The Clarity of Languages for Access-Control Policies

by
Michael Carl Tschantz

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Department of Computer Science at Brown University

Providence, Rhode Island
May 2005

Sections 4.1 and 4.2 of this work were previously published in "The Soundness and Completeness of Margrave with Respect to a Subset of XACML" by Michael Matthew Greenberg, Casey Marks, Leo Alexander Meyerovich, and Michael Carl Tschantz (Technical Report CS-05-0, Providence: Brown University, April, 2005).

Abstract

Languages for the specification of access-control policies should support language features that allow for policies to be written in a clear manner. This work presents a set of language features found in current access-control languages and formalizes a set of intuitive properties the author believes to be relevant to policy clarity. The author analyzes access-control languages with respect to the presented features and properties.

Acknowledgements

I thank my advisor, Shriram, and second reader, Professor Steve Reiss. Conversations with Professor Kathi Fisler and Ms Vicky Weissman have aided me. I have benefited from the work of Messrs. Michael M. Greenberg, Casey Marks, and Leo A. Meyerovich.

Notation

$\mathcal{P}(S)$ is the powerset of a set S.

S^* is the set containing all sequences of zero or more elements of a set S.

$\circ$ is the empty sequence.

$s\ s^*$ is the sequence with head s and tail s^*.

$\bot$ is bottom, logical contradiction.

Contents

Chapter 1

Introduction

Web applications and multi-user systems make controlling the actions a user may perform on each resource critical for the security of a computer system. Since the requirements of these controls often change and apply to multiple applications, programmers increasingly abstract them from the application code to form an access-control policies written in domain-specific languages. In general, an access-control policy dictates a function from requests for access to decisions about whether access should be granted or not.

Completing requirements of expressive power and computational speed makes the design of languages for the specification of access-control languages a balancing act. Attempts to produce languages for the expression of such policies has largely followed one of two routes: 1) using some subset of first-order logic or 2) producing a highly specialized, *ad hoc* language.

The first approach modifies an unspecialized logic language, such as first-order logic or Datalog, for the requirements of access-control. Designers of languages that use first-order logic as a starting point must restrict it to a computationally tractable yet sufficient expressive subset (see *e.g.*, [7]). Those starting with a subset of first-order logic, such as Datalog, must argument it with the primitives needed for access-control (see *e.g.*, [2]).

The second approach, has produced very limited languages such as XACML (an OASIS standard [9] with backing from Sun Microsystems and others) and EPEL (an IBM standard [10]). For example, to express with XACML the common paradigm of hierarchical role-based access control (RBAC) [11], one must use a cumbersome encoding [1]. However, in some ways the policies written in XACML are more transparent than policies written in languages based on first-order logic.

Formalizing the intuition that some languages produce clearer policies than others is the subject of the paper. The set of properties produced by this formalization will be helpful to both language designers and users. Designers can use them a set of goals for their language to meet. Users completing which language to select may find use them to select among those languages that are expressive enough for their policies.

First, some of the design decisions that must be made while designing an access-control language are presented in Chapter 2. Properties that formalize intuitions about the clarity of access-control

languages form Chapter 3. In Chapters 4 and 5, XACML and a language based on first-order logic are presented and analyzed. Related works are presented in Chapter 6 and the conclusion in Chapter 7.

Chapter 2

Features of Access-Control Languages

Many different access-control languages have been proposed. Although they each support a different set of language features, they have enough in common to permit comparison. First, this section informally presents these similarities. Second, it presents an example policy and three interpretations consistent with the common features. Lastly, the language features that must be selected or discarded to disambiguate the policy are delineated.

2.1 Common Features of Access-Control Languages

An access-control language must provide a way of describing the different forms of access and the environment in which they could occur. This information forms a *request*. Many languages break requests into four different parts:

Subject the person or process making the request,

Resource the object, subsystem, person, or process that would be affected (e.g., a file name or a process id),

Action the command or change that would be executed on the resource by the subject (depending on the granularity of the control this could be course like "write" or fine like "write 'x' at line 10"), and

Environment describes any other relevant information such as the time of day, location, or the the previous actions of the subject.

The first three of these make up the form of requested access while the last gives the context in which this access would be exercised.

Languages must also provide a set of *decisions* to convey if a request should be granted or not. Such a set must include some decisions that grant access and some that prohibit access. A policy will associate with each request a decision (or in the case of nondeterministic policies, a policy will relate each request with some number of decisions).

The policy of a firm often consists of fragments from a variety of sources such as the legal department, accounting, and executives. All of these fragments may be encoded as a formal policy in an access-control language. Given a set of such policies, one would like the language to provide *policy connectives* to compose these various policies into one policy. Thus, we assume that access-control languages provide a set of such policy connectives and that policies may consist of other policies. Policies contained within a policy, will be called *sub-policies* of containing policy. Policies that do not contain any sub-policies will be called *atomic policies*.

Within this framework, many choices for how a policy expresses the association of decisions with requests are possible. To demonstrate the consequences of these choices, an example is presented next.

2.2 Motivating Example

Consider the following informal policy written in natural language:

1. If the subject is a faculty member, then permit that subject to assign grades.
2. If the subject is a student, then do not permit that subject to assign grades.
3. If the subject is not a faculty member, then permit that subject to enroll in courses.

One might represent that policy as follows:

$$\texttt{faculty}(s) \implies \mathsf{Permitted}(s, \texttt{grades}, \texttt{assign}) \tag{2.1}$$

$$\texttt{student}(s) \implies \neg\mathsf{Permitted}(s, \texttt{grades}, \texttt{assign}) \tag{2.2}$$

$$\neg\texttt{faculty}(s) \implies \mathsf{Permitted}(s, \texttt{courses}, \texttt{enroll}) \tag{2.3}$$

Let the above formalization be $\wp$ and the first line of the policy be sub-policy $\wp_1$, the second, $\wp_2$, and the third $\wp_3$.

Consider the following natural-language request:

> A student requests to enroll in courses.

For the time, assume that requests simply lists the subject, resource, and action by name if possible and by variable if the name is unknown, along with any other known facts. what is know about the subject, resource, and action. The following is a representation of the natural-language request:

$$(s, \texttt{courses}, \texttt{enroll}) \quad \text{with} \quad \texttt{student}(s) \tag{2.4}$$

Given this request, does the policy grant access? At least three interpretations of the policy are possible:

1. One might conclude that by $\wp_3$, access is granted. The request does not show the subject being a faculty member, and, thus, the last line applies and decision to permit access is produced. However, this relies on the assumption that since the request does not show the subject being a faculty, that the subject is in fact not a faculty. One could drop this assumption.

2. One could conclude that the policy does not apply to the request. One would reason that $\wp_1$ and $\wp_2$ do not apply since they are dealing with assigning grades and not enrolling in courses. Furthermore, one could conclude that $\wp_3$ does not apply since the request does not prove that the subject is not on the faculty. To do so, the request would have been

$$(s, \texttt{courses}, \texttt{enroll}) \quad \text{with} \quad \texttt{student}(s) \wedge \neg \texttt{faculty}(s) \tag{2.5}$$

   After concluding that the policy does not apply to the request, one could then conclude that access is granted by default, is prohibited by default, or that it is nondeterministic.

3. One could conclude by reasoning different than that used in the first interpretation that the request is granted. As in the second interpretation, one could conclude that the request allow does not establish if the subject is a member of the faculty or not. However, one might then go on to note that if the subject was a faculty member, the first two lines together would yield a contradiction: $\wp_1$ would imply that the subject could enroll in courses and $\wp_2$ would imply that the subject could not. Thus, one could conclude that a student who is on the faculty is impossible. Since the subject of the request is clearly a student, he must not be faculty member. Thus, $\wp_3$ applies and access is granted.

