some notation. Let $X = \{x_1, \ldots, x_n\}$ and let S_n denote the symmetric group on the set of elements $\{1, \ldots, n\}$. Then $(S_X, \circ)$ is trivially isomorphic to $(S_n, \circ)$ (via the mapping $x_i \mapsto i$).

THEOREM 2. *For $n \geqslant 2$, S_n is generated by the transpositions $(1\,2), (1\,3), \ldots, (1\,n)$.*

THEOREM 3. *For $1 \leqslant a < b \leqslant n$, the transposition $(a\,b)$ and the cycle $(1\,2\,\ldots\,n)$ generate S_n if and only if the greatest common divisor of $b - a$ and n equals 1.*

In other words, it is possible to find a generating set comprising only transpositions, and it is possible to find a generating set containing only two elements.

4.2 New Unary Operators

We now define three unary operators $\sim_0, \sim_1$ and $\sim_\top$, which swap the value of $\bot$ and the truth value in the operator's subscript. The truth tables for these operators are shown in Figure 5. Note that $\sim_\top$ is identical to the conflation operator $-$. However, in the interests of continuity and consistency we will use the $\sim_\top$ notation in the remainder of this section.

d	$\sim_0 d$	$\sim_1 d$	$\sim_\top d$
$\bot$	0	1	$\top$
0	$\bot$	0	0
1	1	$\bot$	1
$\top$	$\top$	$\top$	$\bot$

Figure 5: $\sim_0, \sim_1$ and $\sim_\top$

Notice that $\sim_0$, $\sim_1$ and $\sim_\top$ permute the elements of $\mathsf{4}$ and correspond to the transpositions $(\bot\, 0)$, $(\bot\, 1)$ and $(\bot\, \top)$, respectively. Thus we have the following elementary result.

PROPOSITION 4. *Any permutation on* $\mathsf{4}$ *can be expressed using only operators from the set* $\{\sim_0, \sim_1, \sim_\top\}$.

PROOF. The operators $\sim_0, \sim_1$ and $\sim_\top$ are the transpositions $(\bot\, 0), (\bot\, 1)$ and $(\bot\, \top)$ respectively. By Theorem 2, these operators generate all the permutations in S_4. □

LEMMA 1. *It is possible to express any function* $\phi : \mathsf{4} \to \mathsf{4}$ *as a formula in* $L(\mathsf{4}_k, \{\sim_0, \sim_1, \sim_\top, \otimes_\mathrm{b}, \oplus_\mathrm{b}\})$.

PROOF. For convenience, we represent the function $\phi : \mathsf{4} \to \mathsf{4}$ as the tuple

$$(\phi(\bot), \phi(0), \phi(1), \phi(\top)) = (a, b, c, d).$$

Then, given $x, y, z \in \mathsf{4}$, we define the function

$$\phi_x^y(z) = \begin{cases} x & \text{if } z = y, \\ \bot & \text{otherwise.} \end{cases}$$

Thus, for example, $\phi_a^\bot = (a, \bot, \bot, \bot)$. Then it is easy to see that for all $x \in \mathsf{4}$

$$\phi(x) = \phi_a^\bot(x) \oplus_\mathrm{b} \phi_b^0(x) \oplus_\mathrm{b} \phi_c^1(x) \oplus_\mathrm{b} \phi_d^\top(x).$$

That is $\phi = \phi_a^\bot \oplus_\mathrm{b} \phi_b^0 \oplus_\mathrm{b} \phi_c^1 \oplus_\mathrm{b} \phi_d^\top$.

Thus, it remains to show that we can represent $\phi_a^\bot$, ϕ_b^0, ϕ_c^1 and $\phi_d^\top$ as formulas using the operators in $\{\sim_0, \sim_1, \sim_\top, \otimes_\mathrm{b}, \oplus_\mathrm{b}\}$. First consider the permutations $\phi_{a,0}$, $\phi_{a,1}$ and $\phi_{a,\top}$, represented by the tuples $(a, \bot, b_1, c_1)$, $(a, b_2, \bot, c_2)$ and $(a, b_3, c_3, \bot)$, respectively.[5] Since $\phi_{a,0}$, $\phi_{a,1}$ and $\phi_{a,\top}$ are permutations, we know they can be written as some combination of the unary operators. Moreover,

$$\phi_a^\bot \equiv \phi_{a,0} \otimes_\mathrm{b} \phi_{a,1} \otimes_\mathrm{b} \phi_{a,\top}$$

Clearly, we can construct ϕ_b^0, ϕ_c^1 and $\phi_d^\top$ in a similar fashion. The result now follows. □

The decision tables showing the construction of $\phi_a^\bot$ (column 5) and ϕ (column 10) are shown in Figure 6.

