Towards a Policy Language for Humans and Computers

Vicky Weissman[1] and Carl Lagoze[2]

[1] Department of Computer Science, Cornell University Ithaca, NY 14853
vickyw@cs.cornell.edu
[2] Department of Information Science, Cornell University Ithaca, NY 14853
lagoze@cs.cornell.edu

Abstract. A policy is a statement that an action is permitted or forbidden if certain conditions hold. We introduce a language for reasoning about policies called Rosetta. What makes Rosetta different from existing approaches is that its syntax is essentially a fragment of English. The language also has formal semantics, and we can prove whether a permission follows from a set of Rosetta policies in polynomial time. These features make it fairly easy for policy language developers to provide translations between their languages and ours. As a result, policy writers and (human) readers can create and access policies via the interface of their choice; these policies can be translated to Rosetta; and once in Rosetta can be translated to an appropriate language for enforcement.

1 Introduction

A policy describes the conditions under which an action, such as copying, modifying, or distributing digital content, is permitted or forbidden. Digital content providers write policies to govern the use of their works. For example, the ACM (Association for Computing Machinery) has a set of policies that regulate access to their digital library. These policies include statements such as ‘members of the ACM are permitted to access the articles in the library for personal use’ and ‘members may not republish the articles without explicit permission from the ACM’ [Ass04]. Providers and managers of digital content want the task of writing policies to be as easy as possible. In addition, they want their policies to be intelligible to human readers and enforcable by computers.

Existing policy languages fail to meet these requirements. Although the details are deferred to the next section, the key points are as follows. Natural language is fairly intuitive for (human) readers and writers, but policies written in a natural language cannot be readily enforced by computers. XML-based languages, such as ODRL[Ian01] and XrML [MPE04], make progress towards enforceability, but have syntax that is too complex for non-expert users. Logic-based languages, such as the tractable fragments of first-order logic considered in [HW03] and Binder [DeT02], are enforceable, but require policy writers and readers to be logicians and, thus, are not appropriate for many digital content creators and managers.

One solution is to create a new policy management system. The system would have a user interface to facilitate the writing and reading of policies, and would have a formal foundation so that policies written via the interface could be enforced in a provably correct manner. Unfortunately, this approach has two fundamental problems. First, it is unlikely that a single user interface will meet the needs of every community that is involved in right's management. Second, it will be difficult to convince industry to provide support for the language (e.g., build operating systems and media players that enforce the policies). Therefore, we are not proposing a new system. Instead, we are introducing a policy language that provides the glue between the different approaches.

We call our language the Rosetta Policy Language, because it is a gateway from one policy language to the next. The key features of Rosetta are:

- Statements in the language are constructed from a set of templates for English sentences. This makes the language well-suited to be the foundation for a variety of user interfaces, each tailored for specific communities (e.g., librarians, repository managers, content creators).
- The language is unambiguous and tractable. More precisely, we give the language formal semantics by providing a translation from expressions in the syntax to formulas in many-sorted first-order logic. It is this formal foundation that makes the language unambiguous, allows us to *prove* whether a permission or prohibition follows from a set of policies, and lets us show that determining if a permission or prohibition follows from a set of policies takes polynomial time.
- The language can serve as a front-end for existing policy language that are inaccessible to non-experts. Because several of the current policy languages in the formal methods community are fragments of many-sorted first-order logic, our translation amounts to one from our syntax to their languages. Furthermore, the translation from our language to ODRL and XrML is straightforward (although the details are left to the full paper).

In short, we believe that Rosetta's English-like syntax and formal semantics allow policy language developers to give translations between their approaches and ours. If this is done, then people can write and read policies via the interface of their choice and those policies can be translated through Rosetta to an appropriate reasoning engine (to answer questions such as what is and is not permitted) and to an appropriate enforcement mechanism (namely the one supported by the relevant industry).

The rest of this paper is organized as follows. In the next section we place our work in a broader context by briefly reviewing various policy languages. In Section 3, we introduce the basic syntax of our language, and in Section 4, we give formal semantics by translating expressions in the language to formulas in many-sorted first-order logic. Roughly speaking, any expression written in this core language can be translated to any of the popular languages discussed in Section 2. In Section 5 we discuss how the core language can be extended to match some of the unique capabilities of other policy languages, including ODRL, XrML, and the formal approaches, and we consider how these changes effect the language's usability and tractability. We conclude in Section 6.