Which of these interpretations is the one intended by the policy depends what choices are made about the language.

2.3 The Structure of Requests

In the above example, requests had a simple form. Under the second interpretation, this form allowed requests to contain too little information to determine which of the sub-policies of $\wp$ applied. Other forms might provide more or less ambiguous requests. For example, if every request had to provide the subject by name and provide a list every possible subject and if that subject was faculty member and/or a student, the above ambiguous would not exist.

Although such requirements might be practical in a small system with a center database of users, it might not be for a large distributed system. In general, collecting information germane to a request might be time consuming. Furthermore, discovering the smallest set of information which is germane can also be time consuming for complex policies. Thus, systems might want to allow requests to be ambiguous by not containing all the information that is possible relevant. In such system, care must be taken to ensure that access is not mistakenly granted, a subject formalized in Section 3.5.

2.4 The Decision Set

The set of decisions of a languages often contains *permit*, which implies that the request should be granted, and *deny*, which implies that the request should not to be granted. However, some languages might provide more decisions or refinements of permit and deny that convey additional information about why a request was granted or not.

Some languages may include a decision of *not applicable* to indicate that the policy remains sentient on a request. For example, under the second interpretation of $\wp$, the policy appeared not to apply to the given request. In this case, the decision of *not applicable* might be returned by some languages.

Despite being syntactically correct, some requests might not have a logical interpretation under a given policy. For example, a request of

$$(s, \texttt{grades}, \texttt{assign}) \quad \text{with} \quad \texttt{faculty}(s) \wedge \texttt{student}(s) \tag{2.6}$$

under the third interpretation of $\wp$ seems to contradict the policy itself. Other times, a request might contain illogical values or require undefined computation (such as division by zero). Furthermore, for generality, a system might like to assign a decision to inputs that are not syntactically requests. In such cases, a decision of *error* or some refinement of it might be appropriate.

One may view the fact that an error state is reached given a syntactically correct request to be a weakness in the policy. However, one may also take it to be a statement about the world in which the policy is being used: that no such requests can exist. In sense, error decisions can be used to enforce preconditions much as exceptions do in programming languages.

Whether the decisions of not applicable and error entail granting or preventing access might vary from language to language. However, to err on the side of caution, it seems more reasonable to not grant access despite that access is not explicitly denied.

Some languages, in practice, treat not applicable as a weak deny. Such languages stipulate that if two sub-policies of one policy are checked one returns not applicable and the other some other result, that the other decision overrides the not applicable. Decisions of error are often treated in the opposite manner, as a strong deny that overrides other policies producing permit, deny, or not applicable. However, many different ways exist resolve conflicts when combining policies.

2.5 Policy Connectives

It is unclear if the request given on line 2.6 should be permitted or denied since $\wp_1$ and $\wp_2$ contradict one another on this issue. The method of combining the three sub-policies of $\wp$ determines how to resolve this conflict. Policy connectives play the role of proving the needed method of combination. Some languages, like the hypothetical language in which $\wp$ is written, might have only one policy-connective that is implicitly applied. Other languages, allow different policies and sub-policies to use different ones. For example, if a policy has sub-policies nested inside of sub-policies, the outer liner may be resolved differently than the inner layer.

Some of the possible policy connectives are:

Permit Overrides If any of the sub-policies produces a permit, return only permit. Otherwise, if any produces a deny, return only deny. Else, return not applicable.

Deny Overrides If any of the sub-policies produces a deny, return only deny. Otherwise, if any produces a permit, return only permit. Else, return not applicable.

First Applicable Return the the decision reached by the first sub-policy to reach one other than not applicable.

All Seen Return a set containing the decisions reached by all the sub-policies.

Either Permit or Deny Nondeterministically select one of the produced decisions to return.

Error Return a error if the sub-policies produces both permit and deny. Otherwise return permit if produced, or deny if produced. Else, return not applicable.

And Conjoin the sub-policies together by logical And and return the implied decision(s).

Additional more complex connectives may be found in the work of De Capitani di Vimercati *et al.* [3]. The nature of the connectives available in a language can greatly impact the clarity of policies written in it.

Notice that many of the above connective behave the same in the absence of the decision of deny. One might conclude from this observation, that allowing the explicit denial of a request as an desirable complication of the language.

2.6 Explicit Denial

Policies must provide a way to associate each decision with a subset of requests. Under most languages, some requests will explicitly be placed into one of these subsets. The other requests will be placed into one of these subset by default. For safety, this default subset should be associated with decision that does not grant access creating a closed policy [3]. The decision of not applicable is intended for this purpose.

With requests being not granted by default, the language must provide some way to specify requests to be granted. Some languages may choose to only allow statements of this type and to disallow statements explicitly implying that access should not be granted for some set of request. The uniformity of statements in such languages, might make the policy easier to read.

Some languages may also allow for the policy to explicitly specify requests for which access should not granted. Such constructs could be useful in determining which decision should be issued if more than one prohibits access (*e.g.*, deny or error). Furthermore, such statements could quickly rule out exceptional cases. For example, such a construct is employed in $\wp$: $\wp_2$ explicitly states that some requests are to be be denied. These benefits comes at the cost of the complication of policy combination noted above.

2.7 Checking for the Absence of Attributes

Consider the meaning of $\neg\texttt{faculty}(s)$ in $\wp_3$. This is testing not for the presence of an attribute, but rather the absence of one. Much like explicit denial, some languages may opt not support this construct. Indeed in some contexts, such as certificate passing systems in which a certificate may be withheld, testing for the absence of an attribute may be impossible or impractical.[1]

If a language does allow for the checking for the absence of attributes, must decides on a semantics for the absence of an attribute. Does the absence of a statement affirming the presence of that attribute from request satisfy the check or is the presence of the negation of the presence of that attribute required? For example, one could take $\neg\texttt{faculty}(s)$ to means either that the request does contain $\texttt{faculty}(s)$ or that the request contains $\neg\texttt{faculty}(s)$. The difference is whether the absence of an attribute from a request implies that the actual absence of the attribute. If yes, we will call the absence of attributes *implicit*; if not, the absence of attributes must be made *explicit*.

One may conclude that $\wp_3$ does apply to the request (the antecedent holds) under implicit absence of attributes, but it does not under explicit absence. Notice that the applicability of $\wp_1$ and $\wp_2$ does not depend on of the implicit explicit absence of attributes is used.

[1] One may argue that certificate passing systems may use negative certificates to achieve the checking of attribute absence. Whether this captures the notion of the absence of an attribute or just the presence of another related attribute is unclear. For example, one could conceivable hold both a positive and a negative certificate for an attribute. Furthermore, having to issue both positive and negative certificates for attributes is undesirable.

Chapter 3

Properties Relevant to Clarity

In Chapter 2, a variety of informal notations were introduced. The general features that access-control languages have in common were presented and the several of the constructs found in some but not all languages were delineated. In this chapter, the definition of an access-control language is formalized in such a way as to accommodate all the features discussed thus far. This formalization is then used to present several properties relevant to clarity that some access-control language may process.

