Towards A Generic Formal Framework for Access Control Systems

Jason Crampton\(^1\) and Charles Morisset\(^2\)

\(^1\) Royal Holloway, University of London
Jason.Crampton@rhul.ac.uk
\(^2\) Newcastle University
Charles.Morisset@ncl.ac.uk

Abstract. There have been many proposals for access control models and authorization policy languages, which are used to inform the design of access control systems. Most, if not all, of these proposals impose restrictions on the implementation of access control systems, thereby limiting the type of authorization requests that can be processed or the structure of the authorization policies that can be specified. In this paper, we develop a formal characterization of the features of an access control model that imposes few restrictions of this nature. Our characterization is intended to be a generic framework for access control, from which we may derive access control models and reason about the properties of those models. In this paper, we consider the properties of monotonicity and completeness, the first being particularly important for attribute-based access control systems. XACML, an XML-based language and architecture for attribute-based access control, is neither monotonic nor complete. Using our framework, we define attribute-based access control models, in the style of XACML, that are, respectively, monotonic and complete.

1 Introduction

One of the fundamental security services in modern computer systems is access control, a mechanism for constraining the interaction between (authenticated) users and protected resources. Generally, access control is enforced by a trusted component (historically known as the reference monitor), which typically implements two functions: an authorization enforcement function (AEF) and an authorization decision function (ADF). The AEF traps all attempts by a user to interact with a resource (usually known as a user request) and transforms that request into one or more authorization queries (also known as authorization requests) which are forwarded to the ADF.

Most access control systems are policy-based. That is, an administrator specifies an authorization policy, which, in its simplest form, encodes those authorization requests that are authorized. The ADF takes an authorization query and an authorization policy as input and returns an authorization decision. For this reason, it is common to refer to the AEF and ADF as the policy enforcement
point (PEP) and policy decision point (PDP), respectively; it is this terminology that we will use henceforth.

An authorization policy is merely an encoding of the access control requirements of an application using the authorization language that is understood by the PDP. It is necessary, therefore, to make a distinction between an ideal policy and a realizable policy: the former is an arbitrary function from requests to decisions; the latter is a function that can be evaluated by the PDP. Given a particular policy language, there might be some ideal policies that are not realizable, which may be a limitation of the policy language in practice. The access control system used in early versions of Unix, for example, is rather limited [1, §15.1.1]. An important consideration, therefore, when designing an access control system is the expressivity of the policy language.

The increasing prevalence of open, distributed, computing environments means that we may not be able to rely on a centralized authentication function to identify authorized users. This means that authorization decisions have to be made on the basis of (authenticated) user attributes (rather than user identities). In turn, this means that the structure of authorization queries needs to be rather more flexible than that used in closed, centralized environments. The draft XACML 3.0 standard, for example, uses a much “looser” query format than its predecessor XACML 2.0. However, if we have no control over the attributes that are presented to the PDP, then a malicious user (or a user who wishes to preserve the secrecy of some attributes) may be able to generate authorization decisions that are more “favorable” by withholding attributes from the PEP [23]. A second important consideration, therefore, is whether authorization policies are guaranteed to be “monotonic” in the sense that providing fewer attributes in an authorization query yields a less favorable outcome (from the requester’s perspective).

There is an extensive literature on languages for specifying authorization policies, most approaches proposing a new language or an extension of an existing one. The proliferation of languages led Ferraiolo and Atluri to raise the question in [4] of whether a meta-model for access control was needed and possible to achieve, hinting at XACML [5] and RBAC [6] as potential candidates. In response, Barker proposed a meta-model [7], which sought to identify the key components required to specify access control policies, based on a term-rewriting evaluation.

In this paper, we do not present “yet another language” for access control policies, nor do we claim to have a “unifying meta-model”. We focus instead on reasoning about the properties of a language. Indeed, we advocate the idea that a language is just a tool for policy designers: just as some programming languages are better suited to particular applications, it seems unlikely that there exists a single access control model (or meta-model) that is ideal in all possible contexts. On the contrary, we believe that providing the structure to formally analyse a language might be valuable to a policy designer, in order to understand the suitability of a particular language as the basis for a specific access control system.
We conclude this section by summarizing the structure and contributions of the paper. In Sec. 2 we propose a general framework for access control, whose role is not to be used as an off-the-shelf language, but as a way to identify and reason about the key aspects of a language. In Sec. 3 we define monotonicity and completeness in the context of our framework. Then in Sec. 4 we define two attribute-based models, respectively monotonic and complete, by building on existing results from the literature on multi-valued and partial logic. The main body of the paper ends with discussions of related and future work.

2 A Framework for Defining Access Control Models

In this section we describe the various components of our framework and introduce our formal definition of access control models and policies. Broadly speaking, we provide a generic method for designing access control models and for furnishing access control policies, which are written in the context of a model, with authorization semantics. We also introduce the notion of an ideal policy, which is an abstraction of the requirements of an organization, and relate this concept to that of an access control policy.

2.1 An Informal Overview

From an external viewpoint, an access control mechanism is a process that constrains the interactions between users and data objects. Those interactions are modeled as access requests, with the mechanism incorporating two functions: one to determine whether a request is authorized or not and one to enforce that decision. The overall process must be total, in the sense that its behavior is defined for every possible interaction (which may include some default behavior that is triggered when the decision function is unable to return a decision). In general, designing a particular access control mechanism for a particular set of requests is the final concrete objective of any access control framework (although we are also clearly interested in expressing general properties of the framework).

We define an access control mechanism using an access control policy, together with an interpretation function which provides the authorization semantics for a policy. Intuitively, a policy is simply a syntactical object, built from atomic policies and policy connectives. The interpretation function provides the denotational semantics of the policy, by returning a function from requests to decision, thus defining the expected behavior of the PDP. Clearly, a policy can be interpreted in different ways, and an interpretation function can interpret different policies, as long as they are built from the same atomic policies and connectives.

