Using First-Order Logic to Reason about Policies*

Joseph Y. Halpern
Cornell University
Ithaca, NY 14853
halpern@cs.cornell.edu

Vicky Weissman
Cornell University
Ithaca, NY 14853
vickyw@cs.cornell.edu

Abstract

A policy describes the conditions under which an action is permitted or forbidden. We show that a fragment of (multi-sorted) first-order logic can be used to represent and reason about policies. Because we use first-order logic, policies have a clear syntax and semantics. We show that further restricting the fragment results in a language that is still quite expressive yet is also tractable. More precisely, questions about entailment, such as 'May Alice access the file?', can be answered in time that is a low-order polynomial (indeed, almost linear in some cases), as can questions about the consistency of policy sets. We also give a brief overview of a prototype that we have built whose reasoning engine is based on the logic and whose interface is designed for non-logicians, allowing them to enter both policies and background information, such as 'Alice is a student', and to ask questions about the policies.

1 Introduction

A policy describes the conditions under which an action, such as reading a file, is permitted or forbidden. Digital content providers have a rough idea of what their policies should be. Unfortunately, policies are typically described informally. As a result, their meaning and consequences are not always clear.

To better understand the problem, consider the statement 'only librarians may edit the on-line catalog'. We can view this statement as a policy, because it governs who may edit the catalog, based on whether or not the editor is a librarian. It is not clear if this policy permits librarians to make changes to the catalog or only forbids anyone who is not a librarian from doing so. The policy could be rewritten to remove this particular ambiguity, but others are likely to exist if policies are written in a natural language. Policy languages such as the Extensible rights Markup Language (XrML) [10] and Open Digital Rights Language (ODRL) [21] have the potential to be more formal (partly because their syntax is more restricted). Currently, however, the only semantics for these languages seems to be an English description of what the syntax means; thus, they also suffer from significant ambiguity. Our goal in this paper is to provide a logic with a clear syntax and semantics that can be used to represent and reason about policies. In addition, we want the logic to be well-suited to the needs of digital content providers. To achieve our objectives, we use a fragment of first-order logic. This automatically gives us a clear syntax and semantics; thus, it remains to argue that the logic is well-suited to the needs of digital content providers.

To be of practical use, a logic must satisfy (at least) the following three desiderata.

1. It must be expressive enough to capture in an easy and natural way the policies that people want to discuss.

2. It must be tractable enough to allow interesting queries about policies to be answered efficiently.

3. It must be usable by non-logicians, because we cannot expect policy makers and administrators to be well-versed in logic.

Of course, whether a logic is sufficiently expressive to meet our first objective depends very much on the application. To evaluate our approach, we gathered a large collection of policies from various libraries, including on-line collections, local and university libraries, the Library of Congress, and Cornell's Digital Library Research Group. We have written these policies in our language. In addition, we have begun to encode government policies in our language, including those that determine a person's eligibility for Social Security. Finally, we have created a translation from most of the XrML Core and all of the XrML Content Extension to our language. Details of the translation and a

*Authors supported in part by NSF under grant CTC-0208535, by ONR under grants N00014-00-1-03-41 and N00014-01-10-511, by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grant N00014-01-1-0795, and by AFOSR under grant F49620-02-1-0101.

more complete discussion of the collected policies are given in a companion paper [19].

For the second desideratum, we focus on two key queries:

- Given a set of policies and an *environment* that provides all relevant facts (e.g., ‘Alice is a librarian’, ‘Anyone who is a librarian for less than a year is a novice’, etc.), does it follow that a particular action, such as Alice editing the on-line catalog, is permitted or forbidden?

- Is a set of policies consistent? In other words, are there no actions that are both permitted and forbidden by the policies in the set? This question is particularly interesting for collaboration. For example, suppose that Alice is writing the policies for her university’s new outreach program. If the union of her policies and the university policies is consistent, then she knows that her policies do not contradict those of the university.

The answers to these questions could be used by enforcement mechanisms and individuals who want to do regulated activities. More importantly, we believe that the answers provide a reasonably good understanding of the policies, increasing our confidence that the formal statements capture the informal rules and the informal rules capture the policy creator’s intent.

To address our third goal, the usability requirement, we developed, and are currently refining and extending, a prototype that allows users to enter policies, as well as facts about their environment, and to ask questions about them. This software will be tested by University of Virginia librarians as part of the Mellon-Fedora project [32] to verify that the language can be used by people who have not been trained in logic.

There have been a number of attempts to give formal semantics to policies, some of which involve first-order logic. Most of the first-order approaches are based on some variant of Datalog [16]. By beginning with Datalog, these solutions start with a language that is tractable, but not sufficiently expressive. They then extend the language to better meet the needs of applications. In particular, they find extensions that permit a limited use of negation and functions. The restrictions that we make are quite different from those made previously. We believe (and will argue throughout this paper) that the resulting language is especially well-suited for many applications, and has a number of advantages over variants of Datalog.

The rest of this paper is organized as follows. In the next section, we formally define our notions of a policy and an environment. We also give examples that illustrate how policies can be represented in an appropriate fragment of first-order logic. In Section 3 we show that, in general, the questions we want to ask about policies are hard to answer. In Section 4 we present some restrictions under which these questions are tractable. We give a brief overview of the prototype that we are building in Section 5. We discuss the Datalog approaches, as well as other related work, in Section 6. The paper concludes in Section 7 with plans for future research. Detailed proofs are left to the full paper.

2 A First-Order Logic for Reasoning About Policies

For the rest of the paper, we assume knowledge of many-sorted first-order logic at the level of Enderton [14]. More specifically, we assume the reader is familiar with the syntax of first-order logic, including constants, variables, predicate symbols, function symbols, and quantification, with the semantics of first-order logic, including relational models and valuations, and with the notions of satisfiability and validity of first-order formulas.

We use many-sorted first-order logic with equality over some vocabulary Φ to express and reason about policies. Let $\mathcal{L}^{fo}(\Phi)$ denote the set of first-order formulas over the vocabulary Φ. For this paper, we assume that there are at least three sorts, $Actions$ (e.g., accessing a file), $Subjects$ (the agents that perform actions; these are sometimes called *principals* in the literature), and $Times$. While these sorts seem natural for any policy logic, other sorts may be desired for particular applications. These sorts, including objects and roles, may be added to the logic without affecting our results.

The vocabulary Φ is application dependent; however, we assume that Φ contains a binary predicate **Permitted** on $Subjects \times Actions$ and a constant **now** of sort $Times$. **Permitted**(t, t') means that subject t is allowed to perform action t'. In practice, it may be useful to add additional arguments to **Permitted**, such as when the action is permitted and who is authorizing the granting or revoking of the permission. We have not included these here to simplify the exposition; including them would not change our results. The constant **now** denotes the current time. In practice, a global clock would determine the interpretation of **now**.

A *policy* is a closed first-order formula of the form

$$\forall x_1 \ldots \forall x_m (f \Rightarrow (\neg)\textbf{Permitted}(t, t')),$$

where f is any first-order formula, t and t' are terms of sort Subject and Action respectively, and the notation $(\neg)$**Permitted** indicates that the **Permitted** predicate may or may not be negated. Defining the policy in this way provides a structure that matches our intuition, namely, that a policy is a set of conditions under which an action is or is not permitted.

To illustrate how policies can be expressed in first-order logic, consider the following examples.

Example 2.1: The policy 'only librarians may edit the catalog' can be characterized by the following two policies

$$\forall x(\neg\textbf{Librarian}(x) \Rightarrow \neg\textbf{Permitted}(x, \text{edit the catalog}))$$
$$\forall x(\textbf{Librarian}(x) \Rightarrow \textbf{Permitted}(x, \text{edit the catalog})).$$

(Depending on the intended meaning of the English statement, the first formula by itself may characterize the policy.) $\blacksquare$

Example 2.2: The policy 'a customer may download any article if she has paid a fee within the past six weeks' can be rewritten as 'if an individual i has paid the fee within the past six weeks, i is a customer, and a is some article, then i may download a'. The policy can be encoded readily in the logic as

$$\forall i \forall t \forall a((\textbf{PaidFee}(i, t) \wedge (\textbf{now} - 6 < t < \textbf{now}) \wedge \textbf{Customer}(i, \textbf{now}) \wedge \textbf{Article}(a)) \Rightarrow \textbf{Permitted}(i, \text{download}(a))).$$

$\blacksquare$

Example 2.3: The policy set 'anyone may sing' and 'anyone who is allowed to sing may dance' can be characterized by the following two formulas:

$$\forall x(\textbf{Permitted}(x, \text{sing}))$$
$$\forall x(\textbf{Permitted}(x, \text{sing}) \Rightarrow \textbf{Permitted}(x, \text{dance})).$$

$\blacksquare$

To determine the consequences of a policy, we need to know what facts are true in the context in which the policies are applied. For example, to decide if the policies in Example 2.1 permit Alice to edit the catalog, we must know if Alice is a librarian. In other words, we must know if the statement **Librarian**(Alice) is true. This fact, along with all the others that are needed to analyze a set of policies, are contained in the *environment*. The *environment* may include very simple statements such as '*The Cat in the Hat* is a children's book' or 'Sally has a junior library card'. More complex statements may also be included, such as the conditions under which a customer is considered to be in good standing and 'at all times, there is a senior staff member who is on call'. All the examples we have considered so far confirm our belief that first-order logic is sufficiently expressive to capture most environments that are likely to arise in practice. Thus, we formally define an environment to be a closed first-order formula that does not contain the **Permitted** predicate. The requirement that the environment not contain **Permitted** encourages the intuitive separation between the environment, which is a description of reality, and the policies, which are the rules governing that reality.