3.1 Formalization of Access-Control Languages

Define an access-control languages to be a tuple $\mathcal{L} = (Q, P, G, N, \langle\langle\cdot\rangle\rangle)$ with

Q a set of requests,

P a set of policies,

G the set of decisions that stipulate that access should be granted (granting decisions),

N the set of decisions that stipulate that access should be prevented (non-granting decisions),

$\langle\langle\cdot\rangle\rangle$ a function from P to a relation between Q and $G \cup N$,

and $G \cap N = \emptyset$. When clear from context, the above symbols will be referenced without explicitly relating them to $\mathcal{L}$ and D will be used for $G \cup N$. The function $\langle\langle\cdot\rangle\rangle : P \to \mathcal{P}(Q \times D)$ gives the meaning of policy p and we write $q\langle\langle p\rangle\rangle d$ for $q \in Q$, $p \in P$, and $d \in D$, when p assigns a decision of d to the request q. Given $\mathcal{L}$ define the partial order $\leq$ on D to be such that $d \leq d'$ if either $d, d' \in N$, $d, d' \in G$, or $d \in N$ and $d' \in G$.

We assume that policies may be constructed from either atomic policies or other policies combined by some policy connective, which indicates how the sub-policies should be combined to make one policy. Let p be policy that consists of the sub-policies p_i with $1 \leq i \leq n$. Let the sub-policies

be connected by the policy connective $\oplus$. Although the actual syntax of languages differ, $p = \oplus(p_1, p_2, \ldots, p_n)$ will represent this condition.[1]

Since policies are composed of sub-policies, $\langle\!\langle \cdot \rangle\!\rangle$ can be extended to provide the meaning of the sub-policies by allowing for contextual information like variable bindings to be considered. Let Σ be the set of all such contextual information. Let $\langle\!\langle p \rangle\!\rangle^\sigma$ be the meaning of sub-policy p under the contextual information $\sigma \in \Sigma$. If $p = \oplus(p_1, p_2, \ldots, p_n)$, then $\langle\!\langle p \rangle\!\rangle^\sigma = \langle\!\langle \oplus \rangle\!\rangle^\sigma(\langle\!\langle p_1 \rangle\!\rangle^\sigma, \langle\!\langle p_2 \rangle\!\rangle^\sigma, \ldots, \langle\!\langle p_n \rangle\!\rangle^\sigma)$.[2]

3.2 Determinism and Nondeterminism

Definition 3.1. *Given a language* $\mathcal{L} = (Q, P, G, N, \langle\!\langle \cdot \rangle\!\rangle)$*, if*

$$\forall p \in P, \forall q \in Q, \forall d, d' \in D, q\langle\!\langle p \rangle\!\rangle d \wedge q\langle\!\langle p \rangle\!\rangle d' \implies d = d' \tag{3.1}$$

then $\mathcal{L}$ *is* deterministic*. Otherwise, it is* nondeterministic*.*

For a deterministic language, we can define $[\![\cdot]\!]^\sigma : P \to (Q \to D)$ to be $\lambda q \,.\, d \in D \,\text{s.t.}\, q\langle\!\langle p \rangle\!\rangle^\sigma d$. For deterministic language, $[\![\cdot]\!]$ may be given instead of $\langle\!\langle \cdot \rangle\!\rangle$ to define the language.

For the remainder of the paper, we assume that all languages are deterministic unless otherwise noted.[3]

3.3 Homomorphism

Consider the third interpretation of $\wp$. Under this interpretation, the meaning of $\wp$ can only to determined by looking at the interactions of the different sub-policies as whole. Notice that any one of these sub-policies would produce a decision of not applicable in isolation, and yet together they interact to produce a permit decision. The third interpretation inhabits using local reasoning about each sub-policy for reasoning about the whole policy. This increases the possibility of unintended results from combining sub-policies into a policy.

The alternative, as found in the first two interpretations, is for the sub-policies to be combined in such a way that only the result of each in isolation matters. This property is formalized as follows:

Definition 3.2. *We say that the policy connective* $\oplus$ *of a language* $\mathcal{L}$ *is* homomorphic *iff*

$$\exists \boxplus : D^* \to D, \forall p_1, p_2, \ldots, p_n \in P, \forall \sigma \in \Sigma, \forall q \in Q,$$
$$[\![\oplus(p_1, p_2, \ldots, p_n)]\!]^\sigma(q) = \boxplus([\![p_1]\!]^\sigma(q), [\![p_2]\!]^\sigma(q), \ldots, [\![p_n]\!]^\sigma(q)) \tag{3.2}$$

If all the connectives of $\mathcal{L}$ *are homomorphic, then* $\mathcal{L}$ *has the* homomorphic *property.*

[1] I assume that the set of connectives in a given language $\mathcal{L} = (Q, P, G, N, \langle\!\langle \cdot \rangle\!\rangle)$ is clear from the structure of P and $\langle\!\langle \cdot \rangle\!\rangle$. If this is not the case for a language, one could explicitly add it to the definition of an access-control language.

[2] As with the set of connectives, I assume that Σ may be inferred given a language, but could be explicitly included if needed.

[3] If it were not for the existence of a nondeterministic language, XACML with obligations, I would not have considered them at all.

One might be tempted to define the homomorphic property as follows:

Definition 3.3. *A policy connective* $\oplus$ *is* weakly homomorphic *iff*

$$\exists \boxtimes : (Q \to D)^* \to (Q \to D), \forall p_1, p_2, \ldots, p_n \in P, \forall \sigma \in \Sigma,$$
$$[\![\oplus(p_1, p_2, \ldots, p_n)]\!]^\sigma = \boxtimes([\![p_1]\!]^\sigma, [\![p_2]\!]^\sigma, \ldots, [\![p_n]\!]^\sigma) \quad (3.3)$$

If all the connectives of $\mathcal{L}$ *are weakly homomorphic, then* $\mathcal{L}$ *has the* weak homomorphic *property.*

Theorem 3.4. *If if a policy connective* $\oplus$ *of an access-control language* $\mathcal{L}$ *is homomorphic, then it is weakly homomorphic.*

Proof. To prove that $\oplus$ is weakly homomorphic, $\boxtimes : (Q \to D)^* \to (Q \to D)$ found in Equation 3.3 will be constructed from the $\boxplus : D^* \to D$ known to exist since $\oplus$ is homomorphic. Let

$$\boxtimes(f_1, f_2, \ldots, f_n) = \lambda q \,.\, \boxplus(f_1(q), f_2(q), \ldots, f_n(q)) \quad (3.4)$$

Then

$$\begin{aligned} \boxtimes([\![p_1]\!]^\sigma, [\![p_2]\!]^\sigma, \ldots, [\![p_n]\!]^\sigma) &= \lambda q \,.\, \boxplus([\![p_1]\!]^\sigma(q), [\![p_2]\!]^\sigma(q), \ldots, [\![p_3]\!]^\sigma(q)) && (3.5)\\ &= \lambda q \,.\, [\![\oplus(p_1, p_2, \ldots, p_n)]\!]^\sigma(q) && (3.6)\\ &= [\![\oplus(p_1, p_2, \ldots, p_n)]\!]^\sigma && (3.7) \end{aligned}$$

□

Theorem 3.5. *A policy connective* $\oplus$ *being weakly homomorphic does not imply that* $\oplus$ *is homomorphic.*

Proof. Consider a rather odd language that has only one unary policy connective, $\oplus$, atomic policies of the form (v) where v is a value, and requests are sets of values. Let the semantics be such that

$$[\![\oplus(p_1)]\!] = \lambda q \,. \begin{cases} \mathsf{permit} & \text{if} \quad [\![p_1]\!](\{v'\}) = \mathsf{permit} \\ \mathsf{deny} & \text{else} \end{cases} \quad (3.8)$$

for some distringished value v', and

$$[\![(v)]\!] = \lambda q \,. \begin{cases} \mathsf{permit} & \text{if} \quad v \in q \\ \mathsf{deny} & \text{else} \end{cases} \quad (3.9)$$

for atomic policies. Such a langauge is weakly homomorphic but not homomorphic.