An access control model defines an access control language, which consists of a set of atomic policies and policy connectives, and an interpretation function. In other words, an access control model specifies a set of access control policies and a unique way to interpret each of these policies. An access control mechanism, then, is an instance of an access control model if its policy belongs to the language of the model and if its interpretation function is that of the model.
2.2 The Framework

In order to provide a framework within which policies can be constructed, we introduce the notion of access control model, which is a tuple $\mathcal{M} = (Q, A, \text{Ops}, \text{Dec}, [\cdot])$, where $Q$ is a set of requests, $A$ a set of atomic authorization policies, Ops a set of policy connectives, Dec a set of (authorization) decisions, and, for each $A \in A$, $[A]$ is a total function from $Q$ to Dec defining the evaluation of policy $A$ for all requests in $Q$.

Each $k$-ary policy connective $\text{op}$ in Ops is identified with a function $\text{op} : \text{Dec}^k \rightarrow \text{Dec}$. We construct an authorization policy $P$ using elements of $A$ and Ops. We extend the evaluation function for atomic policies to arbitrary policies: that is, $[P] : Q \rightarrow \text{Dec}$ provides a method of evaluating requests with respect to a policy $P$. We say that $[\cdot]$ defines the authorization semantics of the model.

The syntax by which policies are defined and the extension of the authorization semantics for atomic policies to non-atomic policies are fixed (for all models), as specified in Definition 1 below. Nevertheless, different choices for Dec, A and $[\cdot]$ give rise to very different models having very different properties.

A policy term $P$ is defined by a (rooted) policy tree, in which leaf nodes are atomic policies and each non-leaf node is a policy connective (we may also use the term policy operator). More formally we have the following definition:

\textbf{Definition 1} Let $\mathcal{M} = (Q, A, \text{Ops}, \text{Dec}, [\cdot])$ be a model. Then every atomic policy in $A$ is a policy term. If $P_1, \ldots, P_k$ are policy terms, then for each $k$-ary operator $\text{op} \in \text{Ops}$, $\text{op}(P_1, \ldots, P_k)$ is a policy term. For each policy term $\text{op}(P_1, \ldots, P_k)$, we define

\[ [\text{op}(P_1, \ldots, P_k)](q) = \text{op}([P_1](q), \ldots, [P_k](q)). \]  

(1)

In other words, authorization policies are represented as policy trees and policies are evaluated from the bottom up by (a) evaluating atomic policies (b) combining the decisions returned for atomic policies using the relevant policy connectives. We write $\mathcal{P}(\mathcal{M})$ to denote the set of policies that can be expressed within $\mathcal{M}$.

Given a set of queries $Q$ and a set of decisions Dec, an ideal access control policy is a total function $\pi : Q \rightarrow \text{Dec}$ such that for any query $q$, $\pi(q) = [P](q)$; in the interests of simplicity we will abuse notation and write $\pi \in \mathcal{P}(\mathcal{M})$ and $\pi = (\mathcal{M}, P)$.

Figure 1 shows two policy trees each having the same atomic policies, $A_1$ and $A_2$. The figure also shows two evaluations of the tree for the same request $q$.

---

3 Strictly speaking, we should use different symbols for a policy connective and the decision operator with which it is associated. We have chosen to abuse notation in the interests of clarity, because little is gained by strict adherence to formality here.

4 Clearly, a policy designer could define a policy extensionally, simply by associating each query with a decision. However, in practice, policies are constructed in a modular fashion, where each component defines a particular security concern and the decisions from different components are combined.
where \([A_1](q) = 1\) and \([A_2](q) = \bot\). The symbols 1, 0 and \(\bot\) denote allow, deny and inapplicable decisions, respectively. The policy trees are evaluated using a post-order traversal, in which each leaf node is assigned a value according to the semantics defined by \([\cdot]\) and each interior node is assigned a value by combining the values assigned to its child nodes. The policies in Figure 1 make use of three operators taken from Table 1. Both \(\triangle\) and \(\land\) are similar to the allow-overrides operator familiar from XACML (and also the two conjunction operators from Kleene’s 3-valued logic) and only differ in the way in which \(\bot\) is combined with 1. The \(\sim\) unary operator implements a deny-by-default rule, thus \([P_1](q) \neq [P_2](q)\).

![Policy Trees and Evaluation](image)

Fig. 1. Illustrative policy trees and their evaluation

In general, an access control model does not specify any policy in particular (unless the language is so restricted that it can only specify one policy). To some extent, an access control model (in the sense in which we use the term in this paper) is analogous to a programming language: it describes the syntax that is used to build access control policies (analogous to programs) and the semantics of the run-time mechanisms that will be used to handle input data (access control requests in this context). A realizable policy is in this case analogous to a program \(P\) written in the syntax of the model \(M\), that is interpreted using the authorization semantics of the model, while an ideal policy is analogous to the set of functional requirements.

Note that an ideal policy can be realized by different access control models: \(\pi \in \mathcal{P}(M)\) and \(\pi \in \mathcal{P}(M')\) with \(M \neq M'\). In other words, different access control mechanisms may be able to enforce the same security requirements. And \(\pi\) may be realizable by different policy terms from the same access control model: \(\pi = (M, P)\) and \(\pi = (M, P')\) with \(P \neq P'\). In other words, security requirements can be enforced by the same mechanism using different policies. However, an ideal policy may not be realizable by any policy term for a given model; the extent to which a model can realize the set of ideal policies provides us with a notion of the completeness of a model (as we discuss in Section 3.2).

2.3 Framework Instantiation

A model provides the global structure from which access control policies can be built. A simple example of a model is the protection matrix model [8], which can
be viewed as a set of triples \((s, o, x)\), where \(s\) is a subject, \(o\) an object and \(x\) an access mode. A query is also a triple \((s, o, x)\), and is authorized if, and only if, it belongs to the set representing the matrix. Hence, we define the set of queries \(Q_{AM}\) to be the set of all triples \((s, o, x)\), the set of decisions \(\text{Dec}_{AM} = \{1, 0\}\), where 1 stands for an authorized access and 0 for a denied one, the set of atomic policies \(A_{AM} = Q_{AM} \cup \{0\}\), the set of operators \(\text{Ops}_{AM} = \{\lor\}\), where \(\lor\) is the standard boolean disjunction, and the interpretation function \([\cdot]_{AM}\) to be:

\[
[p]_{AM}(q) = \begin{cases} 
1 & \text{ if } p = q \\
0 & \text{ otherwise.}
\end{cases}
\]

For instance, the policy authorizing only the accesses \((s_1, o_1, x_1)\) and \((s_2, o_2, x_2)\) can be defined as \((s_1, o_1, x_1) \lor (s_2, o_2, x_2)\).