2 Existing Approaches

Policies are traditionally written in a natural language, such as English. Using a natural language makes the policy writing task relatively easy, because the languages are usually well-known to the writer and are highly expressive. However, the meaning of statements in natural language can be ambiguous. For example, suppose that the ACM has two policies ‘only authors may edit their works in the ACM digital library’ and ‘anyone who is not permitted to edit a work in the ACM digital library may submit a request for edits’. If Alice is the author of some work in the library, then the first policy might allow her to edit that work, or it might simply prevent other people from editing it. Furthermore, if the first policy does not permit authors to edit their works, then the second policy might allow Alice to submit a request for changes to her work (since she is not explicitly permitted to edit it) or it might not (since Alice is not forbidden from editing). Lacking precise meaning, such policies are confusing for human readers and unenforceable by computers.

The ODRL (Open Digital Rights Language) [Ian01] and the XrML (eXtensible rights Markup Language) [MPE04] were designed to capture policies in a way that could be readily manipulated by humans and enforced by computer programs. For many policy writers and readers; however, the languages are unintuitive. If a policy writer is not familiar with XML conventions, which is often the case, then she will find it difficult to write policies in any XML-based language, including ODRL and XrML. This is true despite XML toolkits such as XMLSpy [Alt04] and Oxygen [SS04]. Moreover, even if the policy reader is conversant with XML-conventions, she will probably find policies written in an XML-based language difficult to read, because their meaning is buried in verbose XML syntax.

Not only does ODRL and XrML fail to meet the usability goal, they also fail to meet the enforceability goal. As with natural languages, ODRL is unenforceable because it is ambiguous. For example, a computer program cannot correctly enforce the policy ‘anyone who is not permitted to edit a work may submit a request for edits’ when written in ODRL, because the ambiguity that exists in the English version exists in the ODRL version as well. It also seems unlikely that an efficient enforcement algorithm exists for XrML, because determining if a permission follows from a set of XrML policies is an NP-hard problem [HW04] (i.e., the time needed to answer an arbitrary query is at least exponential in the size of the query).

While ODRL and XrML were under development, the formal methods community was creating its own policy languages. Most of the recent proposals are an extension of Datalog (function-free negation-free horn clauses) [GMUW02]. Perhaps the most popular choice is safe, stratified Datalog, which supports a limited use of negation (see for example, Delegation Logic[LGF03], the RT (Role-based Trust-management) framework [LMW02], Binder [DeT02], and Sd3 [Jim01]). Another option is Datalog with Constraints, which supports a limited use of functions (see for example [LM03] and [BBFS98]). Because these languages have formal semantics, they are unambiguous. Moreover, the languages are tractable; we can prove whether a permission follows from the policies written in one of these languages in polynomial time.

One problem with the Datalog languages is that they do not include policies that forbid actions if certain conditions hold. Instead, these languages assume that any action that is not explicitly permitted is forbidden. To see why this is a problem, suppose that Alice and Bob are collaborating on a multimedia project. Alice wants to show the work in progress to her friends and Bob does not care whether anyone sees the work before it is finished. Intuitively, Alice and Bob’s policies do not contradict each other. However, if the policies are in one of the Datalog languages, then Bob's policy is that no one may view the unfinished work (since he does not explicitly allow the viewing) and, thus, Bob's policies do contradict Alice's. It is not hard to see that any two policy sets will contradict one another, unless they are identical. So, the Datalog approaches are not well-suited to environments in which multiple policy sets regulate the same resource.

To address this deficiency, we considered various fragments of first-order logic [HW03]. We refer to the most expressive tractable fragment discussed in [HW03] as the Lithium language. Lithium includes policies that forbid actions, although some restrictions on negation are needed to get tractability; the language also fully supports the use of functions. In many cases, the policies that people want to discuss can be written in Lithium and we can determine if those policies imply a permission or prohibition in polynomial time (see [HW03] for details).

Although the formal languages meet the enforceability goal and are sufficiently expressive for a number of applications, they are not appropriate for many policy writers or (human) readers. Policy writers are not typically logicians; therefore, expecting them to write policies in a fragment of first-order logic is unrealistic. Similarly, since policy readers are not usually logicians, they are not likely to understand the formulas (even though they are both succinct and unambiguous).

We have created Rosetta as a first step towards solving the usability and enforceability problems seen in the other languages. As discussed in the introduction, it is unlikely that a single interface will meet the usability goal for everyone involved in policy management. However, we believe that Rosetta provides a solid foundation for a family of intuitive interfaces, primarily because it has an English-like syntax and formal semantics. To make languages such as ODRL and XrML enforcable, we need to give them formal semantics (thereby removing ambiguity) and, if necessary, to find tractable fragments. We propose a formal semantics for ODRL in [PW04], and for XrML in [HW04]. (More specifically, we give a translation from policies written in a representative fragment of ODRL and XrML to formulas in many-sorted first-order logic and modal many-sorted first-order logic, respectively.) We have also found a tractable fragment of XrML that is fairly expressive [HW04] and are currently investigating the complexity of ODRL. While this work addresses key problems with both ODRL and XrML, the same problems are likely to exist in the next generation of policy languages (unless, of course, they are created by the formal methods community). Since we cannot rely on logicians retrofitting formal semantics to every new language, we need a way for the language developer to provide semantics without being a logician. This can be done through Rosetta. More specifically, language developers can guarantee that their policies are enforceable by translating them to the English-like (and, thus, fairly intuitive) Rosetta language. Of course, logicians are needed to extend Rosetta to suit more expressive languages; however, we expect that it will be much easier to extend Rosetta as needed than to give formal semantics to every new language directly.