The two types of queries discussed in the introduction can now be formalized. The first query, is an individual t permitted to perform an action t' (where t and t' are closed terms) given an environment E and some policies $p_1, \ldots, p_n$, amounts to asking if the formula $E \wedge p_1 \wedge \ldots \wedge p_n \Rightarrow \textbf{Permitted}(t, t')$ is valid. (Similarly, t is forbidden to do t' if and only if $E \wedge p_1 \wedge \ldots \wedge p_n \Rightarrow \neg\textbf{Permitted}(t, t')$ is valid.) The second query, 'Are the policies consistent?', asks if the formula $E \wedge p_1 \wedge \ldots \wedge p_n$ is satisfiable. For ease of exposition, we focus on determining if an action is permitted (or forbidden). As we show, it is easy to modify our techniques to handle the consistency question.

3 Intractability Results

In general, the queries in which we are interested cannot be answered efficiently. Indeed, the problem in its full generality is easily seen to be undecidable if the vocabulary Φ has at least one binary predicate other than **Permitted** (and closed terms t and t' of sort $Subjects$ and $Actions$, respectively, so that it is possible to actually form queries). To see this, let f be an arbitrary formula that does not contain **Permitted**. Consider the policy $f \Rightarrow \textbf{Permitted}(t, t')$, and let the environment be empty (i.e., **true**). Standard manipulations show that

$$(f \Rightarrow \textbf{Permitted}(t, t')) \Rightarrow \textbf{Permitted}(t, t')$$

is equivalent to

$$f \vee \textbf{Permitted}(t, t').$$

Since f does not mention **Permitted**, the last formula is valid iff f is valid. The validity problem for first-order formulas is well-known to be undecidable, even if we restrict to formulas that contain a single binary predicate; indeed, undecidability holds even if we further restrict to formulas of the form $\exists x \exists y \forall z f'$, where f' is quantifier-free [7]. This means that we cannot determine if a single policy implies a permission when the conditions under which the policy applies must be written in first-order logic as a formula of the form $\exists x \exists y \forall z f'$ where f' has a binary predicate other than **Permitted**.

We can get the same result even without assuming that Φ has a binary predicate other than **Permitted**. This is summarized in the following theorem.

Theorem 3.1: *Let $\mathcal{L}_0$ be the set of closed function-free formulas of the form $(f \Rightarrow \textbf{Permitted}(c, c')) \Rightarrow \textbf{Permitted}(c, c')$, where c and c' are constants of the appropriate sorts, $\exists x \exists y \forall z f'$, and f' is a quantifier-free formula whose only nonlogical symbol is **Permitted**. The validity question for $\mathcal{L}_0$ is undecidable.*

It follows from Theorem 3.1 that we cannot determine if a set of policies imply a permission in an environment when the environment is empty, the policy set has only one policy, and that policy has a single alternation of quantifiers and no function symbols. Not surprisingly, similar undecidability results hold if we allow formulas in the environment to involve nontrivial quantification (provided that there is a binary predicate in the language other than **Permitted**, since we do not allow **Permitted** in the environment). Given Theorem 3.1, it seems that our only hope is to forbid any alternation of quantifiers.

How much quantification do we really need? A quantifier-free environment suffices to capture simple databases. However, we want to allow at least universal formulas in the environment so that we can state general properties, such as 'all freshman are students'. Universal quantification is even more critical in policies. If we do not allow a policy to have any quantification (i.e., define a policy to have the form $f \Rightarrow \textbf{Permitted}(t, t')$ where t and t' are closed terms and f is quantifier-free), then each policy must govern a specific individual and action. For example, we can say 'If Alice is good, she may play outside', but we cannot say 'All good children may play outside'. Because policies typically permit an individual to do an action based on the attributes of that individual and action, we must allow policies to be universally quantified.

Policies with universal quantification (and a quantifier-free antecedent) are sufficiently expressive to capture the policies that we have collected from libraries and government databases. Although some of the collected policies appear to need existential quantification, they can be converted to formulas with universal quantification.

Example 3.2: Consider the policy 'anyone who is accompanied by a librarian may enter the stacks'. A natural way to state this in first-order logic is

$$\forall x_1(\exists x_2(\textbf{Librarian}(x_2) \land \textbf{Accompanies}(x_2, x_1)) \Rightarrow \textbf{Permitted}(x_1, \text{enter(stacks)})).$$

This formula is logically equivalent to

$$\forall x_1 \forall x_2((\textbf{Librarian}(x_2) \land \textbf{Accompanies}(x_2, x_1)) \Rightarrow \textbf{Permitted}(x_1, \text{enter(stacks)})),$$

which uses only universal quantification. ∎

Note that enter is a function in Example 3.2. Unfortunately, it is well known that the validity problem for existential formulas with functions is undecidable [7]. The following result is almost immediate:

Theorem 3.3: *Let $\mathcal{L}_1$ be the set of closed formulas of the form $\forall x_1 \ldots x_m(f \Rightarrow \textbf{Permitted}(t, t')) \Rightarrow \textbf{Permitted}(t, t')$, where t and t' are terms of the appropriate sort, and f is a quantifier-free formula (possibly containing function symbols). The validity problem for $\mathcal{L}_1$ is undecidable.*

Theorem 3.3 suggests that even if we drastically reduce quantification, we still need to disallow functions to get decidability. Once we restrict quantification to a bare minimum and remove functions entirely, then we do get a decidable fragment, but it's not tractable. Recall that Π_2^P is the second level of the polynomial hierarchy, and represents languages that can be decided in co-NP with an NP oracle.

Theorem 3.4: *Let Φ be a vocabulary that contains* ***Permitted****, constants c and c' of sorts $Subjects$ and $Actions$, respectively, and possibly other predicate and constant symbols. Assume there is a bound on the arity of the predicate symbols in Φ (that is, there exists some N such that all predicate symbols in Φ have arity at most N). Finally, let $\mathcal{L}_2$ be the set of all closed formulas in $\mathcal{L}^{fo}(\Phi)$ of the form $E \land p_1 \land \ldots \land p_n \Rightarrow \textbf{Permitted}(c, c')$ such that E is a conjunction of quantifier-free and universal formulas and each policy $p_1, \ldots, p_n$ has the form $\forall x_1 \ldots \forall x_m(f \Rightarrow \textbf{Permitted}(t_1, t_2))$ where t_1 and t_2 are terms of the appropriate sort and f is quantifier-free.*

(a) The validity problem for $\mathcal{L}_2$ is in Π_2^P.

(b) If $\mathcal{L}_3$ is the set of formulas in $\mathcal{L}_2$ in which every policy's antecedent is a conjunction of literals, then the validity problem for $\mathcal{L}_3$ is Π_2^P hard.

(c) If $\mathcal{L}_4$ is the set of $\mathcal{L}_2$ formulas in which E is quantifier-free, then the validity problem for $\mathcal{L}_4$ is NP-hard.

We remark that if we do not require the arity of the predicate symbols in Φ to be bounded, then we must replace Π_2^P by co-NEXPTIME (co-nondeterministic exponential time) in parts (a) and (b) [7].

Theorems 3.1, 3.3, and 3.4 seem to suggest that the questions we are interested in are hopelessly intractable. Fortunately, things are not nearly as bad as they seem.

4 Identifying Tractable Sublanguages

The work on Datalog and its variants mentioned in the introduction demonstrates that there are useful, tractable fragments of first-order logic. In this section we identify a different set of restrictions than those considered by the Datalog community, show that they lead to tractability, and argue that they are particularly well-suited to reasoning about policies.

4.1 Analyzing a restricted set of policies

Define a *standard policy* to be a policy of the form $\forall x_1 \ldots \forall x_n((\ell_1 \land \ldots \land \ell_k) \Rightarrow (\neg)\textbf{Permitted}(t_1, t_2))$ where $\ell_1, \ldots, \ell_k$ are literals and both t_1 and t_2 are terms of the appropriate sort. A *basic environment* is an environment that is a conjunction of ground literals. Basic environments are sufficiently expressive to capture the information in databases and certificates. While this is adequate for many applications, basic environments cannot represent general properties, such as 'all freshmen are students'. To handle these, we define a *standard environment* to be an environment that is a conjunction of quantifier-free formulas and universal formulas of the form $\forall x_1 \ldots \forall x_n(\ell_1 \wedge \ldots \wedge \ell_k \Rightarrow \ell_{k+1})$, where $\ell_1, \ldots, \ell_{k+1}$ are literals. As argued in Section 3, standard policies seem sufficiently expressive to capture most (if not all) policies of interest. Basic environments suffice for many applications of interest; standard environments suffice for all the applications we have considered.