Models can also consider richer sets of queries. Indeed, recent work considers the possibility that, in order to make a decision, an access control system might require more attributes than the traditional subject-object-action triple [2,5,9].

In order to define requests and atomic policies it is necessary to identify sets of attributes and the values that each of those attributes may take. Role-based access control, to take a simple example, defines the sets of roles, users and permissions, together with user-role and permission-role assignment relations.

We now introduce the notions of attribute vocabulary and attribute-based access control, which are intended to be as general as possible and allow for the construction of requests and policies.

**Definition 2** Let \(\mathcal{N}\) denote a set of attribute names, and \(\mathcal{D}\) denote a set of attribute domains. Let \(\text{dom}: \mathcal{N} \to \mathcal{D}\) be a function, where \(\text{dom}(\alpha)\) denotes the set of attribute values associated with attribute \(\alpha\). Then \((\mathcal{N}, \mathcal{D}, \text{dom})\) defines an attribute vocabulary.

When no confusion can occur, we will simply write \(\mathcal{N}\) to denote an attribute vocabulary. A request is modeled as a set of name-value pairs of the form \((\alpha, v)\), where \(\alpha \in \mathcal{N}\). We denote the set of requests by \(Q^*(\mathcal{N})\), omitting \(\mathcal{N}\) when it is obvious from context. We say an attribute name-value pair \((\alpha, v)\) is well-formed if \(\alpha \in \mathcal{N}\) and \(v \in \text{dom}(\alpha)\). We assume that a PDP can recognize (and discard) name-value pairs in a request that are not well-formed.

Attribute-based access control (ABAC) policies are modular. Hence, a policy component may be incomplete or two policy components may return contradictory decisions. Thus, it is common to see additional decisions used to denote a policy “gap” or “doubt” indicating different reasons why policy evaluation could not reach a conclusive (allow or deny) decision [10]. We write \(\text{Three} = \{1, 0, \bot\}\), where \([A](q) = \bot\) indicates that \([A](q)\) is neither 0 nor 1.

In Table 1 we summarize the characteristics of some useful 3-valued operators, most of which are self-explanatory. The ? operator acts as a policy filter: \([p_1 ? p_2] = [p_2]\) if \([p_1] = 1\), and evaluates to \(\bot\) otherwise. The \(\&\) operator models policy unanimity: \(p_1 \& p_2\) evaluates to a conclusive decision only if both \(p_1\) and \(p_2\) do. In Sec. 4.3 we describe a model with a 4-valued decision set.
ABAC is designed for open distributed systems, meaning that authenticated attributes and policy components may need to be retrieved from multiple locations. Thus, some languages assume that policy evaluation may fail: it may be, for example, that a policy server or policy information point is down. PTaCL [2] relies on a three-valued logic, and considers sets of decisions in order to model indeterminacy. XACML 3.0 [5] considers a six-valued decision set, three of those decisions representing different indeterminate answers.

### 3 Monotonicity and Completeness

An access control model provides a policy designer with a language to construct a policy. That language may well have an impact on the policies that can be expressed and the properties of those policies. In this section we study two specific properties of access control models, monotonicity (a kind of safety property) and completeness (an expressivity property), and we present two models satisfying these properties in Section 4.

#### 3.1 Monotonicity

Informally, a policy is monotonic whenever removing information from a request does not lead to a “better” policy decision. Such a property is of particular relevance in open systems, where users might be able to control what information they supply to the access control mechanism. A model in which all realizable policies are monotonic implies that they are not vulnerable to attribute hiding attacks [2]. That is, a malicious user gains no advantage by suppressing information when making a request.

We model information hiding using a partial ordering \( \preceq_Q \) on \( Q \); the intuitive interpretation of \( q \preceq_Q q' \) is that \( q \) contains less information than \( q' \). For instance, an attribute query \( q \) is less than another query \( q' \) when \( q \subseteq q' \). We also need to
specify what it means for a decision to “benefit” a user, and thus we assume the existence of an ordering relation ⩽ on Dec; again, the intuitive interpretation of $d_1 \leq d_2$ is that the decision $d_2$ is of greater benefit than $d_1$. For instance, we can consider the ordering $\leq_3$ over \{1, 0, ⊥\}, such that $x \leq_3 y$ if and only if $x = y$ or $x = ⊥$.

**Definition 3** Given a set of authorization queries $(Q, \leq_Q)$ and a set of decisions $(\text{Dec}, \leq)$, a policy $\phi : Q \to \text{Dec}$ is monotonic if, and only if, for all $q, q' \in Q$, $q \leq_Q q'$ implies $\phi(q) \leq \phi(q')$. We say that an access control model $\mathcal{M} = (Q, A, \text{Dec}, \text{Ops}, [\cdot])$ is monotonic if for all $P \in \mathcal{P}(\mathcal{M})$, $[P]$ is monotonic.

Note that our definition of a monotonic policy applies equally well to an ideal policy $\pi : Q \to \text{Dec}$ or a realizable policy term $P$ with authorization semantics $[P] : Q \to \text{Dec}$. However, the notion of monotonicity is dependent on the request ordering. For instance, without further characterization, the request ordering for the access matrix could be reduced to equality, making any policy trivially monotonic. However, more complex situations can be considered by adding extra information, such as an ordering over subjects or objects.

Tschantz and Krisnamurthi have shown that XACML 2.0 is not monotonic (although they called the property “safety” rather than monotonicity) [11]. We show in Section 4.1—provided certain restrictions are imposed on the structure of requests—that it is possible to develop a monotonic, attributed-based (XACML-like) access control model, using results from partial logic [12].