[5]Note that the specific values of b_i and c_i are not important: it suffices that each of $\phi_{a,0}$, $\phi_{a,1}$ and $\phi_{a,\top}$ are permutations; once b_i is chosen such that $b_i \notin \{a, \bot\}$, then c_i is fixed.

THEOREM 4. $L(\mathsf{4}_k, \{\sim_0, \sim_1, \sim_\top, \otimes_\mathrm{b}, \oplus_\mathrm{b}\})$ *is functionally and canonically complete.*

PROOF. By Lemma 1, it is possible to express any function $\phi : \mathsf{4} \to \mathsf{4}$ as a formula using operators from the set $\{\sim_0, \sim_1, \sim_\top, \otimes_\mathrm{b}, \oplus_\mathrm{b}\}$. In particular, all unary selection operators can be expressed in this way. Hence by Theorem 1, the set of operators $\{\sim_0, \sim_1, \sim_\top, \otimes_\mathrm{b}, \oplus_\mathrm{b}\}$ is functionally complete.

Moreover, all formulae constructed in the proof of Lemma 1 contain only the binary operators $\oplus_\mathrm{b}(\curlyvee)$ and $\otimes_\mathrm{b}(\curlywedge)$, and unary operators defined as compositions of $\sim_0, \sim_1$ and $\sim_\top$. Thus, by definition, the unary selection operators are in normal form. □

COROLLARY 2. $L(\mathsf{4}_k, \{\sim_0, \sim_1, \sim_\top, \otimes_\mathrm{b}\})$ *is functionally and canonically complete.*

PROOF. The conflation operator $-$ and $\sim_\top$ are identical. Hence

$$d \oplus_\mathrm{b} d' \equiv -(-\, d \otimes_\mathrm{b} -\, d') \equiv \sim_\top(\sim_\top d \otimes_\mathrm{b} \sim_\top d').$$

Therefore, the set of operators is canonically suitable, and, by Theorem 4, it is functionally and canonically complete (since we can construct $\oplus_\mathrm{b}$). □

COROLLARY 3. *Let* $\diamond$ *be the unary operator corresponding to the permutation given by the cycle* $(\bot\, 0\, 1\, \top)$. *Then* $L(\mathsf{4}_k, \{\sim_\top, \diamond, \otimes_\mathrm{b}\})$ *is functionally and canonically complete.*

PROOF. By Theorem 3, $\sim_\top$ and $\diamond$ generate all permutations in S_4. The remainder of the proof follows immediately from Lemma 1 and Theorem 4. □

It is important to note that we could choose any transposition $(a\, b)$, such that $\gcd(b - a, n) = 1$. We specifically selected the transposition $(\bot\, \top)$, as this has the effect of reversing the minimum and maximum knowledge values. Another choice for this transposition is one which swaps 0 and 1, specifically the transposition $(0\, 1)$. This transposition is the truth negation operator $\neg$, which in the context of access control is a useful operator, since it swaps allow and deny decisions.

4.3 Unary Operators for Totally Ordered Logics

Having shown the construction for a canonically complete 4-valued logic, in which the set of logical values forms a lattice, we briefly return to totally ordered logics. We construct a totally ordered, canonically complete m-valued logic (thus extending the work of Jobe, who only showed how to construct a canonically complete 3-valued logic).

Let V be a totally ordered set of m truth values, $\{1, \ldots, m\}$, with $1 < \cdots < m$. We define two unary operators $\dagger$ and $\diamond$, which are the transposition $(1\, m)$ and the cycle $(1\, 2\, \ldots\, m)$, respectively. In addition, we define one binary operator $\wedge_t$, where $x \wedge_t y = \max\{x, y\}$.

PROPOSITION 5. *Any permutation on* V *can be expressed using only operators from the set* $\{\dagger, \diamond\}$.

PROOF. The operator $\dagger$ is the transposition $(1\, m)$ and the operator $\diamond$ is the cycle $(1\, 2\, \ldots\, m)$. By Theorem 3, these operators generate all the permutations in S_V. □

PROPOSITION 6. $L(V, \{\dagger, \diamond, \wedge_t\})$ *is canonically suitable.*

x	$\phi_{a,0}$	$\phi_{a,1}$	$\phi_{a,\top}$	$\phi_{a,0} \otimes_\mathrm{b} \phi_{a,1} \otimes_\mathrm{b} \phi_{a,\top}$	$\phi_a^\bot$	ϕ_b^0	ϕ_c^1	$\phi_d^\top$	$\phi_a^\bot \oplus_\mathrm{b} \phi_b^0 \oplus_\mathrm{b} \phi_c^1 \oplus_\mathrm{b} \phi_d^\bot$
$\bot$	a	a	a	a	a	$\bot$	$\bot$	$\bot$	a
0	$\bot$	b_2	b_3	$\bot$	$\bot$	b	$\bot$	$\bot$	b
1	b_1	$\bot$	c_3	$\bot$	$\bot$	$\bot$	c	$\bot$	c
$\top$	c_1	c_2	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	d	d