The language is weakly homomorphic since for $\boxtimes$ such that

$$\boxtimes(f_1) = \begin{cases} \mathsf{permit} & \text{if} \quad f_1(\{v'\}) = \mathsf{permit} \\ \mathsf{deny} & \text{else} \end{cases} \quad (3.10)$$

clearly, $[\![\oplus(p_1)]\!] = \boxtimes([\![p_1]\!])$.

The language is not homomorphic. Assume that it is. Then there exist such a $\boxplus : D^* \to D$ to satifiy Equantion 3.2. For a requests $\{v\}$ and $\{v'\}$, where $v \neq v'$,

$$\mathsf{permit} = [\![\oplus((v'))]\!](\{v'\}) = \boxplus([\![(v')]\!](\{v'\})) = \boxplus(\mathsf{permit}) \tag{3.11}$$

$$\mathsf{deny} = [\![\oplus((v))]\!](\{v\}) = \boxplus([\![(v)]\!](\{v\})) = \boxplus(\mathsf{permit}) \tag{3.12}$$

A constradaction is reached since $\mathsf{permit} \neq \mathsf{deny}$. Thus, the langauge is not homomorphic. □

The weak homomorphic propriety will ensure that sub-policies with the same meaning in isolation will behave the same under the connective. Thus, a language with the weak homomorphic property is more clear than one without it since only the isolated meaning of the sub-policy must known to reason about its use under the connective. However, such a language may still have complicated connectives that will not allow a reader to reason about each sub-policy in isolation with respect to a request. A reader is likely to examine a policy by repeatedly asking what-if questions of the form "What if this request is given to the sub-policies" and observing the decision that each sub-policy yields from such a question. Thus, the homomorphic property, not just the weak homomorphic property, is important to clarity since it allows such a reader to put together all the decisions observed to acquire the decision the whole policy would yield.

3.4 Continuity

As noted at the end of Section 2.5, the decision of deny complicates the policy connectives. One of the reasons for this is that under connectives like deny overrides, a back and forth pattern can arise when considering the decision of the whole policy from the sub-policies. Consider reading each sub-policy one after the other. If the first one yields a decision of not applicable, the decision would a non-granting one if no other sub-policies existed. If the next sub-policy yields permit, this would change to a granting one. Finally, if the third sub-policy yields deny, the decision would change back to a non-granting one.

Note that this back and forth pattern is not exhibited by permit overrides since it is impossible be to go from a granting decision to a non-granting one under permit overrides. Thus, the formalization of this pattern focuses on the transition from a granting to a non-granting decision.

Definition 3.6. *Given $\mathcal{L}$, a connective $\oplus$ is* continuous *iff*

$$\forall p_1, p_2, \ldots, p_n, p_{n+1} \in P, \forall \sigma \in \Sigma, \forall q \in Q,$$
$$[\![\oplus(p_1, p_2, \ldots, p_n)]\!]^\sigma(q) \leq [\![\oplus(p_1, p_2, \ldots, p_i, p_{n+1}, p_{i+1}, \ldots, p_n)]\!]^\sigma(q) \tag{3.13}$$

We say $\mathcal{L}$ is continuous *if every connective is continuous.*

Adding another sub-policy to a continuous connective cannot change the decision from a granting one to a non-granting one.

3.5 Safety

In a system employing access control, the subsystem generating requests must collect the relative information to include in a request. Since collecting facts to embed in the request might be computationally expensive, the request generator, should be optimized to include only relevant facts. Since which facts are relevant is often unclear, a optimization might allow for the generator to produce an incomplete request in hopes that it will provide enough information to produce a different decision. Furthermore, overzealous optimizations and other coding errors, might result in requests being generated that do not contain all the relevant facts. Since unduly preventing access is preferred to unduly granting access, such incomplete requests should only result in access being granted if the complete one would have:

Definition 3.7. *Let $\subseteq$ be a partial ordering on requests of a language $\mathcal{L}$. A policy is* safe *with respect to $\subseteq$ iff*

$$\forall q, q' \in Q, \forall p \in P, q \subseteq q' \implies [\![p]\!](q) \leq [\![p]\!](q'). \tag{3.14}$$

Due to differences in the contents of a requests, for each language a different partial ordering $\subseteq$ will be of interest. The relation should be such that if $q \subseteq q'$, then q' contains more information than q.

For example, consider a language in which requests of sets of facts and the set of decisions is $\{\mathsf{permit}, \mathsf{deny}\}$. Then setting $\subseteq$ to be the subset partial ordering will be of interest since it matches the intuition of information content (assuming a non-contradictory set of facts). If the language is safe with respect to such a defined $\subseteq$, then one may omit facts from the request with out causing access to be granted which otherwise would not.

Although safety is not as closely tied to policy clarity as the homomorphic property or continuity, safe policies will still be easier to understand than the unsafe ones. With a safe policy, one does not need to consider the consequences of adding additional information to a granted request. For an unsafe policy, the system generating the requests and policy readers will incur the extra burden of including all relevant facts to prevent unintended grants of access. Informally, under a safe language, undue access will not be granted provided that the requests tell no lies; whereas under an unsafe language, the requests must additionally tell the whole truth.

Chapter 4

Simplified XACML

This and the next chapter presents access-control languages and applies to them the ideas of Chapters 2 and 3.

The eXtensible Access Control Markup Language (XACML) in it entirety does not fit well into the definition of access-control language given in Section 3.1 [9]. Full XACML includes a feature called obligations, which act as annotations on the the decisions of permit and deny. These annotations specifies actions that the system enforcing access-control must preform before granting or access or upon prohibiting access. Thus, an XACML policy may have effects beyond that of just granting or prohibiting access that the model presented fails to address.

However, the subset of XACML checkable by the verification and change impact tool Margrave [5] fits well in the framework presented. This subset is expressive enough to capture RBAC_0 [11]. This subset is presented and analyzed.

The syntax of Section 4.1 and Semantics of Section 4.2 were previously reported by M. Greenberg *et al.* [6].

4.1 Syntax

For readability we will not use an XML style syntax for our subset of XACML, but rather a scheme-like syntax.

XACML has two syntaxes: one for specifications of policies and one for requests. Since the former has a key word "Policy", we will call it the spec syntax. The latter is called the request syntax.

4.1.1 XACML Spec Syntax

The start nonterminal is S.

S	::=	C
C	::=	Ps | Pol
Ps	::=	(PolicySet Ca T C*)
Ca	::=	First-Applicable | Deny-Overrides | Permit-Overrides
T	::=	((Sub) (Res) (Act))
Sub	::=	Any | Allow$^+$
Res	::=	Any | Allow$^+$
Act	::=	Any | Allow$^+$
Allow	::=	(AVC$^+$)
AVC	::=	(id val)
Pol	::=	(Policy Ca T R*)
R	::=	(Rule T Effect)
Effect	::=	Permit | Deny

The elements of the syntax category R are called "Rules"; Ps, "PolicySets"; Pol, "Policies"; and T, "targets". The elements of syntax categories of Sub, Res, and Act are called the "subtargets". The elements of syntax categories of Ps, Pol, and R are called the "laws".