### 3.2 Completeness

Given a model $\mathcal{M} = (Q, A, \text{Dec}, \text{Ops}, [\cdot])$, any realizable policy $P \in \mathcal{P}(\mathcal{M})$ clearly corresponds to an ideal policy $\pi : Q \to \text{Dec}$. However, there may exist an ideal policy $\pi$ (for $Q$ and Dec) that does not belong to $\mathcal{P}(\mathcal{M})$ and cannot, therefore, be enforced by the policy decision point. Trivially, for example, a model without any atomic policies does not realize any policies. It follows that the set of ideal policies that can be realized by a model represents an intuitive notion of expressivity. A model that can realize every ideal policy is said to be complete. More formally:

**Definition 4** An access control model $\mathcal{M} = (Q, A, \text{Dec}, \text{Ops}, [\cdot])$ is complete if, and only if for any ideal policy $\pi : Q \to \text{Dec}$, $\pi \in \mathcal{P}(\mathcal{M})$.

The completeness of a model $(Q, A, \text{Dec}, \text{Ops}, [\cdot])$ will depend on the authorization vocabulary, the definition of atomic policies, the set $\text{Ops}$ and $[\cdot]$. The access matrix model defined in Section 2.3, for example, is complete.

**Proposition 5** The model $(Q_{AM}, A_{AM}, \text{Dec}_{AM}, \text{Ops}_{AM}, [\cdot]_{AM})$ is complete.

---

Note that we consider this relation to be statically defined over decisions, and to be independent of the request.
On the other hand, it is easy to show that XACML is not complete, unless we allow the inclusion of XACML conditions, which are arbitrary functions. Indeed, consider two attributes \( \alpha_1 \) and \( \alpha_2 \) with two respective attribute values \( v_1 \) and \( v_2 \), it is not possible to construct a policy that evaluates \( q_1 = \{(\alpha_1, v_1)\} \) to \text{Permit} and \( q_2 = \{(\alpha_1, v_1), (\alpha_2, v_2)\} \) to \text{NA}, intuitively because any target not applicable to \( q_2 \) cannot be applicable to \( q_1 \).

We propose an attribute-based access control model in Section 4.2 in which the representation of atomic policies can distinguish attribute name-value pairs, from which we can prove a completeness result. However, it is worth observing that, in general, if a model is both monotonic and complete, then the ordering over requests is limited to the identity relation, as illustrated above with the access matrix.

**Proposition 6** Given any model \( \mathcal{M} = ((Q, \leq_Q), \mathcal{A}, (\text{Dec}, \leq), \text{Ops}, [\cdot]) \), if \( \mathcal{M} \) is complete and monotonic and if \( |\text{Dec}| > 1 \), then \( \leq_Q \) is the identity relation.

Informally, this result states that if we wish to have a (non-trivial) monotonic model then we cannot expect to have a complete model. Instead, what we should aim for is a model that realizes at least all monotonic ideal policies, and such a model is said to be \text{monotonically-complete}. In Section 4, we show how to define monotonically-complete and complete attribute-based access control models that have similar characteristics to XACML.

### 4 Designing Attribute-based Access Control Models

It could be argued that the main objective of XACML is to provide a standard addressing as many practical concerns as possible, rather than a language with formal semantics. Nevertheless, the design choices can and should be analyzed with respect to the properties they entail. We do not claim here that XACML should be monotonic, complete, or monotonically-complete, but we show instead how, building from existing logical results, one can instantiate an access control model with these properties.

The results in this section can provide guidance to the designer of an access control system. She can choose, for example, between a system that realizes only and all monotonic policies, and a system in which all policies are realizable, but some may be non-monotonic. Clearly, the choice depends on the demands of the application and the constraints of the underlying environment. While we cannot make this choice for the policy designer, our framework can only help her make an informed decision.

If the attribute vocabulary were countably infinite (and the cardinality of the decision set is greater than 1) then the number of ideal policies would be uncountably infinite (by a standard diagonalization argument). However, the number of realizable policies can, at best, be countably infinite, by construction. Accordingly, it is only meaningful to consider completeness if we assume that the attribute vocabulary is finite (but unbounded). In practice, of course, all attribute values will be stored as variables and there will be an upper limit on
the size of such variables, so the attribute vocabulary will be finite and bounded, albeit by a very large number.

4.1 \( \text{ABAC}_M \): A monotonic monotonically-complete model

Recall from Definition 2 that, given a vocabulary \( \mathcal{N} \), we write \( \mathcal{Q}^*(\mathcal{N}) \) to denote the set of requests. Note that a request may contain (well-formed) pairs \((\alpha, v_1), \ldots, (\alpha, v_n)\) having the same attribute name and different values. One obvious example arises when \( \alpha \) is the “role” attribute name and \( v_i \) is the identifier of a role. We define the set of atomic policies \( \mathcal{A}(\mathcal{N}) \) to be the set of well-formed name-value pairs. That is \( \mathcal{A}(\mathcal{N}) = \{ (\alpha, v) : \alpha \in \mathcal{N} \text{ and } v \in \text{dom}(\alpha) \} \). Then we define

\[
\llbracket (\alpha, v) \rrbracket (q) = \begin{cases} 
1 & \text{if } q \ni (\alpha, v') \text{ and } v = v', \\
\bot & \text{if } q \notni (\alpha, v'), \\
0 & \text{otherwise.}
\end{cases}
\]

Note that the above interpretation of atomic policies is by no means the only possibility. In the context of a three-value decision set, we might return 0 if \( q \ni (\alpha, v') \) and \( v \not= v' \), \( \bot \) if \( q \notni (\alpha, v') \) and 1 otherwise. In the context of a four-value decision set, we could return \( \top \) if \( q \supseteq \{(\alpha, v'),(\alpha, v)\} \), since such a request both matches and does not match the attribute value \( v \) for attribute \( \alpha \).

We discuss these possibilities in more detail in Sec. 4.3.