3 Syntax

In this section, we introduce the basic syntax for our policy language, which we extend in Section 5. The language is a fragment of English with two exceptions. First, it supports a notion of labeling that is not commonly seen in English. We discuss the benefits of using labels later in the section. Second, for simplicity we relax the formal definitions of compound nominals and prepositions. Specifically, we include compound nominals such as ‘ACM member’ and ‘library stacks’ in our set of common nouns, and include ‘than’ in our set of prepositions even though it is sometimes used as a conjunction. The language is, admittedly, stilted; however, we believe that it provides a solid foundation for more natural user interfaces. Before presenting the language, we motivate our work with a few examples.

Example 1. Consider the simple sentence ‘Alice’s file is confidential’. While this sentence is not in our language, we can write the similar sentence ‘if f is a file and Alice owns f, then f is confidential’, where f is a label. Notice that the two sentences are not exactly equivalent; the first sentence, ‘Alice’s file is confidential’, is ambiguous if Alice has zero files, or if she has more than one. The sentence in our language does not have this ambiguity because it says that every file that Alice has is confidential. ∎

Example 2. Consider the policy ‘if a professor knows a student, then he is permitted to enter the library stacks’. This policy could mean that a student may enter the library stacks if the student has been vouched for by the professor, or it could mean that professors who socialize with students may enter the stacks. To determine the meaning of the policy, we need to know who is referred to by the pronoun ‘he’. In our language, we avoid this ambiguity by using labels in place of pronouns. Specifically, we associate the common nouns professor and student with the labels p and s respectively. Then, we write the policy as ‘if a professor p knows a student s, then s is permitted to enter the library stacks’ or as ‘if a professor p knows a student s, then p is permitted to enter the library stacks’, depending on the policy’s intended meaning. ∎

Example 3. Consider the policy ‘ACM members may republish articles if they have permission from the ACM.’ We could write this policy in our language as ‘if an ACM member m has p and p is a permission and p is from ACM and a is an article, then m is permitted to republish a.’ ∎

The syntax for our language is described by the grammar in Figure 1. To define the grammar, we use the abbreviations given in Figure 2 as well as the following notation. Elements in parenthesis are optional; s^+ denotes one or more occurences of s; and $[s \mid s']$ means s or s'. The start symbols are SS, CxP, and CxF. We note that basic facts such as ‘Alice is a student’ can be encoded in the language. Although these statements are not policies, they provide information that is often needed to determine the implications of policies. For example, to determine if Alice may download the syllabus, given the policy ‘every student s may download the syllabus’, we need to know if Alice is a student. We also remark that the language given below does not include a number of interesting features, including negation; however, the language is extended in Section 5.

We restrict the syntax so that the articles ‘a’ and ‘an’ appear in only if clauses, and the word ‘every’ never appears in an if clause. As a result, sentences such as ‘Bob borrowed a book’ and ‘if every student s finishes the exam, then the teacher is permitted to post solutions’ are not in Rosetta. It turns out that if we did not make this restriction, then determining if a permission follows from a set of statements in the language would

Figure 1: The Grammar

$SP \rightarrow NP$ is permitted to $V_t\ NP$
$SF \rightarrow NP\ VP$
$SS \rightarrow SP \mid SF$
$CdF \rightarrow SF$ (and $SF)^+$
$CdS \rightarrow SS$ (and $SS)^+$
$CxP \rightarrow$ if $[SS \mid CdS]$, then SP
$CxF \rightarrow$ if $[SF \mid CdF]$, then SF
$NP \rightarrow PN \mid the\ CN \mid (a \mid an)\ CN\ L \mid$
$\quad every\ CN\ L \mid L$
$VP \rightarrow (V_{aux})\, V_t\ NP \mid$
$\quad$ is $[Adj \mid (a \mid an)\, CN \mid (Adj)\, Prep\, NP]$
$CN \rightarrow$ employee | book | collection | $\ldots$
$PN \rightarrow$ Alice | University of Bath | $\ldots$
$L \rightarrow$ x | y | $\ldots$
$V_t \rightarrow$ download | edit | distribute | $\ldots$
$V_{aux} \rightarrow$ is | was | does | have | $\ldots$
$Adj \rightarrow$ trusted | hi-res | corrupt | $\ldots$
$Prep \rightarrow$ to | of | about | with | in | $\ldots$