Example 4.1. *The following is an XACML policy:*

```
(Policy First-Applicable
        ((Any) (Any) (Any))
        (Rule (((role fac)) (Any) (Any)) Deny)
        (Rule ((Any) (Any) (Any)) Permit))
```

4.1.2 XACML request syntax

Q	::=	((S) (R) (A))
S	::=	AVP*
R	::=	AVP*
A	::=	AVP*
AVP	::=	(id val)

4.1.3 Parsing

To make the semantics more understandable, we assume the existence of a parser. Informally, we assume that is parser can take a string generated by the grammar and produce denotational object suggested by the strings form, mostly lists. Since the above syntax matches the list notation in Scheme, this parser is equivalent to the Scheme command quote.

$$\frac{S \Subset_S \text{Sub} \qquad R \Subset_R \text{Res} \qquad A \Subset_A \text{Act}}{(S, R, A) \Subset (\text{Sub}, \text{Res}, \text{Act})} \tag{4.1}$$

$$X \Subset \text{Any} \quad \forall X \in \{\text{S}, \text{R}, \text{A}\} \tag{4.2}$$

$$\frac{\exists i \,\text{s.t.}\, \text{S} \Subset \text{Allow}_i}{\text{S} \Subset_S (\text{Allow}_1, \text{Allow}_2, \ldots, \text{Allow}_n)} \tag{4.3}$$

$$\frac{\exists i \,\text{s.t.}\, \text{R} \Subset \text{Allow}_i}{\text{R} \Subset_R (\text{Allow}_1, \text{Allow}_2, \ldots, \text{Allow}_n)} \tag{4.4}$$

$$\frac{\exists i \,\text{s.t.}\, \text{A} \Subset \text{Allow}_i}{\text{A} \Subset_A (\text{Allow}_1, \text{Allow}_2, \ldots, \text{Allow}_n)} \tag{4.5}$$

$$\frac{\forall i \quad X \Subset_X \text{AVC}_i}{X \Subset_X (\text{AVC}_1, \text{AVC}_2, \ldots, \text{AVC}_n)} \text{ where } X \in \{S, R, A\} \tag{4.6}$$

$$\frac{\exists j \,\text{s.t.}\, \text{AVP}_j = \text{AVC}}{(\text{AVP}_1, \text{AVP}_2, \ldots, \text{AVP}_n) \Subset_X \text{AVC}} \text{ where } X \in \{S, R, A\} \tag{4.7}$$

Table 4.1: The Match Relationship

4.2 Semantics

Let P be the set of all policies (members of the syntactic category C of the language Spec). P includes both XACML Policies and PolicySets.[1] Let Q be the set of all requests (members of the syntactic category Q). Let D be the set of all decisions ($D = \{\mathsf{permit}, \mathsf{deny}, \mathsf{na}\}$). We will use the symbol used for the nonterminal in syntax in the natural semantics to refer to any element of the syntax category.

At the core of the semantics of XACML is the notation of a request matching the target of a rule, policy, or policy set. We will denote this relation by $q \Subset t$ where $q \in Q$ and t is a target. We will define $\Subset$ using a natural semantics given below in Table 4.1.

The semantics of the Margrave subset of XACML is given in a natural semantics that makes use of the relation $\Subset$. The results of evaluating a Rule is given in Table 4.2 in terms of the $\models_r$ relationship. The $\models_r$ relation is then used to define the $\models$ relation, which gives the semantics of our subset of XACML. The default behavior of $\models$ is given in Table 4.3. The behavior of $\models$ under each of the combining algorithms are given in Table 4.4 for permit-overrides, in Table 4.5 for deny-overrides, and in Table 4.6 for first-applicable.

[1] The word "policy" when uncapitalized will refer to access-control policies in general. When "policy" is capitalized, it will refer to the XACML tag Policy and the associated structure.

$$\frac{Q \notin T}{((\text{Rule} \quad T \quad \text{Effect}), Q) \models_r \mathsf{na}} \tag{4.8}$$

$$\frac{Q \notin T}{((\text{Rule} \quad T \quad \text{Permit}), Q) \models_r \mathsf{permit}} \tag{4.9}$$

$$\frac{Q \notin T}{((\text{Rule} \quad T \quad \text{Deny}), Q) \models_r \mathsf{deny}} \tag{4.10}$$

Table 4.2: The Rule Relationship $\models_r$

$$\langle((\text{Policy} \quad \text{Ca} \quad T), Q\rangle \models \mathsf{na} \tag{4.11}$$

$$\langle((\text{PolicySet} \quad \text{Ca} \quad T), Q\rangle \models \mathsf{na} \tag{4.12}$$

$$\frac{Q \notin T}{\langle((\text{Policy} \quad \text{Ca} \quad \text{T} \quad \text{R}^*), \text{Q}\rangle \models \mathsf{na}} \tag{4.13}$$

$$\frac{Q \notin T}{\langle((\text{PolicySet} \quad \text{Ca} \quad \text{T} \quad \text{C}^*), \text{Q}\rangle \models \mathsf{na}} \tag{4.14}$$

Table 4.3: Default na Judgments

$$\frac{Q \notin T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models_r \mathsf{permit}}{\langle((\text{Policy} \quad \text{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{permit}} \tag{4.15}$$

$$\frac{Q \notin T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models \mathsf{permit}}{\langle((\text{PolicySet} \quad \text{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{permit}} \tag{4.16}$$

$$\frac{Q \notin T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models_r \mathsf{deny} \qquad \forall j \neq i, \langle C_j, Q\rangle \not\models_r \mathsf{permit}}{\langle((\text{Policy} \quad \text{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{deny}} \tag{4.17}$$

$$\frac{Q \notin T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models \mathsf{deny} \qquad \forall j \neq i, \langle C_j, Q\rangle \not\models \mathsf{permit}}{\langle((\text{PolicySet} \quad \text{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{deny}} \tag{4.18}$$

$$\frac{Q \notin T \qquad \forall i, \langle C_i, Q\rangle \models_r \mathsf{na}}{\langle((\text{Policy} \quad \text{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{na}} \tag{4.19}$$

$$\frac{Q \notin T \qquad \forall i, \langle C_i, Q\rangle \models \mathsf{na}}{\langle((\text{PolicySet} \quad \text{Permit-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{na}} \tag{4.20}$$

Table 4.4: Permit-Overrides Judgments

$$\frac{Q \in T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models_r \mathsf{deny}}{\langle(\text{Policy} \quad \text{Deny-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{deny}} \tag{4.21}$$

$$\frac{Q \in T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models \mathsf{deny}}{\langle(\text{PolicySet} \quad \text{Deny-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{deny}} \tag{4.22}$$

$$\frac{Q \in T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models_r \mathsf{permit} \qquad \forall j \neq i, \langle C_j, Q\rangle \not\models_r \mathsf{deny}}{\langle(\text{Policy} \quad \text{Deny-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{permit}} \tag{4.23}$$

$$\frac{Q \in T \qquad \exists i \text{ s.t. } \langle C_i, Q\rangle \models \mathsf{permit} \qquad \forall j \neq i, \langle C_j, Q\rangle \not\models \mathsf{deny}}{\langle(\text{PolicySet} \quad \text{Deny-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{permit}} \tag{4.24}$$

$$\frac{Q \in T \qquad \forall i, \langle C_i, Q\rangle \models_r \mathsf{na}}{\langle(\text{Policy} \quad \text{Deny-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{na}} \tag{4.25}$$

$$\frac{Q \in T \qquad \forall i, \langle C_i, Q\rangle \models \mathsf{na}}{\langle(\text{PolicySet} \quad \text{Deny-Overrides} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{na}} \tag{4.26}$$