The ordering on \( \mathcal{Q}^* \), denoted by \( \leq_{\mathcal{Q}} \), is simply subset inclusion. We define the ordering \( \leq_3 \) on \( \text{Three} \), where \( x \leq_3 y \) if and only if \( x = y \) or \( x = \bot \). It is worth observing that if a request contains at most one value for each attribute, then each atomic policy is monotonic. More formally, if we define the set of queries \( \mathcal{Q}^? = \{ q \subseteq \mathcal{A}(\mathcal{N}) \mid \forall \alpha \ (\alpha, v) \in q \land (\alpha, v') \in q \Rightarrow v = v' \} \), we can prove the following proposition.

**Proposition 7** For all requests \( q, q' \in \mathcal{Q}^? \) such that \( q \leq_{\mathcal{Q}} q' \) and for all atomic policies \( (\alpha, v) \in \mathcal{A}(\mathcal{N}) \), we have \( \llbracket (\alpha, v) \rrbracket (q) \leq_3 \llbracket (\alpha, v) \rrbracket (q') \).

We will see in the following section that we can define a complete ABAC model that accepts requests from \( \mathcal{Q}^* \), but we can no longer ensure monotonicity.

We now define a monotonic and monotonically-complete attribute-based access control (ABAC) model.

**Definition 8** \( \text{ABAC}_M \) is defined to be \((\mathcal{Q}^?, \mathcal{A}(\mathcal{N}), \text{Three}, \{\neg, \land, \lor, \wedge, \vee, ?\}, \llbracket \rrbracket )\).

\( \text{ABAC}_M \) is not merely of academic interest because it incorporates a number of features that are similar to XACML. In particular, we can

- construct targets from conjunctions and disjunctions of atomic policies;
- use the operators \( \land \) and \( \lor \) to model deny-overrides and allow-overrides policy-combining algorithms;
construct (XACML) rules, policies and policy sets using policies of the form

\( p_1 \land p_2 \), since \( \llbracket p_1 \land p_2 \rrbracket = \llbracket p_2 \rrbracket \) if \( \llbracket p_1 \rrbracket = 1 \) (corresponding to “matching” a request to “target” \( p_1 \) and then evaluating policy \( p_2 \)).

The correspondence between \( \text{ABAC}_M \) and XACML cannot be exact, given that XACML is not monotonic. The main difference lies in the way in which \( \land \) and \( \lor \) handle the \( \bot \) decision. The operators \( \land \) and \( \lor \) are what Crampton and Huth called intersection operators [13], whereas the policy-combining algorithms in XACML are union operators. Informally, an intersection operator requires both operands to have conclusive decisions, while a union operator ignores inconclusive decisions. Thus, for example, \( 1 \land \bot = \bot \), whereas the XACML deny-overrides algorithm would return 1 given the same arguments.

A practical consequence of the design goals of \( \text{ABAC}_M \) is that the \( \bot \) decision will be returned more often than for analogous policies in XACML (or other non-monotonic languages). In practice, the policy enforcement point will have to either (a) ask the requester to supply additional attributes in the request; or (b) deny all requests that are not explicitly allowed.

**Theorem 9** \( \text{ABAC}_M \) is monotonic and monotonically complete.

**Proof.** Let us first observe that the operators \( \neg, \land, \lor, \& \) and \( ? \), as defined in Table 1, are monotonic with respect to \( \leq_3 \). Following Proposition 7, we know that atomic policies are monotonic, and by direct induction, we can conclude that any policy in \( P(\text{ABAC}_M) \) is monotonic, and thus that \( \text{ABAC}_M \) is monotonic.

Now, let \( \pi : Q^I \to \text{Three} \) be a monotonic ideal policy. We show that there exists a policy \( P(\pi) \) such that \( (\text{ABAC}_M, P(\pi)) \) realizes \( \pi \). \( (Q^I, \leq_3) \) is a finite partially ordered set, so we may enumerate its elements using a topological sort. That is we may write \( Q^I = \{ q_0, q_1, \ldots, q_n \} \), for some \( n \) determined by \( N \) and \( \text{dom}(\alpha), \alpha \in N \); and for all \( i \leq j, q_j \not\leq q_i \). In particular, we have \( q_0 = \emptyset \).

For any non-empty request \( q = \{(\alpha_1, v_1), \ldots, (\alpha_m, v_m)\} \), we define the policy \( t_q = (\alpha_1, v_1) \land \ldots \land (\alpha_m, v_m) \). Note that \( \llbracket t_q \rrbracket(q') = 1 \) for all \( q' \geq q \). Now, we have \( \llbracket t_{q_0} \rrbracket(q_0) = \bot \) for all \( q \). Moreover, for \( j > i > 0 \), we have \( \llbracket t_{q_i} \rrbracket(q_i) = 1 \) and \( \llbracket t_{q_j} \rrbracket(q_i) \neq 1 \). In other words, for every request \( q \) there is a value \( m \) such that \( \llbracket t_{q_m} \rrbracket(q) = 1 \) and \( \llbracket t_{q_j} \rrbracket(q) \neq 1 \) for all \( j > m \). We now define the policy

\[
\oplus_\pi(t_{q_1}, \ldots, t_{q_n}),
\]

where

\[
\oplus_\pi(d_1, \ldots, d_n) = \begin{cases} 
\pi(q_m) & \text{where } m = \max \{ i : d_i = 1 \} \\
\pi(\emptyset) & \text{otherwise.}
\end{cases}
\]

defines an \( n \)-ary operator \( \oplus_\pi \).

We now prove that \( \oplus_\pi \) is monotonic. Intuitively, we want to prove that the order over tuples of decisions implies the order over requests, in order to use the monotonicity of \( \pi \). Let \( d_1, \ldots, d_n \in \text{Three} \) and \( d'_1, \ldots, d'_n \in \text{Three} \) with \( d'_i \leq d_i, 1 \leq i \leq n \). Let \( q' \) and \( q \) be the requests identified by \( (d'_1, \ldots, d'_n) \) and \( (d_1, \ldots, d_n) \), respectively. By definition, we have \( \oplus_\pi(d'_1, \ldots, d'_n) = \pi(q') \) and \( \oplus_\pi(d_1, \ldots, d_n) = \pi(q) \). Furthermore, let \( m \) be the maximal index such that
\[ d'_m = 1, \] it follows that \( q' = q_m \). Since \( d'_m \leq d_m \), we can deduce that \( d_m = 1 \), implying that \( q_m \leq Q q \), that is, \( q' \leq Q q \). By hypothesis, \( \pi \) is monotonic, and thus we have \( \pi(q') \leq \pi(q) \), allowing us to conclude that \( \oplus_{\pi} \) is monotonic.