As a first step towards tractability, we consider only basic environments and make what may seem to be rather arbitrary restrictions on policies. (Later in this section we justify the restrictions and discuss standard environments.) One of the restrictions relies on a notion called *bipolarity*, which in turn relies on a well-known technique from theorem proving called *unification* [30].

Two literals ℓ and ℓ' are *unifiable* if there are variable substitutions σ and σ' such that $\ell\sigma = \ell'\sigma'$. For example, $P(x, c_1)$ and $P(c_2, y)$ are unifiable by substituting c_2 for x and c_2 for y, while $P(x, c_1)$ and $P(y, c_2)$ are not unifiable (assuming that c_1 and c_2 are distinct constants). A literal ℓ is *bipolar in formula f*, written in CNF[1], if ℓ is in f and there is another literal ℓ' in f such that ℓ and $\neg\ell'$ are unifiable. The pair ℓ, ℓ' is called a *bipolar pair*. For example, $\textbf{Permitted}(x, \text{nap})$ and $\textbf{Permitted}(\textbf{Advisor}(x), \text{nap})$ are the only bipolar literals in the formula $\forall x(\textbf{Permitted}(x, \text{play}) \wedge \textbf{Permitted}(x, \text{nap}) \Rightarrow \textbf{Permitted}(\textbf{Advisor}(x), \text{nap}))$.

Theorem 4.1: *Let Φ be a vocabulary that contains* ***Permitted*** *(and possibly other predicate, constant, and function symbols). Let $\mathcal{L}_5$ consist of all closed formulas in $\mathcal{L}^{fo}(\Phi)$ of the form $E \wedge P \Rightarrow$ **Permitted**(t, t'), where P is a conjunction of standard policies and both t and t' are closed terms of the appropriate sort, such that*

(a) E is a basic environment,

(b) equality is not used in E or P,

(c) if a variable appears in a policy p in P, then it appears as an argument to ***Permitted*** *in p, and*

(d) there are no bipolars in P.

We can determine the validity of formulas in $\mathcal{L}_5$ in time $O((|E| + |P|) \log |E|)$, where $|\varphi|$ denotes the length of φ, when viewed as a string of symbols.

[1]We say that a first-order formula is in CNF if it has the form $Q_1x_1 \ldots Q_kx_k(\varphi_1 \wedge \ldots \wedge \varphi_n)$, where each φ_i is a (quantifier-free) disjunction of literals and $Q_j \in \{\forall, \exists\}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, k$. Each φ_i is called a *clause*. We sometimes identify a universal formula in CNF with its set of clauses.

Note that the language $\mathcal{L}_5$ includes formulas such as

$$\begin{aligned}&\textbf{Student}(\text{Alice}) \wedge \textbf{Good}(\text{Alice}) \wedge \\ &\forall x(\textbf{Student}(x) \Rightarrow \textbf{Permitted}(x, \text{work})) \wedge \\ &\forall x(\textbf{Student}(x) \wedge \textbf{Good}(x) \Rightarrow \textbf{Permitted}(x, \text{play})).\end{aligned}$$

('Alice is a student, Alice is good, all students may work and all good students may play'). Unlike Theorem 3.4(c), function symbols are allowed in Theorem 4.1. Moreover, there is no assumption that the arity of predicates and functions in Φ is bounded. The price we pay for this added generality and for cutting the complexity to linear in the number of policies (which could well be large) and not much more than linear in the size of the database (which we expect to be relatively small, particularly in certificate-passing systems) is the four restrictions. Before describing the proof of Theorem 4.1, we argue that the restrictions are often met in practice and show how the restrictions can be relaxed so that the result is even more applicable.

As we have already said, basic environments are sufficiently expressive to capture the facts stored in databases and certificates. This is not always enough. For example, the documents that describe who may collect Social Security define an aged person to be anyone 65 years old or older, who is a resident of the U.S., and is either a citizen or an alien residing in the U.S. both legally and permanently. A basic environment cannot capture what it means to be aged, according to Social Security policies. Nevertheless, basic environments seem perfectly adequate for certificate-based permissions in the spirit of SPKI/SDSI [12, 13] and for licenses as described by XrML [10], which assumes a minimal environment containing facts such as the current time and the time of the most recent revocation polling.

The second restriction, that equality is not used, is a serious restriction. Without equality, we cannot express threshold policies ('if at least three different people vouch for Alice, then she can enter the club') nor can we express the identity of two individuals ('Miss Alice Smith = Mrs. Alice Jones'). Nevertheless, there are large classes of policies that do not require equality at all. (This includes the policies in the Social Security database and the library policies that we have considered.)

The third restriction, that every variable appearing in a policy p also appears as an argument to **Permitted** in p, is met if an individual is granted or denied permission based solely on her attributes and the attributes of the regulated action. Notice that the policies in Examples 2.1 and 2.3 have this form, but the policies in Examples 2.2 and 3.2 do not. In particular, whether the policy in Example 3.2 allows x_1 to enter the stacks depends on an attribute of some other person x_2. As we shall see, we can allow variables to appear in policies without appearing as arguments to **Permitted**, as long as the number of such variables in any one policy is small.

The last restriction, that there are no bipolar literals in $p_1 \wedge \ldots \wedge p_n$, is likely to be met if all the policies are *permitting policies* (that is, their conclusions have the form **Permitted**(t_1, t_2)) or all are *denying policies* (that is, their conclusions have the form $\neg$**Permitted**(t_1, t_2)), and policies do not have **Permitted** in their antecedents. To see why, recall that a permitting policy says 'if the following conditions hold, then a particular action is permitted'. These conditions typically include requirements that someone possess one or more credentials, such as a library card or a driver's license. It is fairly rare that *not* having a credential, such as not having a driver's license, increases an individual's rights. Therefore, we do not expect credentials to correspond to bipolars. Similar arguments may be made for other types of information.

If the policy set includes a mix of permitting and denying policies, even if **Permitted** does not appear in the antecedent of policies, then it seems less likely that the bipolar restriction will hold. For example, consider the policy set $\{p_1, p_2\}$ where p_1 is 'faculty members may chair committees' and p_2 is 'students may not chair committees'. Formally,

$$p_1 = \forall x(\textbf{Faculty}(x) \Rightarrow \textbf{Permitted}(x, \text{chair committees}))$$
$$p_2 = \forall x(\textbf{Student}(x) \Rightarrow \neg\textbf{Permitted}(x, \text{chair committees})).$$

The literal **Permitted**$(x, \text{chair committees})$ is a bipolar in $p_1 \wedge p_2$. Once we allow **Permitted** in the antecedent of policies, things can get even worse. Suppose that we extend the **Permitted** predicate to take a third argument that says who is granting (or denying) permission. Now consider the policy set $\{p_1, p_2, p_3\}$ where p_1 is 'Mom allows Alice to play outside', p_2 is 'Dad allows Alice to play outside', and p_3 is 'if both Mom and Dad allow Alice to do something then the Parents allow it'. Formally,

$$p_1 = \textbf{Permitted}(\text{Alice, play outside, Mom})$$
$$p_2 = \textbf{Permitted}(\text{Alice, play outside, Dad})$$
$$p_3 = \forall x(\textbf{Permitted}(\text{Alice}, x, \text{Mom}) \wedge$$
$$\textbf{Permitted}(\text{Alice}, x, \text{Dad}) \Rightarrow \textbf{Permitted}(\text{Alice}, x, \text{Parents}))$$

There are four bipolar literals in $p_1 \wedge p_2 \wedge p_3$.

4.2 Relaxing the restrictions

In this subsection, we discuss the consequences of relaxing some of the conditions in Theorem 4.1. In particular, we consider the effect of allowing standard environments, as opposed to basic ones, allowing a limited use of equality, allowing variables to appear in policies (and the standard environment) without also appearing as arguments to **Permitted**, and allowing each policy (and each environment fact) to have one bipolar. The bipolar restriction is further relaxed in Section 4.3.

We first consider the equality restriction. It turns out that we can allow equality in the quantifier-free portion of the environment. As a result, we can write statements such as 'Miss Alice Smith = Mrs. Alice Jones' and 'hearing $\neq$ listening'. However, if we allow equality to be used in this way, then we need to generalize the definitions of unification and bipolarity. We say that ℓ and ℓ' are *unifiable relative to a set E of equality statements* if there are variable substitutions σ and σ' such that it follows from E that $\ell\sigma = \ell'\sigma'$. For example, $P(a)$ and $P(b)$ are unifiable relative to $a = b$. Similarly, we can talk about a literal ℓ being *bipolar in formula f relative to E*.