Figure 6: Expressing $\phi : \mathsf{4} \to \mathsf{4}$ using operators in $\{\sim_0, \sim_1, \sim_\top, \otimes_\mathrm{b}, \oplus_\mathrm{b}\}$

PROOF. Clearly $x \curlywedge y \equiv x \wedge_t y$, it remains to show the operator $\curlyvee$ can be expressed in L. By Proposition 5 we can express any permutation of V in terms of $\dagger$ and $\diamond$. In particular, we can express the permutation f, where $f(i) = m - i + 1$, which swaps the values 1 and m, 2 and $m-1$, and so on. We denote the unary operator which realizes this permutation by $\updownarrow$. Then $x \curlyvee y \equiv x \vee_t y \equiv \updownarrow(\updownarrow x \wedge_t \updownarrow y)$. □

THEOREM 5. *$L(V, \{\dagger, \diamond, \wedge_t\}$ is functionally and canonically complete.*

We omit the proof, as it proceeds in an analogous manner to those for Lemma 1 and Theorem 4. It is interesting to note that we have constructed a canonically complete m-valued logic which uses only two unary operators. This is somewhat unexpected; intuition would suggest that $m-1$ unary operators are required for a canonically complete m-valued logic.

5. A CANONICALLY COMPLETE 4-VALUED ABAC LANGUAGE

Having identified a canonically complete set of operators for Belnap logic, we now investigate how this set of operators can be used in an ABAC language, and consider the advantages in doing so. Crampton and Williams [7] showed the operators in PTaCL can be replaced with an alternative set of operators, taken from Jobe's logic E, to obtain a canonically complete 3-valued ABAC language. In the remainder of this section, we describe a 4-valued lattice-ordered version of PTaCL, based on the lattice $\mathsf{4}_k$, which we denote by $\mathrm{PTaCL}_4^{\leqslant}$.

5.1 The Decision Set

We first reiterate there is value in having an ABAC language for which policy evaluation can return a fourth value $\top$. Such a value is used in both XACML and PBel, although its use in XACML is somewhat ad hoc and confusing since it can be used to indicate (a) an error in policy evaluation, or (b) a decision that arises for a particular operator during normal policy evaluation.

We will use this fourth value to denote that (normal) policy evaluation has led to conflicting decisions (and we do not wish to use deny-overrides or similar operators to resolve the conflict at this point in the evaluation). (We explain how we handle indeterminacy arising from errors in policy evaluation in Section 5.3.) Two specific operators, "only-one-applicable" (ooa) and "unanimity" (un) could make use of $\top$: the ooa operator returns the value of the applicable sub-policy if there is only one such policy, and $\top$ otherwise; whereas the un operator returns $\top$ if the sub-policies return different decisions, and the common decision otherwise. The decision tables for these operators are shown in Figure 7.

ooa	$\bot$	0	1	$\top$
$\bot$	$\bot$	0	1	$\top$
0	0	$\top$	$\top$	$\top$
1	1	$\top$	$\top$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

un	$\bot$	0	1	$\top$
$\bot$	$\bot$	$\top$	$\top$	$\top$
0	$\top$	0	$\top$	$\top$
1	$\top$	$\top$	1	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

Figure 7: Operators using $\top$

In establishing canonical completeness for PTaCL(E), Crampton and Williams assumed a total order on the set of decisions ($0 < \bot < 1$). This ordering does not really reflect the intuition behind the use of 0, 1 and $\bot$ in ABAC languages. In the context of access control, 0 and 1 are incomparable conclusive decisions, and $\bot$ and $\top$ are decisions that reflect the inability to reach a conclusive decision either because a policy or its sub-policies are inapplicable ($\bot$) or because a policy's sub-policies return conclusive decisions that are incompatible in some sense ($\top$). Moreover, we can subsequently resolve $\bot$ and $\top$ into one of two (incomparable) conclusive decisions using unary operators such as "deny-by-default" and "allow-by-default". (The truth-based ordering on $\mathsf{4}$ does not correspond nearly so well to the above intuitions.)