Table 4.5: Deny-Overrides Judgments

$$\frac{Q \in T \qquad \langle R_1, Q\rangle \models \mathsf{permit}}{\langle(\text{Policy} \quad \text{First-Applicable} \quad T \quad R_1, R_2, \ldots, R_n), Q\rangle \models \mathsf{permit}} \tag{4.27}$$

$$\frac{Q \in T \qquad \langle C_1, Q\rangle \models \mathsf{permit}}{\langle(\text{PolicySet} \quad \text{First-Applicable} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{permit}} \tag{4.28}$$

$$\frac{Q \in T \qquad \langle R_1, Q\rangle \models \mathsf{deny}}{\langle(\text{Policy} \quad \text{First-Applicable} \quad T \quad R_1, R_2, \ldots, R_n), Q\rangle \models \mathsf{deny}} \tag{4.29}$$

$$\frac{Q \in T \qquad \langle C_1, Q\rangle \models \mathsf{deny}}{\langle(\text{PolicySet} \quad \text{First-Applicable} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models \mathsf{deny}} \tag{4.30}$$

$$\frac{Q \in T \qquad \langle R_1, Q\rangle \models \mathsf{na} \qquad \langle(\text{Policy} \quad \text{First-Applicable} \quad T \quad R_2, \ldots, R_n), Q\rangle \models D}{\langle(\text{Policy} \quad \text{First-Applicable} \quad T \quad R_1, R_2, \ldots, R_n), Q\rangle \models D} \quad \forall D \in \mathcal{D} \tag{4.31}$$

$$\frac{Q \in T \qquad \langle C_1, Q\rangle \models \mathsf{na} \qquad \langle(\text{PolicySet} \quad \text{First-Applicable} \quad T \quad C_2, \ldots, C_n), Q\rangle \models D}{\langle(\text{PolicySet} \quad \text{First-Applicable} \quad T \quad C_1, C_2, \ldots, C_n), Q\rangle \models D} \quad \forall D \in \mathcal{D} \tag{4.32}$$

Table 4.6: First-Applicable Judgments

4.3 Analysis

The syntax and semantics of XACML defines $\mathcal{L}^{\mathsf{xacml}} = (Q, P, G, N, \langle\!\langle \cdot \rangle\!\rangle)$ where Q and P are given by the syntaxes, $G = \{\mathsf{permit}\}$, $N = \{\mathsf{na}, \mathsf{deny}\}$, and $\langle\!\langle \cdot \rangle\!\rangle$ is given by the semantics. The language has the policy connectives permit-overrides, deny-overrides, and first-applicable, and has no contextual information. The atomic policies are rules. XACML allows for explicit denials and the checking of the implicit absence of attributes.

Theorem 4.2. $\mathcal{L}^{\mathsf{xacml}}$ *is deterministic.*

Proof. The Semantics of atomic policies is given in Table 4.2. Inspection of the judgments clearly shows that only one of them can hold at a time. Thus, atomic policies are deterministic.

Semantics of the policy connective permit-overrides is given by Table 4.4 combined with Table 4.3. Note that the antecedents of all the judgments in these tables are disjoint, that is, for any policy and request at most one them can hold. Furthermore, at least one of them always holds. Thus, permit-overrides is deterministic. The same argument holds for deny-overrides and first-applicable using Tables 4.5 and 4.6. Thus, all the connectives are deterministic. □

Thus, an XACML policy may be viewed as a function from requests to decisions with $[\![\cdot]\!]$ in place of $\langle\!\langle \cdot \rangle\!\rangle$.

Theorem 4.3. $\mathcal{L}^{\mathsf{xacml}}$ *is homomorphic.*

Proof. By inspecting the judgments for permit-overrides, deny-overrides, and first-applicable clearly each judgment depends solely on the decisions yielded by the sub-policies under the same request. Thus, one can find needed functions from decision sequences to a decision to prove them homomorphic. These functions are:

permit-overrides

$$\mathsf{por}(d\ d^*) = \begin{cases} \mathsf{permit} & \text{if} \quad d = \mathsf{permit} \vee \mathsf{por}(d^*) = \mathsf{permit} \\ \mathsf{deny} & \text{else if} \quad d = \mathsf{deny} \vee \mathsf{por}(d^*) = \mathsf{deny} \\ \mathsf{na} & \text{else} \end{cases} \tag{4.33}$$

$$\mathsf{por}(\circ) = \mathsf{na} \tag{4.34}$$

deny-overrides

$$\mathsf{dor}(d\ d^*) = \begin{cases} \mathsf{deny} & \text{if} \quad d = \mathsf{deny} \vee \mathsf{dor}(d^*) = \mathsf{deny} \\ \mathsf{permit} & \text{else if} \quad d = \mathsf{permit} \vee \mathsf{dor}(d^*) = \mathsf{permit} \\ \mathsf{na} & \text{else} \end{cases} \tag{4.35}$$

$$\mathsf{dor}(\circ) = \mathsf{na} \tag{4.36}$$

first-applicable

$$\mathsf{fa}(d\ d^*) = \begin{cases} d & \text{if} \quad d = \mathsf{permit} \vee d = \mathsf{deny} \\ \mathsf{fa}(d^*) & \text{else} \end{cases} \tag{4.37}$$

$$\mathsf{fa}(\circ) = \mathsf{na} \tag{4.38}$$

where $\circ$ represents the empty sequence. □

Theorem 4.4. *$\mathcal{L}^{\mathsf{xacml}}$ is not continuous.*

Proof. Consider the policy p' in Example 4.1 and the policy p that would be p' without the first rule. Let the request q be `(((role fac)) () ())`. $[\![p]\!](q) = \mathsf{permit}$, but $[\![p']\!](q) = \mathsf{deny}$. Thus, adding a rule to p results in a request going from being granted to not being granted. □

Theorem 4.5. *Let $q \subseteq q'$ if q' contains every attribute-value pair that q contains.*
XACML is not safe with respect to $\subseteq$.

Proof. Consider the policy p shown in Example 4.1 and request $q =$ `(() () ())` and $q' =$ `(((role fac)) () ())`. Clearly $q \subseteq q'$. Yet

$$[\![p]\!](q) = \mathsf{permit} \not\leq \mathsf{deny} = [\![p]\!](q') \tag{4.39}$$

□

Chapter 5

Restrictions of First-Order Logic

Whereas XACML is an attempt to create a access-control language from scratch, other languages are modifications of first-order logic. A series of schemata for such languages is presented by J. Halpern and V. Weissman [7]. Here I present and analyze the the languages produced by the least restrictive of these schemata. Let the set of languages produced by this schema be called FOL. Languages in FOL are each a restriction of a many-sorted first-order logic with a different vocabulary (set of parameters including quantifier symbols, predicate symbols, constant symbols, and function symbols). Let $\mathsf{FOL}(\Phi)$ be the language in FOL with the vocabulary Φ. We assume that Φ includes the shorts S for subjects, R for resources, A for actions, and the predicate symbol $\mathsf{Permitted}$ of the sort $S \times R \times A \to \{\mathsf{T}, \mathsf{F}\}$. Furthermore, Φ may include sorts for environmental concerns such as the current time or location.

5.1 Syntax

An atomic policy in $\mathsf{FOL}(\Phi)$ is of one of the following forms:

$$(\forall x_1 \in X_1, x_2 \in X_2, \ldots x_m \in X_m, (f \implies \mathsf{Permitted}(s, r, a))) \tag{5.1}$$

$$(\forall x_1 \in X_1, x_2 \in X_2, \ldots x_m \in X_m, (f \implies \neg\mathsf{Permitted}(s, r, a))) \tag{5.2}$$

where each x_i is a variable over the sort X_i, s is a term over the sort S, r is a term over the sort R, a is a term over the sort A, and f is a first-order logic formal over Φ. A policy is either a atomic policy or an expression of the form

$$(\texttt{and}\ p^*) \tag{5.3}$$

where p^* is any number of policies.