We also can support equality in the antecedents of policies, but we cannot support *inequalities*. For example, we can handle the policy

$$\forall x_1 \forall x_2((x_1 = \text{Spouse}(x_2)) \Rightarrow \textbf{Permitted}(x_1, \text{SpeakFor}(x_2))),$$

but we cannot handle the policy

$$\forall x_1 \forall x_2((x_1 \neq \text{Spouse}(x_2)) \Rightarrow \neg\textbf{Permitted}(x_1, \text{SpeakFor}(x_2))).$$

(The first policy says 'an individual may speak for her spouse'. The second says 'an individual may not speak for someone who is not her spouse'.)

We now consider the variable restriction; first we relax it and then we remove it entirely. Suppose that every literal in every policy has at most one variable that doesn't appear in **Permitted** (which is the case in Examples 2.2 and 3.2) and there are m constants that appear in the environment. Then the increase in complexity is only $O(m|P|\log|E|)$, and the time needed to answer our queries is $O((|E| + m|P|)\log|E|)$. Therefore, our language will not become intractable if we allow any number of variables to violate our original restriction, provided that each literal has only one such variable.

The NP-hardness result of Theorem 3.4(b) suggests that it will not be possible to get such low complexity in general. We can show that if there are at most k variables in any policy that do not appear as arguments to **Permitted**, then the queries in which we are interested can be answered in time $O((|E| + m^k|P|)\log|E|)$. This result is not simply a generalization of the previous one. Our earlier result might apply to a policy set for which k is greater than one. Consider the policy $\forall x_1 \forall x_2 \forall x_3 \forall x_4 (R_1(x_1, x_4) \wedge R_2(x_2, x_4) \wedge R_3(x_3, x_4) \Rightarrow \textbf{Permitted}(x_4, a))$. The condition in our first result is met by this policy, because each literal has only one variable that does not appear in **Permitted**. The second result applies with $k = 3$, because the policy has three variables that do not appear as arguments to **Permitted**, namely x_1, x_2, and x_3.

It is unlikely that these results can be significantly improved, because even with our bipolar restriction, we can show that the general problem is NP-complete. However, we expect that both m and k will be quite small in practice. Therefore, we can still answer queries efficiently in practice.

The following theorem summarizes the discussion thus far:

Theorem 4.2: *Let Φ be a vocabulary that contains **Permitted** (and possibly other predicate, constant, and function symbols). Let $\mathcal{L}_6$ consist of all closed formulas in $\mathcal{L}^{fo}(\Phi)$ of the form $E \wedge P \Rightarrow$ **Permitted**(t, t'), where P is a conjunction of standard policies and both t and t' are closed terms of the appropriate sort, such that*

(a) E is a basic environment with m constants,

(b) no policy in P has an inequality in its antecedent, and

(c) there are no bipolars in P relative to the equality statements in E.

*If there are at most k variables in a single policy that do not appear as arguments to **Permitted**, then we can determine the validity of the formula in time $O((|E|+m^k|P|)\log|E|)$. Moreover, if each literal in each policy has at most one variable that does not appear in **Permitted**, then we can determine the validity of the formula in time $O((|E| + m|P|)\log|E|)$.*

Note that Theorem 4.2 allows equality in the environment E. Also, note that all of the examples in this paper, including Examples 2.2 and 3.2, meet the condition that every literal in every policy has at most one variable that does not appear in **Permitted**. Thus, we can answer our queries about these policies in time $O((|E| + m|P|)\log|E|)$.

We now extend our results to handle standard environments. For the purposes of this discussion, let P be a conjunction of standard policies and let $E_0 \wedge E_1$ be a standard environment in which E_0 is a conjunction of ground literals and E_1 is a conjunction of universal formulas. Since Theorem 4.1 already handles universal formulas, namely policies, we could support standard environments by replacing every reference to P in Theorem 4.1 with a reference to $P \wedge E_1$. In particular, we could replace the bipolar restriction in Theorem 4.1 with the statement 'there are no bipolars in $P \wedge E_1$'. However, if there are no bipolars in $P \wedge E_1$, then it is not hard to show that (as long as $E_0 \wedge E_1$ is consistent) a permission follows from $E_0 \wedge E_1 \wedge P$ iff it follows from $E_0 \wedge P$. In other words, unless we can relax the bipolar restriction, we cannot support interesting universal formulas in the environment. Fortunately, we can relax the bipolar restriction to allow one bipolar per clause. (As we show later, this is probably the best we can do.)

The result is summarized in the following theorem. The two conclusions regarding complexity correspond to the conclusions in Theorem 4.2, except now we must consider the variables that appear in E_1 as well as those that appear in P.

Theorem 4.3: *Let Φ be a vocabulary that contains **Permitted** (and possibly other predicate, constant, and function symbols). Let $\mathcal{L}_7$ consists of all closed formulas f in $\mathcal{L}^{fo}(\Phi)$ of the form $(E_0 \wedge E_1 \wedge P) \Rightarrow$ **Permitted**(t, t'), where $E_0 \wedge E_1$ is a standard environment, E_0 is a conjunction of ground literals, E_1 is a conjunction of universal formulas, P is a conjunction of standard policies, and both t and t' are closed terms of the appropriate sort, such that*

(a) E_0 has m constants,

(b) no conjunct in $E_1 \wedge P$ has an inequality in its antecedent, and

(c) each conjunct in $E_1 \wedge P$ has at most one literal that is bipolar in $E_1 \wedge P$ relative to the equality statements in E_0.

*We can determine the validity of f in time $O(|E_1 \wedge P|\log|E_1 \wedge P| + b|C_l| + T)$, where b is the number of bipolar pairs in f relative to the equality statements in E_0, C_l is the longest conjunct in f, and T is defined as follows. If every literal that appears in a conjunct in $E_1 \wedge P$ has at most one variable that does not appear as an argument to an instance of **Permitted** in that conjunct, then T is $(|E_0| + m(|E_1 \wedge P| + b|C_l|))\log|E_0|$. Otherwise, T is $(|E_0|+m^k(|E_1\wedge P|+b|C_l|))\log|E_0|$, where k is the largest number of variables appearing in a single conjunct that do not also appear as arguments to an instance of **Permitted** in that conjunct.*

Because the environment, by definition, does not contain the **Permitted** predicate, every variable in a conjunct in E_1 is a variable that does not appear as an argument to **Permitted**.

For the rest of this subsection, we discuss why Theorems 4.1, 4.2, and 4.3 are true, and the role of the restrictions on bipolarity and equality.

These theorems are best understood in the context of the *resolution* procedure from theorem proving [30]. Resolution tries to find clauses C_1 and C_2 and a substitution σ under which C_1 and C_2 refer to the same literal with different polarities (one refers to the literal ℓ, the other to $\neg\ell$). If the search is successful, then a new clause, called the *resolvent*, is created by taking the disjunction of $C_1\sigma$ and $C_2\sigma$ after removing the shared literal from each clause. For example, given the clauses $\neg R(y) \vee \neg S(y)$ and $R(f(x)) \vee$ **Permitted**$(g(x), z)$, the resolution procedure substitutes $f(x)$ for y because, under this substitution, the clauses share the literal $R(f(x))$, with different polarities. The resolvent created from these clauses is then

$\neg S(f(x)) \vee$ **Permitted**$(g(x), z)$.[2] Throughout the rest of the paper, we refer to the clauses C_1 and C_2 as the *parents* of the resolvent and we say that we *resolve on* a literal ℓ (or $\neg\ell$) if that is the shared literal used in creating the resolvent. The *closure* of a universal formula f, denoted $R(f)$, is the smallest set of clauses such that $f \subseteq R(f)$ and if r is a resolvent of two clauses that are in $R(f)$, then r is in $R(f)$. A key property of the resolution procedure is the following statement. If no positive literal in f (written in CNF) involves equality, then $R(f \wedge \forall x(x = x))$ contains **false** iff f is not satisfiable. Thus, we can use resolution to check the validity of an existential formula provided that the formula (in CNF) does not refer to an inequality.

The reason for the equality restriction is that the resolution procedure assumes all constants are distinct, regardless of statements to the contrary. For example, consider the following three statements about Bob and Robert.

$f_1 =$ **Permitted**(Bob, play) $\wedge\ \neg$**Permitted**(Robert, play)
$f_2 =$ **Permitted**(Bob, play) $\wedge\ \neg$**Permitted**(Bob, play)
$f_3 =$ **Permitted**(Bob, play) $\wedge\ \neg$**Permitted**(Robert, play) $\wedge$(Bob = Robert)

It is easy to see that $R(f_1)$ is f_1, $R(f_2)$ contains **false**, and $R(f_3)$ is f_3. The resolution procedure does not resolve the clauses in f_1, because it assumes that Bob could be an individual different from Robert. In this case the assumption is correct and the desired property holds: $R(f_1)$ doesn't contain **false** and f_1 is satisfiable. As for f_2, the resolution procedure recognizes that the constant Bob in the first clause refers to the same individual as the constant Bob in the second. Thus, the procedure resolves the two clauses to create the resolvent **false**, which indicates that f_2 is not satisfiable. Now consider f_3. Because of the last clause in f_3, Bob cannot be a different individual than Robert, however the resolution procedure fails to take this into account. Therefore, it does not resolve the first two clauses and $R(f_3)$ does not contain **false**, even though f_3 is unsatisfiable.