5.2 Operators and Policies

We define the set of operators for $\mathrm{PTaCL}_4^{\leqslant}$ to be $\{\sim_\top, \diamond, \otimes_\mathrm{b}\}$, which we established is canonically complete in Corollary 3. Recall that $\sim_\top$ is equivalent to conflation $-$; we will use the simpler notation $-$ in the remainder of this section. An atomic policy has the form (t, d), where t is a target and $d \in \{0, 1\}$. (There is no reason for an atomic policy to return $\top$ – which signifies a conflict has taken place – in an atomic policy.) Then we have the following policy semantics.

$$\delta_q(t,p) = \begin{cases} \delta_q(p) & \text{if } \nu_q(t) = 1, \\ \bot & \text{otherwise;} \end{cases}$$
$$\delta_q(d) = d;$$
$$\delta_q(-p) = -\delta_q(p); \quad \delta_q(\diamond p) = \diamond\, \delta_q(p);$$
$$\delta_q(p \otimes_\mathrm{b} p') = \delta_q(p) \otimes_\mathrm{b} \delta_q(p').$$

We now show how to represent the operator only-one-applicable (ooa) in normal form. (Recall that it is possible to represent this operator as a formula in PBel; however, it is non-trivial to derive such a formula.) Using the truth table in Figure 7 and by definition of the selection operators

and $\curlyvee$, we have x ooa y is equivalent to

$$S^{\bot}_{(\bot,\bot)}(x,y) \curlyvee S^{0}_{(\bot,0)}(x,y) \curlyvee S^{1}_{(\bot,1)}(x,y) \curlyvee S^{\top}_{(\bot,\top)}(x,y) \curlyvee$$
$$S^{0}_{(0,\bot)}(x,y) \curlyvee S^{\top}_{(0,0)}(x,y) \curlyvee S^{\top}_{(0,1)}(x,y) \curlyvee S^{\top}_{(0,\top)}(x,y) \curlyvee$$
$$S^{1}_{(1,\bot)}(x,y) \curlyvee S^{\top}_{(1,0)}(x,y) \curlyvee S^{\top}_{(1,1)}(x,y) \curlyvee S^{\top}_{(1,\top)}(x,y) \curlyvee$$
$$S^{\top}_{(\top,\bot)}(x,y) \curlyvee S^{\top}_{(\top,0)}(x,y) \curlyvee S^{\top}_{(\top,1)}(x,y) \curlyvee S^{\top}_{(\top,\top)}(x,y).$$

Moreover, $S^{z}_{(x,y)} = S^{z}_{x} \curlywedge S^{z}_{y}$ and S^{y}_{x} is a function $\phi : \mathsf{4} \to \mathsf{4}$, which can be represented as a composition of unary operators. Hence, we can derive a formula in normal form for ooa.

Functional completeness implies we can write any binary operator (such as XACML's deny-overrides policy-combining algorithm) as a formula in $L(\mathsf{4}_k, \{-, \diamond, \otimes_{\mathrm{b}}\}$, and hence we can use any operator we wish in PTaCL$^{\leqslant}_{4}$ policies. However, canonical completeness and the decision set $(\mathsf{4}, \leqslant_k)$ allows for a completely different approach to constructing ABAC policies. Suppose a policy administrator has identified three sub-policies p_1, p_2 and p_3 and wishes to define an overall policy p in terms of the decisions obtained by evaluating these sub-policies. Then the policy administrator can tabulate the desired decision for all relevant combinations of decisions for the sub-policies, as shown in the table below. The default decision is to return $\bot$, indicating that p is "silent" for other combinations.

p_1	p_2	p_3	p
$\bot$	0	0	0
0	0	0	0
1	0	0	$\top$
1	1	0	1
1	1	1	1

Then, treating p as a function of its sub-policies, we have

$$p \equiv\ S^{0}_{(\bot,0,0)} \curlyvee S^{0}_{(0,0,0)} \curlyvee S^{\top}_{(1,0,0)} \curlyvee S^{1}_{(1,1,0)} \curlyvee S^{1}_{(1,1,1)}.$$

The construction of p as a "disjunction" ($\curlyvee$) of selection operators ensures that the correct value is returned for each combination of values (in much the same as disjunctive normal form may be used to represent the rows in a truth table). Note that if p_1, p_2 and p_3 evaluate to a tuple of values other than one of the rows in the table, each of the selection operators will return $\bot$ and thus p will evaluate to $\bot$. Each operator of the form $S^{d}_{(a,b,c)}$ can be represented as the "conjunction" ($\curlywedge$) of unary selection operators (specifically $S^{d}_{a} \curlywedge S^{d}_{b} \curlywedge S^{d}_{c}$).

Of course, one would not usually construct the normal form by hand, as we have done above. Indeed, we have developed an algorithm which takes an arbitrary policy expressed as a decision table as input, and outputs the equivalent normal form expressed in terms of the operators $\{-, \diamond, \otimes_{\mathrm{b}}\}$. In order to develop this algorithm, we also derived expressions for the unary selection operators in terms of the operators $\{-, \diamond, \otimes_{\mathrm{b}}\}$. (In Lemma 1 we only showed that such expressions exist.) Our implementation of the algorithm, comprising just less than 150 lines of Python code, shows the ease with which construction of policies can be both automated and simplified, utilizing the numerous advantages that have been discussed throughout this paper.[6]

5.3 Indeterminacy

[6]Code and test results available at goo.gl/0TM0RD.