Example 5.1. *Let Φ' be such that it contains*

1. *the sorts $S = \{\texttt{shriram}, \texttt{carl}, \texttt{steve}\}$, $R = \{\texttt{grades}, \texttt{courses}\}$, and $A = \{\texttt{assign}, \texttt{enroll}\}$;*

2. *and the predicates* $\mathsf{Permitted} : S \times R \times A \to \{\mathsf{T}, \mathsf{F}\}$, $\mathtt{faculty} : S \to \{\mathsf{T}, \mathsf{F}\}$, *and* $\mathtt{student} : S \to \{\mathsf{T}, \mathsf{F}\}$.

Then the following policy is in $\mathsf{FOL}(\Phi')$*:*

$$(\mathtt{and} \quad (\forall s \in S, \mathtt{faculty}(s) \implies \mathsf{Permitted}(s, \mathtt{grades}, \mathtt{assign})) \tag{5.4}$$

$$(\forall s \in S, \mathtt{student}(s) \implies \neg\mathsf{Permitted}(s, \mathtt{grades}, \mathtt{assign})) \tag{5.5}$$

$$(\forall s \in S, \neg\mathtt{faculty}(s) \implies \mathsf{Permitted}(s, \mathtt{courses}, \mathtt{enroll})) \quad) \tag{5.6}$$

Requests in $\mathsf{FOL}(\Phi)$ are of the form (s, r, a, e) where $s \in S$ is the subject making the request; $r \in R$, the requested resource; $a \in A$, the requested action to be preformed by the subject on the resource; and e is a first-order expression over Φ giving information about the environment.

Example 5.2.

$$(\mathtt{carl}, \mathtt{courses}, \mathtt{enroll}, \mathtt{student}(\mathtt{carl}) \wedge \mathtt{faculty}(\mathtt{shriram}) \wedge \neg\mathtt{student}(\mathtt{shriram})) \tag{5.7}$$

is a request in $\mathsf{FOL}(\Phi')$ *where* Φ' *is defined in Example 5.1.*

5.2 Semantics

Let $\mathcal{M}(p)$ for policies p be defined as

$$\mathcal{M}((\mathtt{and}\ p_1\ p_2\ \ldots\ p_n)) = \bigwedge_{i=1}^{n} \mathcal{M}(p_i) \tag{5.8}$$

$$\mathcal{M}((\mathtt{and}\ )) = \mathsf{T} \tag{5.9}$$

$$\mathcal{M}(p) = p \quad \text{where } p \text{ is an atomic policy} \tag{5.10}$$

p defines a relation $\langle\!\langle p \rangle\!\rangle$ between requests and $\{\mathsf{permit}, \mathsf{deny}\}$ as follows:

$$(s, r, a, e)\langle\!\langle p \rangle\!\rangle\,\mathsf{permit} \iff (\mathcal{M}(p) \wedge e) \vdash \mathsf{Permitted}(s, r, a) \tag{5.11}$$

$$(s, r, a, e)\langle\!\langle p \rangle\!\rangle\,\mathsf{deny} \iff (\mathcal{M}(p) \wedge e) \vdash \neg\mathsf{Permitted}(s, r, a) \tag{5.12}$$

where $\vdash$ is the proves relation for many-sorted first-order logic over Φ.

However, to define a deterministic version of $\langle\!\langle \cdot \rangle\!\rangle$, we will let the set of decisions be expended to $D = \{\mathsf{na}, \mathsf{permit}, \mathsf{deny}, \mathsf{error}\}$ and define

$$[\![p]\!] = \lambda(s, r, a, e)\,.\begin{cases} \mathsf{error} & \text{if} & (\mathcal{M}(p) \wedge e) \vdash \bot, \\ \mathsf{permit} & \text{else if} & (\mathcal{M}(p) \wedge e) \vdash \mathsf{Permitted}(s, r, a), \\ \mathsf{deny} & \text{else if} & (\mathcal{M}(p) \wedge e) \vdash \neg\mathsf{Permitted}(s, r, a), \\ \mathsf{na} & \text{else.} & \end{cases} \tag{5.13}$$

The first case takes care of when $(s, r, a, e)\langle\!\langle p \rangle\!\rangle\,\mathsf{permit}$ and $(s, r, a, e)\langle\!\langle p \rangle\!\rangle\,\mathsf{deny}$, and the last case takes care of when $\langle\!\langle p \rangle\!\rangle$ relates (s, r, a, e) to neither permit or deny.

5.3 Analysis

The language $\mathsf{FOL}(\Phi)$ for a vocabulary Φ defines the access-control language $(Q, P, G, N, [\![\cdot]\!])$ where Q and P are given by the syntax above, $G = \{\mathsf{permit}\}$, $N = \{\mathsf{na}, \mathsf{deny}, \mathsf{error}\}$, and $[\![\cdot]\!]$ is defined above. Since $[\![\cdot]\!]$ was defined to be deterministic, $\mathsf{FOL}(\Phi)$ is deterministic. From the syntax, the languages of FOL has only policy connective, `and`. No contextual information is used by $[\![\cdot]\!]$. FOL allows for explicit denials and the checking of the explicit absence of attributes.

Theorem 5.3. *For some values of* Φ*,* $\mathsf{FOL}(\Phi)$ *is not homomorphic.*

Proof. Let Φ' be as in the Example 5.1. Assume that `and` is homomorphic for $\mathsf{FOL}(\Phi')$. Since $[\![\cdot]\!]$ does not make use of contextual information, then there exists $\sqcap : D^* \to D$ such that

$$\forall p_1, p_2, \ldots, p_n \in P, \forall q \in Q, [\![(\texttt{and } p_1\ p_2\ \ldots\ p_n)]\!](q) = \sqcap([\![p_1]\!](q), [\![p_2]\!](q), \ldots, [\![p_n]\!](q)) \tag{5.14}$$

Consider the policy p in Example 5.1. Let p_1 be the first sub-policy in p, p_2, the second, and p_3, the third. Let q be the request found in Example 5.2. Note that

$$[\![p_1]\!](q) = \mathsf{na} \tag{5.15}$$
$$[\![p_2]\!](q) = \mathsf{na} \tag{5.16}$$
$$[\![p_3]\!](q) = \mathsf{na} \tag{5.17}$$
$$[\![p]\!](q) = \mathsf{permit}. \tag{5.18}$$

where the last line holds by the same reasoning as used in the third interpretation of $\wp$ in Chapter 2. Thus, we must conclude that $\sqcap(\mathsf{na}, \mathsf{na}, \mathsf{na}) = \mathsf{permit}$.