If equality occurs only in clauses that are ground literals, then the fix is straightforward. We simply compute, for each constant, the set of constants equal to it according to the equality statements among the ground literals. This partitions the constants into equivalence classes. We then choose a representative element from each equivalence class, and replace each occurrence of a constant by the equivalent representative element. For example, given f_3, we would replace every occurrence of Bob with Robert (or vice-versa), since Bob and Robert are in the same equivalence class. Note that, after the substitution, there are two bipolar literals in f_3 when originally there were none. Since this procedure can add bipolars to the formula and we need to restrict bipolars for tractability, our theorems must refer to the number of bipolars after the substitutions have been made. This is why the theorems refer to the number of bipolars relative to a set of equality statements (if the environment has equality).

[2] Actually, the resolution procedure looks for a particular type of substitution called a *most general unifier* (mgu). This is why, in our example, we substitute $f(x)$ for y, instead of, say substituting $f(a)$ for y and a for x. (See [30] for details.)

We remark that, in general, dealing with equality in the context of resolution is nontrivial; it requires techniques such as *paramodulation* [8]. Our restrictions guarantee that these additional procedures are unnecessary.

The problem with applying resolution is that, in general, the number of clauses in $R(f)$ can be infinite, even if f is a function-free formula with only two clauses.

Example 4.4: Suppose we have two policies; the first is 'Alice may play' and the second is 'for any individuals x_1 and x_2, if x_1 may play and x_2 is x_1's boss, then x_2 may play'. We could write these policies as

$p_1 =$ **Permitted**(Alice, play)
$p_2 = \forall x_1, x_2($**Permitted**$(x_1, \text{play}) \wedge$ **BossOf**$(x_2, x_1) \Rightarrow$ **Permitted**$(x_2, \text{play}))$

It is not hard to see that for any integer n, the closure of $p_1 \wedge p_2$ includes the clause $(\bigvee_{i=1,\ldots,n} \neg$**BossOf**$(x_i, x_{i-1})) \vee \neg$**BossOf**$(x_0, \text{Alice}) \vee$ **Permitted**(x_n, play). ∎

It turns out that the source of the difficulty in this example is the fact that **Permitted**(x_1, play) and **Permitted**(x_2, play) are bipolar literals. If we restrict the number of bipolar literals, the problem does not occur. Further restrictions give us tractability, as the following result shows. Parts c(i) and c(ii) of the proposition are again analogues of the two conclusions in Theorem 4.2.

Proposition 4.5: *Let f be a conjunction of ground literals. Let f' be a formula in CNF with n' bipolar pairs and n clauses such that every clause has at most one instance of a bipolar literal in f' relative to the equality statements in f and no disjunct of the form $(t = t')$, where t and t' are terms.*

- (a) *$R(f')$ has $n + n'$ clauses. Moreover, the resolution procedure runs in time $O(|f'|\log|f'| + n'|C_l|)$, where C_l is the longest clause in f'.*
- (b) *If $R(f') = \{C_1, \ldots, C_n\}$, then $R(f \wedge f') = R(f) \cup (\bigcup_{i \le n} R(C_i \wedge f))$.*
- (c) *Suppose f has m constants and C is a clause in $R(f')$. Let L_C be the set of literals in C that unify with no more than one literal in f relative to the equality statements in f. Let V_C be the set of variables in C that do not appear in any literal in L_C.*
  - (i) *If every literal in C has no more than one variable that is in V_C, then we can determine if $R(f \wedge C)$ contains* **false** *in time $O((|f| + m|C|)\log|f|)$.*

(ii) *If $|V_C| = k$, then we can determine if $R(f \wedge C)$ contains* **false** *in time $O((|f| + m^k|C|)\log|f|)$.*

The reason that Proposition 4.5(a) holds is that we resolve only on literals that are bipolar. It follows that the only resolvents created from the clauses in f' are those created by the bipolar pairs, and only one resolvent is created per pair. Furthermore, these are the only resolvents in the closure, because none of the resolvents created by this process have a bipolar literal. To prove Proposition 4.5(b), we need one more fact (proved in the full paper): For any resolvent r_k with one parent in $R(C_i \wedge f)$ and another in $R(C_j \wedge f)$, there is a resolvent C_k whose parents are C_i and C_j (thus C_k is in $R(f')$) such that r_k is in $R(C_k \wedge f)$. $R(C \wedge f)$ can contain **false** iff one of the following Finally, for Proposition 4.5(c), it is easy to show that $R(C \wedge f)$ contains **false** iff either

(i) $R(f)$ contains **false**, or

(ii there is a variable substitution σ such that, for every disjunct ℓ in $C\sigma$, there is a conjunct of f equivalent to $\neg\ell$ relative to the equality statement in f.

Since f is a conjunction of ground literals, it is satisfiable unless there is a bipolar in f (in which case resolving on the bipolar produces **false**). We can check this in time $O(|f|\log|f|)$, using an appropriate dictionary structure. Clearly, we can check if the second statement holds in time $m^k|C|\log|f|$ by simply trying all possible substitutions. (This observation leads to the result in Proposition 4.5(c)(ii).) Note that if a literal ℓ in C unifies with only one literal in f, relative to the equalities in f, then the substitution σ is essentially determined for the free variables in ℓ. Thus, if the hypotheses of Proposition 4.5(c)(i) hold, then, after making the all the required substitutions, each literal has at most one variable that has not yet been assigned a value. It is not hard to show that, in this case variable can be considered independently of the others. Therefore, we can try all possible assignments in time $O((|f|+m|C|)\log|f|)$.

The proof of Theorem 4.3 follows readily from Proposition 4.5. Consider a formula g in $\mathcal{L}_7$ and formulas f and f' defined as follows

$$\begin{array}{rcl} g & = & (E_0 \wedge E_1 \wedge P) \Rightarrow \textbf{Permitted}(t,t'), \\ f & = & E_0 \wedge \neg\textbf{Permitted}(t,t'), \text{ and} \\ f' & = & E_1 \wedge P. \end{array}$$

Recall that g is valid iff $R(f \wedge f')$ contains **false**. Because every conjunct in $E_1 \wedge P$ has at most one bipolar in f', Proposition 4.5(a) applies, and we can calculate $R(f')$ in time $O(|f'|\log|f'|+n'|C_l|)$ where n' and C_l are as defined in the proposition. By Proposition 4.5(b), we can calculate $R(g)$ by calculating $R(f)$ and $R(f \wedge C_i)$ for every clause C_i in $R(f')$. Finally, by Proposition 4.5(c), we can determine efficiently if any of these sets contain **false**. In particular, if every literal in f' has at most one variable that does not appear as an argument to **Permitted**, then Proposition 4.5(c)(i) applies, because **Permitted** appears in only the policies and the query **Permitted**(t,t') (and, thus appears only once in f).

4.3 Beyond the bipolar restriction

As we have already observed, the bipolar restriction in Theorems 4.1, 4.2, and 4.3 might not hold in practice. In this section, we discuss two situations in which the restriction is unlikely to hold, and what can be done about it. The first is when policies use predicates that are, intuitively, defined in the environment. The second is when the policy set includes both permitting and denying policies (that is, the set has policies with **Permitted** in the conclusion and policies with ¬**Permitted** in the conclusion).

To understand the role of definitions, consider the policy ‘any minor who is intoxicated may go to jail’. Now, suppose that an individual is a minor in New York if she is under twenty-one and she is a minor in Alaska, if she is under eighteen. Also, an individual is intoxicated if she fails a breathalyzer test, can’t touch her nose, or can’t walk straight. Formally,

$$\begin{array}{l} p_1 = \forall x(\textbf{Minor}(x) \wedge \textbf{Intox}(x) \Rightarrow \textbf{Permitted}(x, \text{go to jail})) \\ e_1 = \forall x(\textbf{Under21}(x) \wedge \textbf{InNY}(x) \Rightarrow \textbf{Minor}(x)) \\ e_2 = \forall x(\textbf{Under18}(x) \wedge \textbf{InAK}(x) \Rightarrow \textbf{Minor}(x)) \\ e_3 = \forall x(\text{FailsBreathalyzer}(x) \Rightarrow \textbf{Intox}(x)) \\ e_4 = \forall x(\neg\text{CanTouchNose}(x) \Rightarrow \textbf{Intox}(x)) \\ e_5 = \forall x(\neg\text{CanWalkStraight}(x) \Rightarrow \textbf{Intox}(x)) \end{array}$$

Roughly speaking, e_1 and e_2 define the notion of being a minor, while e_3, e_4, and e_5 define the notion of being intoxicated. These definitions are used in p_1 to regulate who may go to jail. It is easy to see that p_1 has two bipolars in $p_1 \wedge e_1 \wedge \ldots \wedge e_5$, namely **Minor** and **Intox**. Therefore, the bipolar restriction that we rely on for tractability is not met.