XACML uses the indeterminate value in two distinct ways:

1. as a decision returned (during normal evaluation) by the "only-one-applicable" policy-combining algorithm; and

2. as a decision returned when some (unexpected) error has occurred in policy evaluation has occurred.

In the second case, the indeterminate value is used to represent alternative outcomes of policy evaluation (had the error not occurred). We believe that the two situations described are quite distinct and require different policy semantics. However, the semantics of indeterminacy in XACML are confused because (i) the indeterminate value is used in two different ways, as described above, and (ii) there is no clear and uniform way of establishing the values returned by the combining algorithms when an indeterminate value is encountered.

We have seen how $\top$ may be used to represent decisions for operators such as ooa and un. We handle errors in target evaluation (and thus indeterminacy) using sets of possible decisions [5, 6, 10]. (This approach was adopted in a rather ad hoc fashion in XACML 3.0, using an extended version of the indeterminate decision.) Informally, when target evaluation fails, denoted by $\nu_q(t) = ?$, PTaCL assumes that either $\nu_q(t) = 1$ or $\nu_q(t) = 0$ could have been returned, and returns the union of the (sets of) decisions that would have been returned in both cases. The formal semantics for policy evaluation in PTaCL$^{\leqslant}_{4}$ in the presence of indeterminacy are defined in Figure 8.

$$\delta_q(t,p) = \begin{cases} \delta_q(p) & \text{if } \nu_q(t) = 1, \\ \{\bot\} & \text{if } \nu_q(t) = 0, \\ \{\bot\} \cup \delta_q(p) & \text{if } \nu_q(t) = ?, \end{cases}$$
$$\delta_q(d) = \{d\}\,;$$
$$\delta_q(-p) = \{-d : d \in \delta_q(p)\}\,;$$
$$\delta_q(\diamond\, p) = \{\diamond\, d : d \in \delta_q(p)\}\,;$$
$$\delta_q(p_1 \otimes_{\mathrm{b}} p_2) = \{d_1 \otimes_{\mathrm{b}} d_2 : d_i \in \delta_q(p_i)\}\,.$$

Figure 8: Semantics for PTaCL$^{\leqslant}_{4}$ with indeterminacy

The semantics for the operators $\{-, \diamond, \otimes_{\mathrm{b}}\}$ operate on sets, rather than single decisions, in the natural way. A straightforward induction on the number of operators in a policy establishes that the decision set returned by these extended semantics will be a singleton if no target evaluation errors occur; moreover, that decision will be the same as that returned by the standard semantics.

5.4 Leveraging the XACML Architecture

XACML is a well-known, standardized language, and many of the components and features of XACML are well-defined. However, it has been shown that the rule- and policy-combining algorithms defined in the XACML standard suffer from some shortcomings [10], notably inconsistencies between the rule- and policy-combining algorithms. PTaCL, on which PTaCL$^{\leqslant}_{4}$ is based, is a tree-structured ABAC language that is explicitly designed to use the same general policy structure and evaluation methods as XACML.

However, PTaCL differs substantially from XACML in terms of policy combination operators and semantics.

Thus, we suggest that PTaCL$_4^{\leqslant}$ operators could replace the rule- and policy-combining algorithms of XACML, while those parts of the language and architecture that seem to function well may be retained. Specifically, we use the XACML architecture to: (i) specify requests; (ii) specify targets; (iii) decide whether a policy target is applicable to a given request; and (iv) use the policy decision point to evaluate policies. In addition, we would retain the enforcement architecture of XACML, in terms of the policy decision, policy enforcement and policy administration points, and the relationships between them.

We believe it would be relatively easy to modify the XACML PDP to

- handle four decisions, extending the current set of values ("allow", "deny" and "not-applicable") to include "conflict";
- implement the policy operators $\{-, \diamond, \otimes_{\mathrm{b}}\}$ as custom combining algorithms; and
- work with decision sets, in order to handle indeterminacy in a uniform manner.

For illustrative purposes, Appendix D specifies the modified decision set and pseudocode for the operator $\otimes_{\mathrm{b}}$ in the format used by the XACML standard.

The main difference to end-users would be in the simplicity of policy authoring. Using standard XACML, policy authors must decide which rule- and policy-combining algorithms should be used to develop a policy or policy set that is equivalent to the desired policy. This is error-prone and it may not even be possible to express the desired policy using only the XACML combining algorithms. Using XACML with the policy-combining mechanisms of PTaCL$_4^{\leqslant}$, we can present an entirely different interface for policy authoring to the end-user. The policy author would first specify the atomic policies (XACML rules), then combine atomic policies using decision tables to obtain more complex policies (as illustrated in Section 5.2). Those policies can be further combined by specifying additional decision tables. At each stage a back-end policy compiler can be used to convert those policies into policy sets (using PTaCL$_4^{\leqslant}$ operators) that can be evaluated by the XACML engine.