Figure 2: Abbreviations

SP	simple policy
SF	simple fact
SS	simple sentence
CdF	compound fact
CdS	compound sentence
CxP	complex policy
CxF	complex fact
NP	noun phrase
VP	verb phrase
CN	common noun
PN	proper noun
L	label
V_t	transitive verb
V_{aux}	auxiliary verb
Art	article
Adj	adjective
$Prep$	preposition

be an undecidable problem. We discuss the technical details and their implications in the full paper. Also in the full paper, we extend the syntax so that common nouns do not need to be followed by labels. For example, in the full paper the sentence ‘if every student finishes the exam, then the teacher is permitted to post solutions’ is in the language, as is ‘if a teacher vouches for a student s then s is permitted to enter the library stacks.’ A preprocessing step adds a label after ‘every student’ and ‘a teacher’ before the translation (as described here) is done.

In practice, the set of terminal symbols (e.g., the adjectives, common nouns, and transitive verbs) will depend on the application. For example, the terminal symbols defined in a digital library application might include a proper noun for each employee, the common nouns ‘adult’ and ‘child’, the adjective ‘hearing impaired’, and the transitive verb ‘download’. The symbols could be defined by the system before any policies are written; however, we expect that some of the terms will be imported from other applications or languages (XrML and ODRL both define a set of terms that are appropriate for various policies) and others will be created by policy writers ‘as they go’. As an aside, the interface designer does not have to create a text-based application to use Rosetta; she simply has to provide a translation from the user’s input (in whatever form it is given) to statements in our language.

In this section we have presented a simple grammar that is sufficiently expressive to capture a number of policies that are of practical interest. In Section 5, we consider various extensions to increase the language’s expressivity. First, however, we give formal semantics to the language defined thus far.

4 Semantics

In this section we give a translation from statements in the grammar to formulas in many-sorted first-order logic. For the rest of this discussion, we assume knowledge of first-order logic at the level of Enderton [End72]. More specifically, we assume familiarity 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 assume that the application provides a set $\mathit{properNouns}$ of proper nouns, a set $\mathit{commonNouns}$ of common nouns, a set $\mathit{adjectives}$ of adjectives, a set $\mathit{prepositions}$ of prepositions, a set $\mathit{transitiveVerbs}$ of transitive verbs, and a set $\mathit{auxilliaryVerbs}$ of auxiliary verbs. In addition, we define the set $\mathit{verbs} = \{\mathit{transitiveVerbs}\} \cup \{v_{aux}v_t \mid v_{aux} \in \mathit{auxilliaryVerbs}$ and $v_t \in \mathit{transitiveVerbs}\}$. The vocabulary includes two sorts $\mathit{ProperNouns}$ (e.g., Alice, University of Bath) and $\mathit{Actions}$ (e.g., downloading 'Finding Nemo', editing a budget report). The vocabulary also includes the following symbols:

– a constant pn of sort $\mathit{ProperNouns}$ for each $\mathit{pn} \in \mathit{properNouns}$;
– a constant theCn of sort $\mathit{ProperNouns}$ and a unary predicate Cn that takes an argument of sort $\mathit{ProperNouns}$ for each $\mathit{Cn} \in \mathit{commonNouns}$;
– a unary predicate Adj that takes an argument of sort $\mathit{ProperNouns}$ for each adjective $\mathit{Adj} \in \mathit{adjectives}$;
– a binary predicate Prep that takes two arguments of sort $\mathit{ProperNouns}$ for each preposition $\mathit{Prep} \in \mathit{prepositions}$;
– a binary predicate $\mathit{AdjPrep}$ that takes two arguments of sort $\mathit{ProperNouns}$ for each pair $(\mathit{Adj}, \mathit{Prep})$, where $\mathit{Adj} \in \mathit{adjectives}$ and $\mathit{Prep} \in \mathit{prepositions}$;
– a binary predicate that takes two arguments of sort $\mathit{ProperNouns}$ and a unary function with signature $\mathit{ProperNouns} \longrightarrow \mathit{Actions}$ for each $v \in \mathit{verbs}$; and
– a binary predicate **Permitted** that takes arguments of sort $\mathit{ProperNouns}$ and $\mathit{Actions}$.