Finally, we know, by a result of Blamey [12], that any monotonic operator can be built from \( \{\neg, \land, \lor, \triangleright\} \). In other words, the policy \( \oplus_{\pi}(t_{q_1}, \ldots, t_{q_n}) \) belongs to \( \text{ABAC}_M \), and therefore we can conclude that \( \pi \) can be realized by \( (\text{ABAC}_M, \oplus_{\pi}(t_{q_1}, \ldots, t_{q_n})) \).

Theorem 9 demonstrates that all access control policies built and evaluated using \( \text{ABAC}_M \) are monotonic, and that all monotonic ideal policies can be realized by a policy in \( \text{ABAC}_M \). Hence, if policy monotonicity is an important feature for a system designer, then \( \text{ABAC}_M \) provides that guarantee. However, \( \text{ABAC}_M \) is not complete, since some (non-monotonic) ideal policies cannot be built from it. We propose in the following section a complete model.

### 4.2 \( \text{ABAC}_C \): A complete model

In some situations, one might want to define non-monotonic policies or we might want to consider a query set in which the same attribute can have different values within a given query. In such situations, we consider the model \( \text{ABAC}_C \) which, in addition to being complete, is defined over the set of queries \( Q^* \).

**Definition 10** \( \text{ABAC}_C \) is defined to be \( (Q^*, A(\mathcal{N}), \text{Three}, \{\neg, \sim, \lor\}, \llbracket \cdot \rrbracket) \).

**Theorem 11** \( \text{ABAC}_C \) is complete.

*Proof.* The structure of this proof is similar to that of Theorem 9 with the difference that the ideal policy \( \pi \) need not be monotonic. We show that there exists a policy \( p(\pi) \) in \( \text{ABAC}_C \) that realizes \( \pi \).

Concretely, we consider the enumeration over the requests, and we build the same operator \( \oplus_{\pi} \). However, in this case, we do not need to prove the monotonicity of \( \oplus_{\pi} \), and we use instead the fact that the logic \( \{1, 0, \bot, \neg, \sim, \lor\} \) is functionally complete [2], which ensures that \( \oplus_{\pi} \) can be built from \( \{\neg, \sim, \lor\} \).

It is trivial to see that by considering \( Q^* \), we lose the monotonicity with respect to the inclusion ordering \( \leq_Q \). In particular, for \( v, v' \in \text{dom}(\alpha) \) with \( v \neq v' \), we have \( \{(\alpha, v')\} \leq_Q \{(\alpha, v), (\alpha, v')\} \), but \( 0 = \llbracket(\alpha, v]\llbracket(\{(\alpha, v')\}) \neq \llbracket(\alpha, v]\llbracket(\{(\alpha, v), (\alpha, v')\}) = 1 \).

The function \( \llbracket \cdot \rrbracket \) is monotonic for atomic policies and requests in \( Q^* \) if we adopt the ordering \( \bot < 0 < 1 \). This means that omitting attributes can cause the evaluation of an atomic policy to change from 1 to 0 or \( \bot \), or from 0 to \( \bot \). While this seems to be reasonable, when combined with operators such as \( \Delta \), omitting attributes can cause the evaluation of a policy to change from 0 to 1. (Thus a user may be able to construct a request \( q \subseteq q' \) that is allowed when \( q' \) is not.

Any non-monotonic language, such as XACML, incorporates this vulnerability.)
4.3 ABAC with Explicit Conflict

The above choice to evaluate an atomic policy \((\alpha, v)\) to 1 if both \((\alpha, v)\) and \((\alpha, v')\) belongs to the query with \(v \neq v'\) and \(v, v' \in \text{dom}(\alpha)\) could be regarded as being logically inconsistent in the sense that the request also contains a non-matching value. One could equally well argue, for example, that the request should evaluate to 0.

In order to cope with such situations, we might, therefore, choose to work with the 4-valued logic \(\text{Four} = \{1, 0, \bot, \top\}\), using \(\top\) to denote conflicting information (in contrast to \(\bot\) which signifies lack of information). We introduce the ordering \(\leq 4\), where \(d_1 \leq 4 d_2\) if, and only if, \(d_1 = \bot\), \(d_1 = d_2\) or \(d_2 = \top\), and we define

\[
[(\alpha, v)]=\begin{cases} 
\top & \text{if } q \ni (\alpha, v), (\alpha, v') \text{ such that } v, v' \in \text{dom}(\alpha) \text{ and } v' \neq v, \\
1 & \text{if } q \ni (\alpha, v) \text{ and } q \not\ni (\alpha, v') \text{ such that } v' \neq v, \\
0 & \text{if } q \not\ni (\alpha, v) \text{ and } q \ni (\alpha, v') \text{ such that } v' \neq v, \\
\bot & \text{otherwise.}
\end{cases}
\]

The definition of the policy operators \(\nabla\) and \(\triangle\) can be extended to unary operators on \(\text{Four}\), where

\[
\nabla d = \begin{cases} 
1 & \text{if } d = \top, \\
d & \text{otherwise;}
\end{cases} \quad \Delta d = \begin{cases} 
0 & \text{if } d = \top, \\
d & \text{otherwise.}
\end{cases}
\]

Then the policy \(\nabla(\alpha, v)\) allows a request \(q\) whenever \(q\) contains a matching attribute, while the policy \(\Delta(\alpha, v)\) denies a request \(q\) whenever \(q\) contains a non-matching attribute value. Although the notion of conflicting policy decision has already been studied [10], to the best of our knowledge, this is the first time this notion of conflict has been used to evaluate targets. Intuitively, a conflict indicates that the request provides too much information for this particular policy. It is worth observing that atomic policies are monotonic with respect to the ordering \(\leq 4\), i.e., for all requests \(q\) and \(q'\) such that \(q \leq 4 q'\) and for all atomic policies \((\alpha, v)\), we have:

\[
[(\alpha, v)](q) \leq 4 [(\alpha, v)](q').
\]