6. CONCLUDING REMARKS

Attribute-based access control is of increasing importance, due to the increasing use of open, distributed, interconnected and dynamic systems. The introduction of *canonically complete* ABAC languages [7] provides the ability to express any desired policy in a normal form, which allows for the possibility of specifying policies in the form of a decision table and then automatically compiling them into the language.

In this paper, we make important contributions to the understanding of canonical completeness in multi-valued logics and thus in ABAC languages. First, we extend Jobe's work on canonical completeness to multi-valued logics to the case where the set of truth values forms a lattice. We show that the Belnap set of operators [2] (and thus any subset thereof) is not canonically complete, hence any ABAC language based on these operators cannot be canonically complete. In particular, PBel [4], probably the most well known 4-valued ABAC language, is not canonically complete. We introduce a new four-valued logic $L(\mathbf{4}_k, \{-, \diamond, \otimes_{\mathrm{b}}\})$ which is canonically complete, without having to explicitly construct the unary selection operators in normal form (unlike Jobe [9] and Crampton and Williams [7]). By identifying the connection between the generators of the symmetric group and the unary operators of logics, we have developed a simple and generic method for identifying a set of unary operators that will guarantee the functional and canonical completeness of an m-valued lattice-based logic. We also showed that there is a set of operators containing only three connectives which is functionally complete for Belnap logic, in contrast to the set of size four identified by Arieli and Avron [1].

Second, we show in PTaCL$_4^{\leqslant}$ how the canonically complete set of operators $\{-, \diamond, \otimes_{\mathrm{b}}\}$ can be used in an ABAC language, and present the advantages of doing so. In particular, we are no longer forced to use a totally ordered set of three decisions to obtain canonical completeness (as in the case in PTaCL(E)). Moreover, the overall design of PTaCL and hence PTaCL$_4^{\leqslant}$ is compatible with the overall structure of XACML policies. We discuss how the XACML decision set and rule-combining algorithms can be modified to support PTaCL$_4^{\leqslant}$. Doing so enables us to retain the rich framework provided by XACML for ABAC (in terms of its languages for representing targets and requests) and its enforcement architecture (in terms of the policy enforcement, policy decision and policy administration points). Thus, we are able to propose an enhanced XACML framework within which any desired policy may be expressed. Moreover, the canonical completeness of PTaCL$_4^{\leqslant}$, means that the desired policy may be represented in simple terms by a policy author (in the form of a decision table) and automatically compiled into a PDP-readable equivalent policy.

Our work paves the way for a considerable amount of future work. In particular, we intend to develop a modified XACML PDP that implements the PTaCL$_4^{\leqslant}$ operators. We also hope to develop a policy authoring interface in which users can simply state what decision a policy should return for particular combinations of decisions from sub-policies. This would enable us to evaluate the usability of such an interface and compare the accuracy with which policy authors can generate policies using standard XACML combining algorithms compared with the methods that PTaCL$_4^{\leqslant}$ can support.

On the more technical side, we would like to revisit the notion of *monotonicity* [6] in targets and how this affects policy evaluation in ABAC languages. The definition of monotonicity is dependent on the ordering chosen for the decision set and existing work on monotonicity assumes the use of a totally ordered 3-valued set (comprising 0, $\bot$ and 1). So it will be interesting to consider how the use of a 4-valued lattice-ordered decision set affects monotonicity. We also intend to investigate methods of *policy compression*, analogous to the minimization of Boolean functions [13], where we take the canonical form of a policy (generated from a decision table) and rewrite it in such a way as to minimize the number of terms in the policy.

Acknowledgements

Conrad Williams is a student in the Centre for Doctoral Training in Cyber Security, supported by EPSRC award EP/K035584/1.

7. REFERENCES

[1] Arieli, O., and Avron, A. The value of the four values. *Artif. Intell. 102*, 1 (1998), 97–141.

[2] Belnap Jr, N. D. A useful four-valued logic. In *Modern uses of multiple-valued logic*. Springer, 1977, pp. 5–37.

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

[4] Bruns, G., and Huth, M. Access control via Belnap logic: Intuitive, expressive, and analyzable policy composition. *ACM Trans. Inf. Syst. Secur. 14*, 1 (2011), 9.

[5] Crampton, J., and Huth, M. An authorization framework resilient to policy evaluation failures. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings* (2010), D. Gritzalis, B. Preneel, and M. Theoharidou, Eds., vol. 6345 of *Lecture Notes in Computer Science*, Springer, pp. 472–487.