Definitions in this spirit arise frequently in the Social Security database. Thus, it is important to be able to handle them. Perhaps the simplest approach is just to rewrite the policy p_1 so as to replace **Minor** and **Intox** by their definitions. If we do this, then p_1 is replaced by the following six policies:

$$\begin{array}{rl} p'_1 = & \forall x(\textbf{Under21}(x) \wedge \textbf{InNY}(x) \wedge \\ & \text{FailsBreathalyzer}(x) \Rightarrow \textbf{Permitted}(x, \text{go to jail})) \\ p'_2 = & \forall x(\textbf{Under21}(x) \wedge \textbf{InNY}(x) \wedge \\ & \neg\text{CanTouchNose}(x) \Rightarrow \textbf{Permitted}(x, \text{go to jail})) \\ p'_3 = & \forall x(\textbf{Under21}(x) \wedge \textbf{InNY}(x) \wedge \\ & \neg\text{CanWalkStraight}(x) \Rightarrow \textbf{Permitted}(x, \text{go to jail})) \end{array}$$

$$p'_4 = \forall x(\textbf{Under18}(x) \wedge \textbf{InAK}(x) \wedge \text{FailsBreathalyzer}(x) \Rightarrow \textbf{Permitted}(x, \text{go to jail}))$$

$$p'_5 = \forall x(\textbf{Under18}(x) \wedge \textbf{InAK}(x) \wedge \neg\text{CanTouchNose}(x) \Rightarrow \textbf{Permitted}(x, \text{go to jail}))$$

$$p'_6 = \forall x(\textbf{Under18}(x) \wedge \textbf{InAK}(x) \wedge \neg\text{CanWalkStraight}(x) \Rightarrow \textbf{Permitted}(x, \text{go to jail})).$$

Notice that there are no bipolars in $p'_1 \wedge \ldots \wedge p'_6$ and the policies permit the same actions as $p_1 \wedge e_1 \wedge \ldots \wedge e_5$. Our translation also illustrates the potential problem with this approach: it can blow up the size of the policy set. Suppose that a policy p has m bipolar literals and that literal i is defined using c_i clauses. Rewriting would result in replacing policy p by $c_1 \times \cdots \times c_m$ policies. Each of the new policies can also be longer than p, although the total length of each one can be no more than $|E_1|$, where E_1 is the first-order part of the environment. Is this so bad? Examples in the social security database suggest that m is typically less than 3. In most cases, a bipolar is defined by only one clause. Thus, replacement does not typically increase the number of policies, although the individual policies are longer. These examples suggest that, in practice, definitions will not significantly reduce the efficiency of these procedures.

We next provide a condition that allows us to support policy sets that have both permitting and denying policies. This task would be easy if we could consider only the permitting policies (ignoring the denying policies) when determining if an action is permitted. Unfortunately, if we do this, then we might not answer queries correctly.

To see why, consider an environment E that says 'Alice is a student' and a policy set $\mathcal{P} = \{p_1, p_2, p_3\}$, where p_1 says 'faculty members may chair committees', p_2 says 'students may not chair committees', and p_3 says 'anyone who is not a faculty member may take naps'. We can write these policies as

$$p_1 = \forall x(\textbf{Faculty}(x) \Rightarrow \textbf{Permitted}(x, \text{chair committees})),$$
$$p_2 = \forall x(\textbf{Student}(x) \Rightarrow \neg\textbf{Permitted}(x, \text{chair committees})),$$
$$p_3 = \forall x(\neg\textbf{Faculty}(x) \Rightarrow \textbf{Permitted}(x, \text{nap})).$$

Clearly, p_1 and p_3 are permitting policies and p_2 is a denying policy. Because p_1 is equivalent to $\forall x(\neg\textbf{Permitted}(x, \text{chair committees}) \Rightarrow \neg\textbf{Faculty}(x))$, p_1 and p_2 together imply that no student is a faculty member. (Intuitively, students cannot be faculty members, because no one can be both permitted and not permitted to chair committees.) Because students are not faculty members, Alice, being a student, is not a faculty member and, by p_3, may take a nap. We cannot determine that Alice may nap if we consider only the permitting policies, because to derive the permission we need the environment fact that is implied by $p_1 \wedge p_2$.

If each fact implied by a permitting and denying policy together were derivable from either the environment or a single policy, then we could separate the permitting policies from the denying policies. Intuitively, this is because the interaction would not provide any information that wasn't already known. To formalize this intuition, note that each implied fact corresponds to a resolvent of a permitting and denying policy. In the previous example, the implied fact that students are not faculty members corresponds to the resolvent of p_1 and p_2, namely $\forall x(\textbf{Faculty}(x) \Rightarrow \neg\textbf{Student}(x))$. Therefore, if every resolvent of a permitting and denying policy is already implied by the environment or a single policy, then we can separate the policies. Continuing our example, we could separate the policies if the environment said that students were not faculty members. A closer analysis shows that, because we are determining permissions and prohibitions, we need to consider only those resolvents that are created by resolving on a literal that involves **Permitted**.

We formalize all of this in the following theorem. But to do so, we need to discuss permitting and denying policies in a bit more detail. Note that a policy such as $\forall x(\textbf{Permitted}(\text{Alice}, a) \Rightarrow \textbf{Permitted}(\text{Bob}, a))$ is logically equivalent to both a permitting policy and a denying policy. (The denying policy is $\forall x(\neg\textbf{Permitted}(\text{Bob}, a) \Rightarrow \neg\textbf{Permitted}(\text{Alice}, a))$.) We say that a policy is *pure* if it is not logically equivalent to both a permitting and a denying policy. Note that policies that do not mention **Permitted** in the antecedent (which is the case for almost all the policies we have collected) are guaranteed to be pure.

Theorem 4.6: *Suppose that E is a standard environment and $\mathcal{P} = \{p_1, \ldots, p_n, d_1, \ldots, d_m\}$ is a set of policies, where $p_1, \ldots, p_n$ are pure permitting policies and $d_1, \ldots, d_m$ are (not necessarily pure) denying policies. If it is the case that for every pure permitting policy $p \in \mathcal{P}$, every (pure or impure) denying policy $d \in \mathcal{P}$, and every resolvent f created by resolving p and d on a literal that involves **Permitted**, either $E \Rightarrow f$ is valid or $q \Rightarrow f$ is valid for some q in $\mathcal{P}$, then $E \wedge p_1 \wedge \ldots \wedge p_n \wedge d_1 \wedge \ldots \wedge d_m \Rightarrow$ **Permitted**(t, t') is valid iff $E \wedge p_1 \wedge \ldots \wedge p_n \Rightarrow$ **Permitted**(t, t') is valid for any terms t and t' of the appropriate sort.*

Of course, a similar result holds for prohibitions.

Given an environment and a set of policies, we can always add clauses to obtain an equivalent environment and policy set that meets the theorem's conditions. Therefore, the question isn't 'how likely are these conditions to be met in practice', but 'how many clauses are we going to have to add so that these conditions are met'. Example 4.4 shows that we may need to add an infinite number of policies to the set. However, for policy sets where **Permitted** appears only in the conclusions of policies, it is easy to see that every resolvent is an environment fact and there is, at most, one resolvent per pair of permitting and denying policies.

So, if the policy set consists of n policies, then we can satisfy the antecedent of Theorem 4.6 by adding at most n^2 clauses to the environment.

Instead of adding these clauses to the environment automatically, it may be better to verify the changes with the policy maker. To see why, recall the two policies 'faculty members may chair committees' and 'students may not chair committees'. We could satisfy the antecedent of Theorem 4.6 by adding the fact 'no student is a faculty member' to the environment. But suppose that there is (or could one day be) a student who is also a faculty member. Then the policy maker may want to revise the policies to take this into account, rather than allowing the environment to (possibly) become inconsistent. In general, we expect that the additional facts needed to satisfy the antecedent of Theorem 4.6 will be ones that either the user would agree should have been there all along or are ones that should not be there and in fact suggest that the policies should be rewritten. By querying policy makers, we help them to write better policies.

Another advantage of querying the policy maker is that the implied facts may remind her of a general fact that should be added to the environment. For example, the policies 'men under 65 may apply for health plan A', 'men who do not smoke may apply for health plan A', and 'women may not apply for health plan A' imply the facts 'men under 65 are not women' and 'men who do not smoke are not women'. Rather than adding both facts to the environment, the policy maker may prefer to add the fact 'men are not women' and in this way simplify the environment.