A possible way to extend the operators defined in Table 1 is to consider the value \(\top\) as absorbing: for any operator \(\oplus : \text{Three}^k \to \text{Three}\), we define the operator \(\hat{\oplus} : \text{Four}^k \to \text{Four}\) as follows:

\[
\hat{\oplus}(d_1, \ldots, d_k) = \begin{cases} 
\top & \text{if } d_i = \top \text{ for some } i \in [1, k], \\
\oplus(d_1, \ldots, d_k) & \text{otherwise.}
\end{cases}
\]

Clearly, given an operator \(\oplus\) defined over \(\text{Three}\), if \(\oplus\) is monotonic according to \(\leq 3\), then \(\hat{\oplus}\) is also monotonic with respect to \(\leq 4\). It follows that we can still
safely use the operators generated by the operators ¬, ∧, ∨ and  ≡, and we can
deduce that any realizable policy is also monotonic. However, we lose the result
of monotonic completeness, and we can no longer ensure that any monotonic
operator can be generated from this set of operators. Obtaining such a result
requires a deeper study of four-valued logic, and we leave it for future work.

5 Related Work

Much of the work on specification of access control languages can be traced back
to the early work of Woo and Lam, which considered the possibility that different
policy components might evaluate to different authorization decisions [14]. More
recent work has considered larger sets of policy decisions or more complex policy
operators (or both), and propose a formal representation of the corresponding
metamodel [15,16,10,13,17,18,19,20,21,22,5,23]. The “metamodels” in the lit-
erature are really attempts to fix an authorization vocabulary, by identifying the
sets and relations that will be used to define access control policies.

In contrast, our framework makes very few assumptions about access control
models and policies that are written in the context of a model. In this, our frame-
work most closely resembles the work of Tschantz and Krishnamurthi [11], which
considered a number of properties of a policy language, including determinism,
totality, safety and monotonicity.

The notion of a monotonic operator (as defined by [11]) is somewhat different
from ours. This is in part because a different ordering on the set of decisions Three
is used and because monotonicity is concerned with the inclusion of sub-policies
and the effect this has on policy evaluation. This contrasts with our approach,
where we are concerned with whether the exclusion of information from a re-
quest can influence the decision returned. (In fact, our concept of monotonicity
is closer to the notion of safety defined in [11]: if a request q is “lower” than q′,
then the decision returned for q is “lower” than that of q′.) We would express
their notion of monotonicity in the following way: a policy operator ⊕ is mono-
tonic (in the context of model M) if for all p₁, . . . , pₜ ∈ P(M) and all q ∈ Q, if
⊕(p₁ . . . pₜ)(q) ∈ {1, 0}, then ⊕(p₁ . . . pᵢ, pᵢ₊₁, . . . , pₜ)(q) ̸= ⊥ for any i and any
policy p′ ∈ P(M). Moreover, our framework is concerned with arbitrary autho-
rization vocabularies and queries, unlike that of Tschantz and Krishnamurthi,
which focused on the standard subject-object-action request format. The only
assumption we make is that all policies can be represented using a tree-like struc-
ture and that policy decisions can be computed from the values assigned to leaf
nodes and the interpretation of the operators at each non-leaf node.

In addition, we define the notion of completeness of a model, which is con-
cerned with the expressivity of the policy operators. There exists prior work on
comparing the expressive power of different access control models or the extent
to which one model is able to simulate another [24,25,26]. In this paper, we show
how our framework enables us to establish whether a model based on a particular
set of atomic policies, decision set and policy connectives is complete. We can,
therefore, compare the completeness of two different models by, for example, fix-
ing an authorization vocabulary and comparing the completeness of models that differ in one or more of the models’ components (that is, ones that differ in the set of connectives, decision sets, atomic policies and authorization semantics).

While this is similar in spirit to earlier work, this is not the primary aim of this paper, although it would certainly be a fertile area for future research.

6 Concluding Remarks

We have presented a generic framework for specifying access control controls, within which a large variety of access control models arise as special cases, and which allows us to reason about the global properties of such systems. A major strength of our approach is that we do not provide “yet another access control language”. The framework is not intended to provide an off-the-shelf policy language or PDP (unlike XACML, for example), nor is it intended to be an access control model (in the style of RBAC96, say). Rather, we try to model all aspects of an access control system at an abstract level and to provide a framework that can be instantiated in many different ways, depending on the choices made for request attributes, atomic policies, policy decisions and policy evaluation functions. In doing so we are able to identify (i) how and why an access control system may fail to be sufficiently expressive (completeness), and (ii) how and why having an expressive access control system may lead to vulnerabilities (monotonicity).

There are many opportunities for future work. The notions of monotonicity and completeness are examples of general properties of an access control model that we can characterize formally within our framework. We have already noted that there are at least two alternative semantics for atomic policies having the form \((\alpha, v)\) for a three-valued decision set and even more alternatives for a four-valued decision set. It would be interesting to see how these alternative semantics affect monotonicity and completeness. We would like to study the composition of access control models, and under what circumstances composition preserves monotonicity and completeness. Further properties that are of interest include policy equivalence, policy ordering (where, informally, one policy \(P_1\) is “more restrictive” than \(P_2\) if it denies every request that is denied by \(P_2\)), which may allow us to define what it means for a realizable policy to be “optimal” with respect to an (unrealizable) ideal policy. Moreover, our definition of monotonicity is dependent on the ordering on the set of decisions. Monotonicity, in the context of the ordering \(0 < \perp < 1\), for example, is a stronger property than the one we have considered in this paper. Again, it would be interesting to investigate the appropriateness of different forms of monotonicity. Furthermore, although XACML is proven not to be monotonic, it is not known under which conditions it can be monotonically-complete, and if additional operators are needed to prove this property, which is also likely to depend on the decision orderings considered.