We remark that a common noun preceeded by the article 'the' refers to a specific object in much the same way that a proper noun does. For this reason, we associate each common noun in the syntax with a proper one in the translated language. We also associate each common noun with a predicate. Intuitively, $\mathit{Cn}(n)$ means the proper noun n is a Cn. For example, $\mathbf{Book}(\text{'}\mathit{Moby\ Dick}')$ means that 'Moby Dick' is a book. The other predicates have a similar meaning. It is worth noting that if verb is a predicate, then $\mathit{verb}(n_1, n_2)$ means n_1 did verb to n_2. On the other hand, if verb is a function, then $\mathit{verb}(n_1, n_2)$ refers to the action of n_1 doing verb to n_2. For example, $\mathbf{Edits}(\mathit{Alice}, \mathit{the\ report})$ is the statement 'Alice edits the report' if $\mathbf{Edits}$ is a predicate, and it is the action of Alice editing the report if $\mathbf{Edits}$ is a function. Also, **Permitted**(n, a) means n is permitted to do a. For example, **Permitted**$(\mathit{Alice}, \mathit{edits}(\mathit{Alice}, \mathit{the\ report}))$ means Alice is permitted to edit the report.

The translation is given below.

– For every simple sentence, complex policy, and complex fact s in the language $[\![s]\!]^T = \forall x_1, \ldots, \forall x_n (\bigwedge_{(c,x) \in C_s} c(x) \Rightarrow ([\![s_L]\!]^I))$, where $x_1, \ldots, x_n$ are the labels in s and $C_s = \{(c, x) \mid$ the common noun c and the label x are in the same noun phrase in $s\}$.

- If $s =$ ‘ if S, then T' is a complex policy or a complex fact, then $[\![s]\!]^I = [\![S]\!]^I \Rightarrow [\![T]\!]^I$.
- If $s = S_1$ and ... and S_n is a compound sentence, then $[\![s]\!]^I = [\![S_1]\!]^I \wedge \ldots \wedge [\![S_n]\!]^I$.
- $[\![NP_1 \textit{ is permitted to } V_t\ NP_2]\!]^I =$ **Permitted**$([\![NP_1]\!]^O, [\![V_t]\!]^F([\![NP_2]\!]^O))$.
- $[\![NP_1\ (V_{aux})\ V_t\ NP_2]\!]^I = [\![V_{aux}\, V_t]\!]^P([\![NP_1]\!]^O, [\![NP_2]\!]^O)$.
- $[\![NP \textit{ is } Adj]\!]^I = Adj([\![NP]\!]^O)$.
- $[\![NP \textit{ is } (Art) CN]\!]^I = CN([\![NP]\!]^O)$.
- $[\![NP_1 \textit{ is } Prep\ NP_2]\!]^I = Prep([\![NP_1]\!]^O, [\![NP_2]\!]^O)$.
- $[\![NP_1 \textit{ is } Adj\ Prep\ NP_2]\!]^I = AdjPrep([\![NP_1]\!]^O, [\![NP_2]\!]^O)$, where $AdjPrep$ is the predicate associated with the pair $(Adj, Prep)$.
- $[\![PN]\!]^O = PN$, $[\![the\ CN]\!]^O = theCN$, which is the constant associated with CN, $[\![(a \mid an) CN\ L]\!]^O = L$, $[\![every\ CN\ L]\!]^O = L$, $[\![L]\!]^O = L$, $[\![verb]\!]^P$ is the predicate associated with $verb$, and $[\![verb]\!]^F$ is the function associated with $verb$.

To illustrate how the translation works, we revisit each of the examples in Section 3. For ease of exposition, we change the fonts and capitalization to match standard conventions For example, the common noun boy is associated with the predicate $\mathbf{Boy}$ and the constant $theBoy$.

Example 4. The translation of ‘if f is a file and Alice owns f, then f is confidential’ is $\forall f(\mathbf{File}(f) \wedge \mathbf{Owns}(Alice, f) \Rightarrow \mathbf{Confidential}(f))$. ∎

Example 5. The translation of the complex policy ‘if a professor p knows a student s, then s is permitted to enter the library stacks’ is $\forall p \forall s(\mathbf{Professor}(p) \wedge \mathbf{Student}(s) \Rightarrow (\mathbf{Knows}(p, s) \Rightarrow$ **Permitted**$(s, \mathbf{Enter}(the\ library\ stacks))))$, which is logically equivalent to $\forall p \forall s(\mathbf{Professor}(p) \wedge \mathbf{Student}(s) \wedge \mathbf{Knows}(p, s) \Rightarrow$ **Permitted**$(s, \mathbf{Enter}(the\ library\ stacks)))$. ∎