4.4 Consistency

In this section, we consider the problem of checking consistency. (Recall that an environment E and policy set $P = \{p_1, \dots, p_n\}$ is consistent if $E \wedge p_1 \wedge \dots \wedge p_n$ is satisfiable.) Clearly $E \wedge p_1 \wedge \dots \wedge p_n$ is not consistent iff $E \wedge p_1 \wedge \dots \wedge p_n$ implies both **Permitted**(c_1, c_2) and $\neg$**Permitted**(c_1, c_2), for some arbitrary constants c_1 and c_2. Thus, we can apply our previous techniques to checking consistency. However, we can say even more. If the condition of Theorem 4.6 (or the corresponding condition for determining prohibitions) is met, then we automatically have consistency, provided that E is consistent.

Theorem 4.7: *Suppose that E is a simple environment and $\mathcal{P} = \{p_1, \dots, p_n, d_1, \dots, d_m\}$ is a set of policies such that the antecedent of Theorem 4.6 holds. Then $E \wedge p_1 \wedge \dots \wedge p_n \wedge d_1 \wedge \dots \wedge d_m$ is satisfiable iff E is satisfiable.*

Thus, in addition to making it feasible to check the consequences of policies, our conditions essentially prevent users from writing inconsistent policies. This is a major benefit of adhering to these restrictions!

5 Prototype

We have presented an expressive, tractable logic for reasoning about policies. But how can policy writers and administrators (users), who are not logicians, benefit from such a logic? We believe that an appropriate interface would allow users to state their policies and the relevant facts, as well as to make queries, without writing formulas. Their input could be translated into our fragment of first-order logic and then answers to their queries could be translated back into natural language to produce reasonable answers to the original (pre-translated) questions. We are in the process of building a prototype that allows users to enter information by filling in blanks in English sentences. Although many of the details are still being refined, we have completed a basic interface and a translation from the interface to first-order logic. Due to space constraints, we do not give a complete description of the interface, nor do we provide a formal translation from the fields entered by users to first-order formulas. We are preparing a paper that will discuss this in detail. Here we just present the highlights of our approach.

A user creates policies and states environment facts by filling in blanks in English sentences. For example, the user could record the fact that Alice Smith paid her dues to Bob Jones at 10 AM on May 1, 2002, by selecting the appropriate environment form and filling in the white boxes as shown in Figure 1.

When designing the prototype, we need to decide which English sentences should be supported, where the blanks should go, and what symbols may go in each blank. The first two questions can be answered by analyzing the structure of the policies that we collected and the environment facts on which they rely. Addressing the last question is more interesting. Based on the structure of the sentences, it is easy to decide which blanks should take terms and which should take predicate symbols, but it is less clear what those symbols should be. This choice depends on the application, and for any particular application the appropriate choice may change over time. For example, a library may want a constant symbol for each patron. The set of library patrons, however, is not fixed. To handle this, we allow users to create symbols on the fly, while filling in the sentences. We can, for the most part, infer what type of symbol it is (Subject, predicate, etc.) from its use. The only exception is that the user must help us distinguish constants from variables.

A drawback to having a nonfixed language is that a user may have difficulty remembering precisely which terms have been defined. For example, a policy maker may wonder if a predecessor used the term 'graduate student', 'grads', 'gradStudent', or something else to refer to the graduate population. To minimize this confusion, we provide a directory system for the various sorts

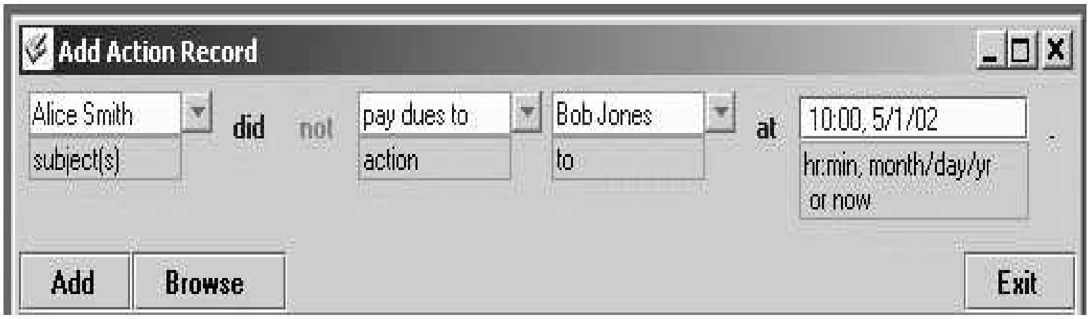

Figure 1. A typical action record.

(e.g., the subject 'grad' may be in the directory 'subjects/university/students'). When a new symbol is defined, the policy maker puts the symbol in the appropriate directory where appropriateness, like the directories themselves, are determined by the users. When searching for the forgotten symbol, the user can consult these directories, which are accessible from the main menu and by clicking on the buttons beside the blanks in the English sentences.

The interface to the system has been designed to support standard environments and policies. In other words, the user cannot fill in the English sentences in such a way that our translation creates either a nonstandard environment or a nonstandard policy. The user, however, can enter a permitting policy and a denying policy that together imply a fact that is not in the environment or a policy that is not implied by one already in the set. Therefore, the antecedent of Theorem 4.6 may not hold. If we are not certain that it holds, then we ask the user if a conflict could occur and, if so, how it should be handled. We then either extend the environment to include the missing fact or we modify the policies to reflect the policy maker's actual intent.

6 Related Work

Our work has been heavily influenced by the work of Halpern, van der Meyden, and Schneider [18]. Their paper discusses key issues that must be addressed when designing a policy language, evaluates various solutions that have been proposed in the literature, and recommends directions for future research. Our design incorporates three of their suggestions. First, Halpern et al. seem to favor first-order logic for handling policies. Second, they advocate defining sorts for principals, actions, and time, which is common in the literature. Third, they suggest having a **Permitted** predicate that takes an individual and an action argument (and perhaps others). (This usage of **Permitted** is much in the spirit of how it is used in modal deontic logic [28, 29].) In essence, we have tailored a logic that was based on their recommendations to serve the expressive and tractability needs of applications.

Many people in the trust management and access control communities have defined tractable policy languages using a fragment of first-order logic. The standard approach (see, for example, Delegation Logic [25], the RT (Role-based Trust-management) framework [27], Binder [11], SD3 [24], and FAF (Flexible Authorization Framework) [23]) is to describe policies in such a way that they can be analyzed using a variant of Datalog, typically either *safe stratified* Datalog [16] or Datalog with *constraints* [31, 33]. Datalog is an efficient well-understood reasoning engine that was originally designed for function-free negation-free Horn clauses. The variants allow some use of functions and negation, while preserving tractability.

There are relatively few policy languages that support functions, but those that do (e.g. [2, 26]) seem to favor a variant of Datalog called Datalog with constraints. By using this variant, many structured resources, such as directories and even time, can be expressed using functions. However, function symbols may not appear in intentional predicates (predicates whose relations are computed by applying Datalog rules, as opposed to being stored in a database). Also, for tractability, additional restrictions are often made. For example, Li and Mitchell [26] do not allow formulas in constraints to have more than one variable.

There are a number of policy languages that support negation. This is typically done using safe, stratified Datalog (e.g. [25], [27], [11], and [24]). Safe, stratified Datalog allows *some* use of negation in the body of rules. The relaxation is not sufficient for all permitting policies of interest. For example, the policy

$$\forall x(\neg \textbf{BadCredit}(x) \Rightarrow \textbf{Permitted}(x, \text{apply for loan}))$$

(anyone without bad credit may apply for a loan) is not supported. More importantly, denying policies cannot be written in safe, stratified Datalog, because the language does not allow negation in the conclusion of rules.

This limitation may not seem to be particularly troublesome. After all, the standard approach, used in relational databases [17], as well as by UNIX [35], SPKI/SDSI

[34, 13, 12], KeyNote [4], and almost all of the Datalog-based approaches, is to assume that everything that is not explicitly permitted is prohibited. However, it is difficult to believe that most policy makers really want to forbid *every* action that they do not explicitly permit. Thus, the assumption may be acceptable in various instances, but it does not capture the policy maker's actual intent. This becomes a problem when different policy makers want to combine their policies. For example, consider a group of libraries that want to merge their policies so that patrons are effected by the same regulations, regardless of which library they visit. When merging the policy sets, we clearly want to detect conflicts (e.g. one library lets minors check-out adult books and another does not). Unfortunately, if a language can state only what is permitted, then this will be impossible. If we put the permitting policies from each library into one large set, then that set will be consistent (it is satisfied in the model that permits everything), regardless of which policies are in the set. Alternatively, we could require that no library permits an action that another forbids (which is what we want to do) under the assumption that every unregulated action is forbidden. It is not hard to see that this approach will always detect a conflict between sets of library policies, unless the policies are essentially identical. For example, if one library allows patrons to access the coat room and another library's policies don't mention a coat room (perhaps because that library doesn't have one) then the policy sets would be flagged as inconsistent, since one allows access and the other forbids it by not explicitly permitting. The bottom line is that it seems unlikely that a policy language will be able to support mergers, unless the language supports both permitting and denying policies. We believe that the issue of merging policies has by and large been ignored, but is an increasingly significant one.