In this paper, we have assumed that there exists an ideal policy and that such a policy is fixed. Generally, however, a system evolves over time, and an access control policy will need to be updated to cope with changes to that system that
affect the users, resources, or context. Thus it may be more realistic to specify an initial ideal policy, which might be extremely simple, and the access control policy that best approximates it, and then define rules by which the access control policy may evolve. With this in mind, it makes sense to regard the access control policy (or components thereof) as a protected object. Security is then defined in terms of properties that “reachable” access control policies must satisfy. Typical examples of such properties are “liveness” and “safety” [27]. Including administrative policies within our framework and investigating properties such as liveness and safety will be an important aspect of our future work in this area.

References

1. M. Bishop, *Computer Security: Art and Science.* Addison-Wesley, 2002.
2. J. Crampton and C. Morisset, “PTaCL: A language for attribute-based access control in open systems,” in *Principles of Security and Trust - First International Conference, (POST 2012), Proceedings*, ser. Lecture Notes in Computer Science, vol. 7215, 2012, pp. 390–409.
3. A. Griesmayer and C. Morisset, “Automated certification of authorisation policy resistance,” in *ESORICS*, ser. Lecture Notes in Computer Science, J. Crampton, S. Jajodia, and K. Mayes, Eds., vol. 8134. Springer, 2013, pp. 574–591.
4. D. Ferraiolo and V. Atluri, “A meta model for access control: why is it needed and is it even possible to achieve?” in *Proceedings of the 13th ACM symposium on Access control models and technologies.* ACM, 2008, pp. 153–154.
5. *eXtensible Access Control Markup Language (XACML) Version 3.0,* OASIS, 2010, committee Specification 01.
6. D. F. Ferraiolo and D. R. Kuhn, “Role-based access control,” in *Proceedings of the 15th National Computer Security Conference,* 1992, pp. 554–563.
7. S. Barker, “The next 700 access control models or a unifying meta-model?” in *SACMAT,* D. F. Ferraiolo and I. Ray, Eds. ACM, 2006, pp. 160–169.
8. M. Harrison, W. Ruzzo, and J. Ullman, “Protection in operating systems,” *Communications of the ACM,* vol. 19, no. 8, pp. 461–471, 1976.
9. P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo, “An algebra for fine-grained integration of XACML policies,” in *SACMAT,* B. Carminati and J. Joshi, Eds. ACM, 2009, pp. 63–72.
10. G. Bruns and M. Huth, “Access control via Belnap logic: Intuitive, expressive, and analyzable policy composition,” *ACM Transactions on Information and System Security,* vol. 14, no. 1, p. 9, 2011.
11. M. C. Tschantz and S. Krishnamurthi, “Towards reasonability properties for access-control policy languages,” in *SACMAT,* D. F. Ferraiolo and I. Ray, Eds. ACM, 2006, pp. 160–169.
12. S. Blamey, *Handbook of Philosophical Logic.* Kluwer Academic Publishers, 2002, vol. 5, ch. Partial Logic, pp. 261–353.
13. J. Crampton and M. Huth, “An authorization framework resilient to policy evaluation failures,” in *ESORICS*, ser. Lecture Notes in Computer Science, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds., vol. 6345. Springer, 2010, pp. 472–487.
14. T. Y. C. Woo and S. S. Lam, “Authorizations in distributed systems: A new approach,” *Journal of Computer Security,* vol. 2, no. 2–3, pp. 107–136, 1993.
15. E. Bertino, B. Catania, E. Ferrari, and P. Perlasca, “A logical framework for reasoning about access control models,” *ACM Transactions on Information and System Security*, vol. 6, no. 1, pp. 71–127, 2003.

16. P. Bonatti, S. De Capitani Di Vimercati, and P. Samarati, “An algebra for composing access control policies,” *ACM Transactions on Information and System Security*, vol. 5, no. 1, pp. 1–35, 2002.

17. J. Crampton and M. Huth, “A framework for the modular specification and orchestration of authorization policies,” in *NordSec*, ser. Lecture Notes in Computer Science, T. Aura, K. Järvinen, and K. Nyberg, Eds., vol. 7127. Springer, 2010.

18. N. Damianou, N. Dulay, E. Lupu, and M. Sloman, “The Ponder policy specification language,” in *POLICY*, ser. Lecture Notes in Computer Science, M. Sloman, J. Lobo, and E. Lupu, Eds., vol. 1995. Springer, 2001, pp. 18–38.

19. S. Jajodia, P. Samarati, M. Sapino, and V. Subrahmanian, “Flexible support for multiple access control policies,” *ACM Transactions on Database Systems*, vol. 26, no. 2, pp. 214–260, 2001.

20. N. Li, Q. Wang, W. H. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin, “Access control policy combining: theory meets practice,” in *SACMAT*, B. Carminati and J. Joshi, Eds. ACM, 2009, pp. 135–144.

21. Q. Ni, E. Bertino, and J. Lobo, “D-algebra for composing access control policy decisions,” in *ASIACC*, W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadharajan, Eds. ACM, 2009, pp. 298–309.

22. *eXtensible Access Control Markup Language (XACML) Version 2.0*, OASIS, 2005, committee Specification.

23. D. Wijesekera and S. Jajodia, “A propositional policy algebra for access control,” *ACM Transactions on Information and System Security*, vol. 6, no. 2, pp. 286–235, 2003.

24. L. Habib, M. Jaume, and C. Morisset, “A formal comparison of the Bell & LaPadula and RBAC models,” in *IAC*, M. Rak, A. Abraham, and V. Casola, Eds. IEEE Computer Society, 2008, pp. 3–8.

25. S. Osborn, R. Sandhu, and Q. Munawer, “Configuring role-based access control to enforce mandatory and discretionary access control policies,” *ACM Transactions on Information and System Security*, vol. 3, no. 2, pp. 85–106, 2000.

26. M. V. Tripunitara and N. Li, “A theory for comparing the expressive power of access control models,” *Journal of Computer Security*, vol. 15, no. 2, pp. 231–272, 2007.

27. F. B. Schneider, “Enforceable security policies,” *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, pp. 30–50, 2000.

28. B. Carminati and J. Joshi, Eds., *SACMAT 2009, 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, June 3-5, 2009, Proceedings*. ACM, 2009.