Example 6. The translation of ‘if an ACM member m has p and p is a permission and p is from ACM and a is an article, then m is permitted to republish a’ is $\forall m \forall p \forall a(\mathbf{ACMmember}(m) \Rightarrow (\mathbf{Has}(m, p) \wedge \mathbf{Permission}(p) \wedge \mathbf{From}(p, ACM) \wedge \mathbf{Article}(a) \Rightarrow$ **Permitted**$(p, republish(a))))$, which is logically equivalent to $\forall m \forall p \forall a(\mathbf{ACMmember}(m) \wedge \mathbf{Has}(m, p) \wedge \mathbf{Permission}(p) \wedge \mathbf{From}(p, ACM) \wedge \mathbf{Article}(a) \Rightarrow$ **Permitted**$(p, republish(a)))$. ∎

We now formally define when a permission follows from a set of statements written in Rosetta, where a permission is a simple label-free policy, such as ‘Alice is permitted to enter the library stacks’.

Definition 1. *Let $\{s_1, \ldots, s_n\}$ be a set of simple sentences, complex facts, and complex policies. A permission p* follows from *$\{s_1, \ldots, s_n\}$ iff the formula $[\![s_1]\!]^T \wedge \ldots \wedge [\![s_n]\!]^T \Rightarrow [\![p]\!]^T$ is valid.* ∎

Theorem 1. *Let $\mathcal{L}_0$ be the set of formulas of the form $[\![s_1]\!]^T \wedge \ldots \wedge [\![s_n]\!]^T \Rightarrow [\![p]\!]^T$, where each s_i is a simple sentence, complex sentence, or complex policy, and p is a permission. The validity problem for $\mathcal{L}_0$ is decidable in polynomial time.*

This result is immediate from the fact that every statement in the language translates to a formula that is essentially in Datalog. (Datalog does not include functions; however, the function symbols in our language cannot be nested. Including function symbols in this way does not effect tractability.) In fact, every statement in our language translates to a formula in Lithium and to an XrML policy (called a license in the XrML literature). Also, every policy in our language translates to a policy in ODRL. (ODRL does not include simple statements or complex facts.) This is not surprising; the language presented thus far arguably has the core features of any policy language. Thus, it is likely that every Rosetta policy can be translated to every language of interest. Of course, a policy that can be translated to a particular language of interest might not be expressible in Rosetta. Therefore, we consider various extensions to Rosetta in the next section.

5 Extensions

In this section, we consider key features of ODRL, XrML, Lithium, and the Datalog approaches that are not in our language, and we discuss the consequences of adding them.

5.1 Simple Extensions

There are a number of straightforward ways in which we can extend the language.

– Consider the sentence ‘Bob is department chair from August, 1, 2002 to July 31, 2005’. Although this statement is not in our language, it is easy to modify our approach to include it. To do this, we extend the definition of verb phrases in the grammar to include strings of the form $is\ CN(Prep\,NP)^+$; we add an $(n+1)ary$ predicate $CN\,Prep_1 \ldots Prep_n$ to the vocabulary for each sequence of a common noun and n prepositional symbols; and we add the following to the translation:

$$[\![NP_0\ is\ CN\,Prep_1 NP_1 \ldots Prep_n NP_n]\!]^I = \mathbf{CnPreps}([\![NP_0]\!]^O, \ldots, [\![NP_n]\!]^O),$$

where $\mathbf{CnPreps}$ is the predicate associated with the sequence $CN, Prep_1, \ldots, Prep_n$. We remark that, as expected, $[\![NP_0\ is\ CN\,Prep_1 NP_1 \ldots Prep_n NP_n]\!]^I = [\![NP_0\ is\ CN]\!]^I$ if $n = 0$ (i.e., if there are no prepositions following the common noun). Also, the sentence ‘Bob is department chair from August, 1, 2002 to July 31, 2005’ is in the extended language, and translates to

$$\mathbf{DepartmentChairFromTo}(Bob, August, 1, 2002, July31, 2005).$$

We can further extend the language to support an even wider range of statements by allowing sequences of prepositions to appear in noun phrases, at the end of every verb phrase, etc. These modifications are straightforward. Moreover, the additional predicates do not effect the language’s tractability; we can still determine if a permission follows from a set of sentences in polynomial time. The only problem is that the translation does not attach any intrinsic meaning to words such as ‘from’ and ‘to’. Therefore, it is up to the policy creator to include statements such as ‘if x is department chair from t_i to t_f and t is greater than t_i and t is less than t_f then x is department chair at time t’. We discuss this in the full paper.