Although we do not know of a Datalog variant that allows negation in the conclusions of rules, there is an extension that allows unrestricted use of negation in the body of rules. Jajodia et al. [23] show that in certain settings this extension, called Datalog with negation, can capture negated conclusions. But this approach to adding negation to Datalog, although it does support both permitting and denying policies, has its own problems. Datalog with negation is tractable because it makes the *closed world assumption*. According to this assumption, if we cannot prove that a positive literal is true, we assume it is false. Unfortunately, the closed world assumption can lead to unintuitive (and probably unintended) results. For example, consider the single policy 'If Alice is not a student, then she may play' and suppose that the reasoning engine can recognize a student only when she presents her ID. If Alice is a student who does not present her ID and the reasoning engine makes the closed world assumption, then the reasoning engine will incorrectly assume that Alice is not a student and, thus, permit her to play.

If a policy language can capture both permitting and denying policies, then conflicts can be detected and resolved in some prescribed way. For example, FAF [23] expects the user to create an overriding policy such as 'if an action is both permitted and forbidden, then it is forbidden'. However, as we have already seen, there are problems with the FAF approach to dealing with conflicts. Similar approaches are taken in [9, 22]. In our language, as long as all pairs of permitting and denying policies satisfy the antecedent of Theorem 4.6, policies cannot be inconsistent, so we do not need overriding policies.

One way in which it may seem that our language is restricted is that we we do not provide explicit support for *groups* and *roles*. Many policy languages talk about groups, where a group is a set of subjects such that if a group has a property, then every member of the group has the property (cf. [1, 23]). In role-based access control models [27, 36, 15, 20], roles are an intermediary between individuals and rights. More specifically, an individual obtains a right by assuming a role that is associated with that right. For example, Alice may need to assume the role of Department Chair in order to obtain the budget.

We do not need to support groups and roles explicitly because we can easily capture both in first-order logic using appropriate predicates. For example, if we want to say that Alice is a member of the faculty and any faculty member may chair committees, then we can represent the group using the predicate **Faculty**. The environment fact is encoded as **Faculty**(Alice); the policy is then

$$\forall x(\textbf{Faculty}(x) \Rightarrow \textbf{Permitted}(x, \text{chair committees})).$$

Similarly, the policy 'Alice, acting as the Department Chair, may sign the budget' can be written as

$$\textbf{Dept. Chair}(\text{Alice}) \Rightarrow \textbf{Permitted}(\text{Alice, sign the budget}).$$

The fact **Dept. Chair**(Alice) would be added to the environment when Alice assumes the role and would be removed when she relinquishes it. Alternatively, we could add a sort *Roles* to our logic along with the predicate **As** (as suggested in [1]), where $\textbf{As}(e, r)$ means that entity e is acting as role r (in other words, e has assumed role r). Continuing our example, 'Alice, acting as the Department Chair, may sign the budget' could be written in the logic as $\textbf{As}(\text{Alice, Dept. Chair}) \Rightarrow \textbf{Permitted}(\text{Alice, sign the budget})$. The second encoding for roles may be more in keeping with the spirit of the role-based model, but we believe that both approaches are reasonable (and our results apply to both choices).

Finally, we should note that the KeyNote system [3] (formerly called PolicyMaker [4]) is more flexible than our approach in that the application can write its policies in a number of different languages. More specifically, the application gives to Keynote programs (which can be written in a variety of programming languages) that determine if a policy applies to a request and a requestor. Because KeyNote essentially views these programs as black boxes, it is quite limited in its ability to reason about policies. As discussed in [5], the system needs to put restrictions on the programs to ensure correct analysis. This is in fact done in [6], but at the price of a substantial reduction in the expressive power of the language.

7 Conclusion

We have considered a fragment of first logic that, based on the policies we collected, is likely to be sufficiently expressive for many applications. We proved that, for typical policies, we could efficiently determine if actions are permitted or prohibited by the policies. Finally, we briefly discussed a prototype that allows non-logicians to benefit from our logic (see [19] for details). As we said earlier, all approaches using first-order logic restrict it in some way to get tractability. The examples that we have been collecting suggest that our language is expressive enough to capture the policies that people want to write. Moreover, we believe that our approach has significant advantages over approaches that cannot express prohibitions, such as approaches based on Datalog, when it comes to merging policy sets.

In terms of future research, we are in the process of using our logic to give semantics to the popular, though ambiguous, XrML rights language [10]. As we said, we are also investigating online databases of policies to check if our language is expressive enough to capture everything that policy writers want to say. This investigation has already led to improvements in our language. For example, it showed us that we need to support definitions. We expect that it will prove useful to find extensions of our logic that remain tractable. One avenue to explore is to consider a hybrid of our approach and Datalog. We plan to pursue this in future work.

Acknowledgements

We would like to thank Carl Lagoze for his advice on the policy needs of digital libraries, Riccardo Pucella for numerous discussions on the material presented here, Thomas Bruce for pointing us to the documents on Social Security, and Moshe Vardi for discussions about the complexity of fragments of first-order logic.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. D. Plotkin. A calculus for access control in distributed systems. *ACM Trans. Prog. Lang. Syst.*, 15(4):706–734, 1993.

[2] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Transactions on Database Systems*, 23(3):231–285, 1998.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The keynote trust management system. http://www.cis.upenn.edu/∼angelos/keynote.html.

[4] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[5] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the PolicyMaker trust management system. In *Proceedings of Financial Cryptography 1998*, pages 254–274, 1998.

[6] M. Blaze, J. Ioannidis, and A. Keromytis. Trust management for IPsec. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, pages 139–151, 2001.

[7] E. Borger, E. Gradel, and Y. Gurevich. *The Classical Decision Problem*. Springer-Verlag, Berlin Heidelberg, 1nd edition, 1997.

[8] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Boston, 1973.

[9] J. Chomicki, J. Lobo, and S. Naqvi. A logic programming approach to conflict resolution in policy management. In *Principles of Knowledge Representation and Reasoning: Proc. Ninth International Conference (KR '00)*, pages 121–132, 2000.

[10] ContentGuard. XrML: The digital rights language for trusted content and services. http://www.xrml.org/, 2001.

[11] J. DeTreville. Binder, a logic-based security language. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 95–103, 2002.

[12] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Simple public key certificate. At http://world.std.com/∼cme/spki.txt, 1999. Internet RFC 2693.

[13] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. At http://www.ietf.org/html.charters/spki-charter.html, 1999. Internet RFC 2693.

[14] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York, 1972.

[15] D. Ferraiolo, J. Barkley, and D. Kuhn. A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 1(2):34–64, 1999.

[16] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, New Jersey, 2002.

[17] P. Griffiths and B. Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems*, 1(3):242–255, 1976.

[18] J. Y. Halpern, R. van der Meyden, and F. Schneider. Logical foundations for trust management. manuscript, 1999.

[19] J. Y. Halpern and V. Weissman. A practical approach to analyzing policy languages. In preparation., 2003.

[20] M. Hitchens and V. Varadharajan. Tower: A language for role based access control. In *Proc. International Workshop of Policies for Distributed Systems and Networks*, pages 88–106, 2001.

[21] R. Iannella. ODRL: The open digital rights language initiative. http://odrl.net/, 2001.

[22] Y. Ioannidis and T. Sellis. Supporting inconsistent rules in database systems. *Journal of Intelligent Information Systems*, 1(3/4):243–270, 1992.

[23] S. Jajodia, P. Samarati, M. L. Sapino, and V. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–260, 2001.

[24] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings 2001 IEEE Symposium on Security and Privacy*, pages 106–115, 2001.

[25] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation Logic: A logic-based approach to distributed authorization. *ACM Transaction on Information and System Security (TISSEC)*, Feb. 2003.

[26] N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages*, Jan. 2003.

[27] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[28] L. McCarty. Permissions and obligations. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pages 287–294, 1983.

[29] J.-J. C. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29(1), 1988.

[30] A. Nerode and R. Shore. *Logic for Applications*. Springer-Verlag, New York, 2nd edition, 1997.

[31] P. R. P. C. Kanellakis, G. M. Kuper. Constraint query languages. *Journal of Computer and System Sciences*, 51(1):26–52, 1995.

[32] S. Payette and T. Staples. The Mellon Fedora project: Digital library architecture meets XML and web services. In *European Conference on Research and Advanced Technology for Digital Libraries*, pages 406–421, 2002.

[33] P. Revesz. Constraint databases: A survey. In *Proceedings of the Workshop on Semantics in Databases*, pages 209–246, 1998.

[34] R. Rivest and B. Lampson. SDSI — a simple distributed security infrastructure. http://theory.lcs.mit.edu/∼cis/sdsi.html, 1996.

[35] A. Robbins. *UNIX in a nutshell: System V Edition*. O'Reilly and Associates, Inc., 1999.

[36] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.