[6] Crampton, J., and Morisset, C. PTaCL: A language for attribute-based access control in open systems. In *Principles of Security and Trust - First International Conference, POST 2012, Proceedings*, P. Degano and J. D. Guttman, Eds., vol. 7215 of *Lecture Notes in Computer Science*. Springer, 2012, pp. 390–409.

[7] Crampton, J., and Williams, C. On completeness in languages for attribute-based access control. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies, SACMAT 2016, Shanghai, China, June 5-8, 2016* (2016), X. S. Wang, L. Bauer, and F. Kerschbaum, Eds., ACM, pp. 149–160.

[8] Fitting, M. Bilattices and the semantics of logic programming. *J. Log. Program. 11*, 1&2 (1991), 91–116.

[9] Jobe, W. H. Functional completeness and canonical forms in many-valued logics. *The Journal of Symbolic Logic 27*, 04 (1962), 409–422.

[10] Li, N., Wang, Q., Qardaji, W. H., Bertino, E., Rao, P., Lobo, J., and Lin, D. Access control policy combining: theory meets practice. In *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Proceedings* (2009), pp. 135–144.

[11] Łukasiewicz, J. Philosophische Bemerkungen zu mehrwertigen Systemen des Aussagekalküls. *Comtes rendus des séances de la Société des Sciences et des Lettres de Varsovie Classe III*, vol. 23 (1930), 55–57.

[12] Marinovic, S., Dulay, N., and Sloman, M. Rumpole: An introspective break-glass access control language. *ACM Trans. Inf. Syst. Secur. 17*, 1 (2014), 2:1–2:32.

[13] McCluskey, E. J. Minimization of boolean functions. *Bell System Technical Journal 35*, 6 (1956), 1417–1444.

[14] Moses, T. eXtensible Access Control Markup Language (XACML) Version 2.0 OASIS Standard, 2005. http://docs.oasis-open.org/xacml/2.0/access-control-xacml-2.0-core-spec-os.pdf.

[15] Ni, Q., Bertino, E., and Lobo, J. D-algebra for composing access control policy decisions. In *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009, Sydney, Australia, March 10-12, 2009* (2009), W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadharajan, Eds., ACM, pp. 298–309.

[16] Rao, P., Lin, D., Bertino, E., Li, N., and Lobo, J. An algebra for fine-grained integration of XACML policies. In *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, June 3-5, 2009, Proceedings* (2009), pp. 63–72.

[17] Rissanen, E. eXtensible Access Control Markup Language (XACML) Version 3.0 OASIS Standard, 2012. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-os-en.html.

[18] Tsankov, P., Marinovic, S., Dashti, M. T., and Basin, D. A. Decentralized composite access control. In *POST* (2014), vol. 8414 of *Lecture Notes in Computer Science*, Springer, pp. 245–264.

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

APPENDIX

A. JOBE'S CANONICALLY COMPLETE 3-VALUED LOGIC

Consider the 3-valued logic J [9], whose operators $\wedge_{\mathrm{e}}, \sim_1$ and $\sim_2$ are defined in Figure 9.

x	$\sim_1 x$	$\sim_2 x$
0	1	2
1	0	1
2	2	0

$\wedge_{\mathrm{e}}$	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

Figure 9: The operators in Jobe's logic

It is easy to establish that

$$x \curlywedge y \equiv x \wedge_{\mathrm{e}} y \quad \text{and} \quad x \curlyvee y \equiv \sim_2(\sim_2(x) \wedge_{\mathrm{e}} \sim_2(y)).$$

Thus J is canonically suitable [9, Theorem 6]. The normal-form formulas for the unary selection operators are shown in Figure 10. (Note that S_i^0 is the same for all i.) Thus J is functionally and canonically complete [9, Theorem 7]. Hence, it is possible to construct a canonically complete 3-valued logic using the operators $\{\wedge_{\mathrm{e}}, \sim_0, \sim_1\}$.

$S_i^0(x)$	$x \wedge_{\mathrm{e}} \sim_1(x) \wedge_{\mathrm{e}} \sim_2(x)$
$S_0^1(x)$	$\sim_1(x) \wedge_{\mathrm{e}} \sim_2 \sim_1(x)$
$S_1^1(x)$	$x \wedge_{\mathrm{e}} \sim_2(x)$
$S_2^1(x)$	$\sim_1 \sim_2(x) \wedge_{\mathrm{e}} \sim_2 \sim_1 \sim_2(x)$
$S_0^2(x)$	$\sim_2(x) \wedge_{\mathrm{e}} \sim_1 \sim_2(x)$
$S_1^2(x)$	$\sim_2 \sim_1(x) \wedge_{\mathrm{e}} \sim_2 \sim_1 \sim_2(x)$
$S_2^2(x)$	$x \wedge_{\mathrm{e}} \sim_1(x)$