Now consider the request $q' = (\texttt{steve}, \texttt{courses}, \texttt{enroll}, \texttt{student(carl)} \wedge \texttt{faculty(shriram)} \wedge \neg\texttt{student(shriram)})$. Now

$$[\![p_1]\!](q') = \mathsf{na} \tag{5.19}$$
$$[\![p_2]\!](q') = \mathsf{na} \tag{5.20}$$
$$[\![p_3]\!](q') = \mathsf{na} \tag{5.21}$$
$$[\![p]\!](q') = \mathsf{na}. \tag{5.22}$$

Thus, we must conclude that $\sqcap(\mathsf{na}, \mathsf{na}, \mathsf{na}) = \mathsf{na}$. However, this is contradiction since $\sqcap(\mathsf{na}, \mathsf{na}, \mathsf{na}) = \mathsf{na} \neq \mathsf{permit} = \sqcap(\mathsf{na}, \mathsf{na}, \mathsf{na})$. Thus, no such $\sqcap$ may exist and `and` is not homomorphic. □

Given two requests, $q = (s, r, a, e)$ and $q' = (s', r', a', e')$, if $s \neq s'$, $r \neq r'$, $a \neq a'$ we consider the two requests incomparable. If $s = s'$, $r = r'$, and $a = a'$, then we would like $\subseteq$ to order requests containing more information as higher than those with less information. Thus, one might conclude that $q \subseteq q'$ if $e' \implies e$. However, suppose $e' \implies \bot$. Then e' contains no information and yet it implies e. Similarly, if $\mathcal{M}(p) \wedge e' \implies \bot$, then e' contains no information with respect to p. Thus, we define $\subseteq_p$ as follows:
Let $(s, r, a, e) \subseteq_p (s', r', a', e')$ iff

1. $s = s'$, $r = r'$, and $a = a'$; and

2. $\mathcal{M}(p) \wedge e'$ implies $\mathcal{M}(p) \wedge e$ but not $\bot$, or $\mathcal{M}(p) \wedge e$ implies $\bot$.

Theorem 5.4. *For all vocabularies Φ, all $p \in P$ for $\mathsf{FOL}(\Phi) = (Q, P, G, N, [\![\cdot]\!])$ are safe with respect to $\subseteq_p$.*

Proof. Assume $p \in P$ is not safe. Then there must exist $q, q' \in Q$ such that $q \subseteq_p q'$ and $[\![p]\!](q) \not\leq [\![p]\!](q')$.

Since $G = \{\mathsf{permit}\}$, $[\![p]\!](q) = \mathsf{permit}$ and $(\mathcal{M}(p) \wedge e) \vdash \mathsf{Permitted}(s, r, a)$. Furthermore, since $N = \{\mathsf{na}, \mathsf{deny}, \mathsf{error}\}$, $[\![p]\!](q')$ must be either na, deny, or error.

Since $q \subseteq_p q'$, two cases arise:

1. $\mathcal{M}(p) \wedge e'$ implies $\mathcal{M}(p) \wedge e$ but not $\bot$. Since $(\mathcal{M}(p) \wedge e) \vdash \mathsf{Permitted}(s, r, a)$ and $e' \implies e$, $(\mathcal{M}(p) \wedge e') \vdash \mathsf{Permitted}(s, r, a)$. Thus, $[\![p]\!](q')$ is either permit or error. However, if $[\![p]\!](q') = \mathsf{error}$, then $\mathcal{M}(p) \wedge e' \implies \bot$, a contradiction. Furthermore, $[\![p]\!](q') = \mathsf{permit}$ is also a contradiction.

2. $\mathcal{M}(p) \wedge e$ implies $\bot$. In this case, $[\![p]\!](q) = \mathsf{error}$, a contradiction.

Under both cases, contradictions are reached, and thus, P must be safe w.r.t. $\subseteq_p$.

□

Chapter 6

Related Work

De Capitani di Vimercati *et al.* discuses the language construct of explicit denial and how it introduces the need for policy connectives that reduce the clarity of the language [3]. The authors list various policy connectives that are possible, many of which are more complex than those I present. The paper includes discussion a few access-control languages including XACML and a language grounded in first-order logic. The paper does not, however, attempt to systemically compare them.

The work of Mark Evered and Serge Bögeholz concerns the quality of an access-control language [4]. After conducting a case study of the access-control requirements of a Health Information System, they propose a list criteria for an access-control languages. They state that languages should be concise, clear, aspect-oriented (*i.e.*, separate from the application code), fundamental (*i.e.*, integrated with the middleware, not an *ad hoc* addition), positive (*i.e.*, lists what is allowed, not what is prohibited), supportive of needs-to-know, and efficient. Although they compare four languages based on these criteria, they do not formalize the criteria.

J. Malcolm examines the clarity of command languages (languages for interacting with a shell, or scripting languages) [8]. Primarily, he uses the number of lines and the number of tokens in the code to preform a benchmarking task to compare languages. However, he does comment on the desirable ability to break a program into discrete parts as opposed to having to "understand the whole at once" (p56). Although he does not formalize this notion, it captures much the same intuition behind the homomorphic property.

Chapter 7

Conclusion

Chapter 2 delineated various language constructs and features found in access-control languages. Chapter 3 presented properties that access-control languages can have that are relevant to the clarity of the language. Chapters 4 and 5 tested preceding framework. The two languages (or more correctly, the one language and one language schema) differ as follows under the framework:

Decision Set Both had permit, deny, and not applicable, but only FOL had error.

Policy Connectives XACML provided three connectives: permit-overrides, deny-overrides, and first-applicable; whereas, FOL only provided one: `and`.

Checking for the Absence of Attributes XACML employed implicit checking while FOL supported explicit checking.

Homomorphism XACML was homomorphic and allowed for reasoning about a policy by reasoning about the sub-policies separately; whereas, FOL was not homomorphic and requires reasoning about the whole policy at once.

Safety FOL provided safety for the most natural definition of the "contains more information" ordering; whereas XACML did not, which implies that missing information from an XACML request could result in unintended access being granted.

These differences are not orthogonal. Clearly the connectives selected determines if the language will have the homomorphic property. Furthermore, implicit checking of attributes will result in the lose of safety.

As noted in Chapter 4, the current framework for the comparison of clarity among languages must be generalized to treat language with more exotic constructs like obligations. Furthermore, user studies should be conconducted to confirm that the properties listed in Chapter 3 are indeed relevent to langauge clarity. Lastly, this framework must be coupled with one for measuring the expressive power of an access-control language before fair judgment may be passed on languages.

Bibliography

[1] Anne Anderson. Core and hierarchical role based access control (RBAC) profile of XACML, version 2.0. Technical report, OASIS, September 2004.

[2] Moritz Y. Becker and Peter Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *CSFW '04: Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, page 139, Washington, DC, USA, 2004. IEEE Computer Society.

[3] Sabrina De Capitani di Vimercati, Pierangela Samarati, and Sushil Jajodia. Policies, models, and languages for access control. In Subhash Bhalla, editor, *Databases in Networked Information Systems: 4th International Workshop*, volume 3433 of *Lecture Notes in Computer Science*. Springer-Verlag GmbH, March 2005.

[4] Mark Evered and Serge Bögeholz. A case study in access control requirements for a health information system. In *CRPIT '32: Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 53–61, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[5] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change impact-analysis of access-control policies. In *27th International Conference on Software Engineering ICSE '05*, St. Louis, Missouri, May 2005. To appear.

[6] Michael Matthew Greenberg, Casey Marks, Leo Alexander Meyerovich, and Michael Carl Tschantz. The soundness and completeness of margrave with respect to a subset of xacml. Technical Report CS-05-05, Brown University, April 2005.

[7] J. Halpern and Vicky Weissman. Using first-order logic to reason about policies. In *16th IEEE Computer Security Foundations Workshop (CSFW '03)*, pages 187–201, 2003.

[8] James A. Malcolm. Brevity and clarity in command languages. *SIGPLAN Not.*, 16(10):53–59, 1981.

[9] T. Moses. eXtensible Access Control Markup Language (XACML) version 1.0. Technical report, OASIS, February 2003.

[10] Calvin Powers and Matthias Schunter. Enterprise privacy authorization language (EPAL 1.2). W3C Member Submission, November 2003.

[11] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.