– ODRL, among other languages, supports a notion of action sequences. For example, Alice might be permitted to do the action sequence 'play 'Finding Nemo' and then pay five dollars'. We could extend our language to capture this idea by allowing verb phrases to have the form $(V_{aux})\, V_t\ NP(\ and\ then\ (V_{aux})\, V_t\ NP)^+$. The translation is analogous to our treatment for sequences of prepositional phrases; the extension does not effect tractability.
– Instead of having the sort $ProperNouns$, policy languages typically have a sort for principals (i.e., agents such as Alice) and resources (i.e., items such as the book 'Moby Dick'). Some languages have additional sorts for times, roles, and other useful categories. It is not difficult to mimic their approaches. To illustrate how this could be done, suppose that we wanted to replace our sort $ProperNouns$ with two sorts $Princ$ (for principals) and $Rsrc$ (for resources). To do this, we would change the grammar so that $PN \rightarrow PN_P \mid PN_R$, where PN_P are proper nouns that are principals (e.g., Alice) and PN_R are proper nouns that are resources (e.g. 'Moby Dick'). We would need to make a similar split between common nouns, labels, adjectives, and prepositions; otherwise we would not be able to translate these to predicates that took arguments of the appropriate sort and, in the case of labels, to variables of the appropriate sort. In short, we can adapt our language to accommodate a variety of sorts in place of $ProperNouns$, but the result will be a language that is larger and, thus, less easy to use.
  We believe that a better approach is to define common nouns such as 'principal' and 'resource' in our (unaltered) language. Then instead of defining an entity to be one sort or the other, we could make the same distinction, more or less, by adding sentences such as 'Alice is a principal' and "Moby Dick' is a resource' to our set of statements. This approach does not have quite the same effect as multiple sorts because a proper noun can be described by several adjectives. As a result, a proper noun can be both a principal and a resource, or neither. Nevertheless, we feel that this approach strikes a better balance between expressivity and usability.
– ODRL classifies simple sentences based on whether or not a user has the ability to make the sentence true. For example, Alice can make the sentence 'Alice paid five dollars' true by paying five dollars, but if she is five years old, then there is little that she can do to make the sentence 'Alice is an adult' true. We can add this distinction to our language by splitting simple sentences into those that are within the user's control and those that are not. (Essentially, this is the same technique that we use to replace the sort $ProperNouns$ with the sorts $Princ$ and $Rsrc$.)
– XrML and ODRL allow an entity (principal, resource, etc.) to be a set of entities. We capture these groups in our language, by assuming that each one corresponds to a proper noun in $properNouns$. The relationship between a group and its members can be captured as policies and facts in the extended language. For example, to say that a group knows a password if a member of the group knows it, we could include the following complex fact in the set of statements: 'if a principal p knows a password w and p is a member of a group g, then g knows w'.
– ODRL supports statements such as 'at least one of the following policies hold: $p_1, \ldots, p_n$', 'exactly one of the following policies hold: $p_1, \ldots, p_n$', and 'all of the following policies hold: $p_1, \ldots, p_n$'. The last statement can be captured by simply including p_1 through p_n in the set of statements. To capture the other statements,

we need to extend our language to support negation. To see this, notice that the statement 'at least one of the following policies hold: p_1, p_2' means 'if p_1 does not hold, then p_2 holds and if p_2 does not hold, then p_1 holds'. Similarly,'exactly one of the following policies hold: p_1, p_2' means 'if p_1 does not hold, then p_2 holds, if p_2 does not hold, then p_1 holds, if p_1 holds, then p_2 does not hold, and if p_2 holds, then p_1 does not hold.' Adding negation to our language is not straightforward. We discuss it in some detail below.

5.2 Negation

Our language does not include sentences such as 'if Alice is not on disciplinary probation, then she is permitted to join the swim team' and 'Alice is not permitted to impersonate the professor'. It is easy to extend the language to include negation by replaying the definition of simple policies and verb phrases given in Section 3 to be, respectively,

$$\begin{aligned}&SP \rightarrow NP \text{ is permitted to } V_t\ NP \mid NP \text{ is not permitted to } V_t\ NP\\&VP \rightarrow (V_{aux})(\text{not})\, V_t\ NP \mid \text{is(not) } S,\end{aligned}$$

where $S \rightarrow [Adj \mid (a \mid an)CN \mid (Adj)Prep NP]$. Accordingly, the translation could be extended to include the following definitions:

$$\begin{aligned}&[\![NP_1 \text{ is not permitted to } V_t\ NP_2]\!]^I = \neg[\![NP_1 \text{ is permitted to } V_t\ NP_2]\!]^I\\&[\![(V_{aux}) \text{ not } V_t\ NP]\!]^I = \neg[\![(V_{aux})\, V_t\ NP]\!]^I\\&[\![\text{is not } S]\!]^I = \neg[\![\text{is } S]\!]^I,\end{aligned}$$

where S is again $[Adj \mid (a \mid an)CN \mid (Adj)Prep NP]$.