Figure 10: Normal forms for the unary selection operators

B. OPERATORS IN BELNAP LOGIC

$\wedge_b$	0	$\bot$	$\top$	1
0	0	0	0	0
$\bot$	0	$\bot$	0	$\bot$
$\top$	0	0	$\top$	$\top$
1	0	$\bot$	$\top$	1

(a) $\wedge_b$

$\vee_b$	0	$\bot$	$\top$	1
0	0	$\bot$	$\top$	1
$\bot$	$\bot$	$\bot$	1	1
$\top$	$\top$	1	$\top$	1
1	1	1	1	1

(b) $\vee_b$

$\otimes_b$	$\bot$	0	1	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
0	$\bot$	0	$\bot$	0
1	$\bot$	$\bot$	1	1
$\top$	$\bot$	0	1	$\top$

(c) $\otimes_b$

$\oplus_b$	$\bot$	0	1	$\top$
$\bot$	$\bot$	0	1	$\top$
0	0	0	$\top$	$\top$
1	1	$\top$	1	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

(d) $\oplus_b$

$\supset_b$	0	$\bot$	$\top$	1
0	1	1	1	1
$\bot$	1	1	1	1
$\top$	0	$\bot$	$\top$	1
1	0	$\bot$	$\top$	1

(e) $\supset_b$

d	$\neg\, d$
0	1
$\bot$	$\bot$
$\top$	$\top$
1	0

(f) $\neg$

Figure 11: Operators in Belnap logic

C. PROOF OF PROPOSITION 2

The decision table in Figure 12 establishes the equivalence of $x \oplus_b y$ and $-(x \otimes_b -y)$, which proves that $L((\mathsf{4}, \leqslant_k), \{-, \otimes_b\})$ is a canonically suitable logic (Proposition 2).

d	d'	$-\,d$	$-\,d'$	$-\,d \otimes_b -\,d'$	$-(-\,d \otimes_b -\,d')$	$d \oplus_b d'$
$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$
$\bot$	0	$\top$	0	0	0	0
$\bot$	1	$\top$	1	1	1	1
$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\top$	$\top$
0	$\bot$	0	$\top$	0	0	0
0	0	0	0	0	0	0
0	1	0	1	$\bot$	$\top$	$\top$
0	$\top$	0	$\bot$	$\bot$	$\top$	$\top$
1	$\bot$	1	$\top$	1	1	1
1	0	1	0	$\bot$	$\top$	$\top$
1	1	1	1	1	1	1
1	$\top$	1	$\bot$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\top$
$\top$	0	$\bot$	0	$\bot$	$\top$	$\top$
$\top$	1	$\bot$	1	$\bot$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$

Figure 12: Encoding $\oplus_b$ using $-$ and $\otimes_b$

D. ENCODING PTACL DECISIONS AND OPERATORS

In Figures 13 and 14 we illustrate how $\text{PTaCL}_4^{\leqslant}$ extensions could be incorporated in XACML by encoding the $\text{PTaCL}_4^{\leqslant}$ decisions and $\otimes_b$ operator using the syntax of the XACML standard.

```
<xs:element name=''Decision''
            type=''xacml:DecisionType''/>
<xs:simpleType name=''DecisionType''>
 <xs:restriction base=''xs:string''>
   <xs:enumeration value=''Permit''/>
   <xs:enumeration value=''Deny''/>
   <xs:enumeration value=''Conflict''/>
   <xs:enumeration value=''NotApplicable''/>
 </xs:restriction>
</xs:simpleType>
```

Figure 13: The $\text{PTaCL}_4^{\leqslant}$ decision set in XACML syntax

```
Decision ptaclCombiningAlgorithm(Node[] children)
{
 Boolean atLeastOneDeny  = false;
 Boolean atLeastOnePermit  = false;
 for( i=0 ; i < lengthOf(children) ; i++ )
 {
   Decision decision = children[i].evaluate();
   if (decision == NotApplicable)
   {  return NotApplicable;  }
   if (decision == Permit)
   {
     atLeastOnePermit = true;
     continue;
   }
   if (decision == Deny)
   {
     atLeastOneDeny = true;
     continue;
   }
   if (decision == Conflict)
   {  continue;  }
 }
 if (atLeastOneDeny && atLeastOnePermit)
 {  return NotApplicable;  }
 if (atLeastOneDeny)
 {  return Deny;  }
 if (atLeastOnePermit)
 {  return Permit;  }
 return Conflict;
}
```

Figure 14: The $\text{PTaCL}_4^{\leqslant}$ operator $\otimes_b$ encoded as an XACML combining algorithm