Unfortunately, this solution leads to a language that is decidable, but not tractable.

Theorem 2. *Let $\mathcal{L}_1$ be the set of formulas of the form*

$$[\![s_1]\!]^T \wedge \ldots \wedge [\![s_n]\!]^T \Rightarrow [\![p]\!]^T,$$

where $\{s_1, \ldots, s_n\}$ is a set of statements (simple sentences, complex facts, and complex policies) and p is a permission in the language given in Section 3 extended to include negation. The validity problem for $\mathcal{L}_1$ is decidable; it is NP-hard.

The main reason for the NP-hardness result is that sentences can combine to imply new sentences without forming a chain. For example, consider the policies 'an employee e is permitted to enter the library stacks' and 'a student s is not permitted to enter the library stacks'. Together, these policies imply that employees are not students, because no one can be both permitted and not permitted to enter the stacks. Similarly, the policies 'if s is a good student, then s is permitted to watch 'Finding Nemo" and 'if p is not a student, then p is permitted to watch 'Finding Nemo" together imply that anyone who is good may watch the movie (since both a good student and a good non-student have permission). Determining the consequences of all these interactions is what leads to intractability.

By restricting the language appropriately, we can limit the ways in which statements interact and, thus, we can obtain a tractable fragment of the language that is still quite expressive. In [HW03] we give precise conditions under which a first-order policy language is tractable; these conditions readily apply to the language given here. We can restrict the language beyond what is strictly needed for tractability to create one that is still reasonably expressive and is also fairly easy to explain to users. (Details are given in the full paper.) In addition, if the set of statements include only simple facts and complex policies of the form ' if $[SF \mid CdF]$, then SP', then it is likely that we can determine if the statements imply a permission in a reasonable period of time. (See [HW03] for the empirical argument.)

Alternatively, we could tailor our support for negation according to how it is used in the Datalog languages or in ODRL. (XrML does not support negation; so, we cannot base our work on theirs.) However, as mentioned in Section 2, the Datalog approaches do not support negation in simple sentences or in the then clauses of complex policies or complex facts. Therefore, the languages might not support enough negation to be useful. Also, it is not clear how we could explain when negation could be used in the if clauses of complex sentences (the only place in which negation can appear), even if we were willing to restrict the language to make the task easier. As for negation in ODRL, we do believe that we could explain the ODRL restrictions on negation to a general audience. However, complexity results for ODRL are not available yet; so, we cannot rely on previous work to ensure that our language, extended to handle negation in the ODRL way, is tractable.

6 Conclusion

In this paper, we have introduced Rosetta, a policy language that is well-suited to be both the back-end for user interfaces and the front-end for policy languages that are otherwise inaccessible to non-experts. In the near future, we hope to work with our colleagues in human-computer interaction to design prototypes that exploit Rosetta's capabilities.

Acknowledgments

We would like to thank Joseph Halpern and Riccardo Pucella for giving helpful comments on earlier drafts. Authors supported in part by NSF under grants CTC-0208535 and IIS-9817416, 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.

References

[Alt04] Altova. Xmlspy. http://www.xmlspy.com/products_ide.html, 2004.

[Ass04] Association for Computing Machinery. The ACM guide: Terms of usage. at `http://portal.acm.org/info/usage.cfm`, 2004.

[BBFS98] 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.

[DeT02] J. DeTreville. Binder, a logic-based security language. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 95–103, 2002.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York, 1972.

[GMUW02] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, New Jersey, 2002.

[HW03] Joseph Halpern and Vicky Weissman. Using first-order logic to reason about policies. In *Proc. 16th IEEE Computer Security Foundations Workshop*, pages 187–201, 2003.

[HW04] Joseph Halpern and Vicky Weissman. A formal foundation for XrML. In *Proc. 17th IEEE Computer Security Foundations Workshop*, 2004.

[Ian01] R. Iannella. ODRL: The open digital rights language initiative. http://odrl.net/, 2001.

[Jim01] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings 2001 IEEE Symposium on Security and Privacy*, pages 106–115, 2001.

[LGF03] 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)*, February 2003. To appear.

[LM03] 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*, January 2003. To appear.

[LMW02] 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.

[MPE04] MPEG. Information technology—Multimedia framework (MPEG-21) – Part 5: Rights expression language (ISO/IEC 21000-5:2004). http://www.iso.ch/iso/en/, 2004.

[PW04] R. Pucella and V. Weissman. A formal foundation for ODRL rights. In *Workshop on Issues in the Theory of Security (WITS)*, 2004.

[SS04] Ltd SyncRo Soft. Oxygen. http://www.oxygenxml.com/javawebstart, 2004.